Online Social Networks From A Malicious Perspective: Novel Attack Techniques and Defense Mechanisms

Iasonas Polakis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Computer Science
in the Graduate Division
of the University of Crete

February 2014
Heraklion, Greece

University of Crete
Computer Science Department

Online Social Networks from a Malicious Perspective: Novel Attack Techniques and Defense Mechanisms

A dissertation submitted by: Iasonas Polakis
in partial fulfillment of the requirements for the
degree of Doctor of Philosophy in Computer Science
in the Graduate Division of the University of Crete

The dissertation of Iasonas Polakis is approved by:

Committee:

Evangelos P. Markatos
Professor, Thesis Supervisor

Dimitris Nikolopoulos
Professor

Sotiris Ioannidis
Principal Researcher

Maria Papadopouli
Assistant Professor

Angelos Keromytis
Associate Professor

Stefano Zanero
Assistant Professor

Polyvios Pratikakis
Researcher

Departmental:

Panos Trahanias
Professor, University of Crete — Chairman of the Department

Heraklion, February 2014

Abstract

Social networking services have become the most popular digital services, occupying the majority of the time users spend online. These services have greatly evolved from the first generation of social networks, and offer an expansive set of functionality ranging from user interaction and content sharing, to online gaming and single sign-on services. These services have inadvertently and irrevocably affected the World Wide Web, and forever altered the notion of privacy in the digital era. A natural consequence of their popularity was to also draw the attention of the Internet miscreants that target users for profit.

The vast amounts of personal information and interests that users divulge in these services, along with the high amount of trust users implicitly show to communication received within such networks, has rendered online social networks the ideal springboard for deploying highly-profitable personalized attacks. Attacks in social networks can build upon the expertise of more traditional attack vectors (e.g., email spam) and also incorporate novel techniques for creating complex and intricate attacks. The ever-evolving nature of these networks and the continuous incorporation of novel functionality introduces new design vulnerabilities which can be exploited by adversaries.

Security research in social networks mandates that researchers assume the role of the adversary when exploring the security aspects of these services. Their proprietary nature restricts their deployment in the controlled environment of a laboratory, and may require a black-box testing approach as their internal mechanisms are often unknown. As such, researchers must interact with the actual services and their users. Only then will they be able to "anticipate" techniques that adversaries may employ in the future, and develop effective defense mechanisms that will hinder the actual attacks.

The dissertation demonstrates that by misusing functionality found in various online services and social networks, one can build and deploy effective novel attacks. The results of the practical experiments reveal the vulnerable design of existing defense mechanisms employed by social networks and their inability to protect their assets from adversaries. The characteristics of the attack techniques and the outcome of the experiments guide the design and implementation of new defense mechanisms.

Specifically, we identify the following resources as the "assets" of social networking services, which should be protected against adversaries: (i) user information, (ii) user accounts and (iii) user actions. We assume the role of the attacker and deploy attacks that bypass any mechanisms (if any) designed to protect each type of asset. First, we explore various techniques for harvesting and correlating (personal) user information that can be used for crafting personalized attacks. Next, we demonstrate the effectiveness of automated attacks again photo-based authentication mechanisms designed to hinder adversaries from compromising user accounts. Finally, we conduct extensive experiments to explore the defense mechanisms deployed by social networks to detect and remove actions by malicious users in regards to location-based functionality. In each case, based on the insight gained from the experiments we design mechanisms for mitigating or (if possible) preventing these novel attacks.

Supervisor: Professor Evangelos P. Markatos

Acknowledgments

I would like to thank all the people that have influenced me during this unexpected journey.

First of all, my advisor, Professor Evangelos Markatos, for his continuous support in my work and studies, for giving me the chances he did and, most importantly, for allowing me the freedom to tread my own path. Next, Sotiris Ioannidis for being the voice of dissent, trusting me, and pushing me towards my goals. Also, the remaining members of my committee, Angelos Keromytis, Stefano Zanero, Polyvios Pratikakis, Dimitris Nikolopoulos and Maria Papadopouli, for the valuable comments and questions during my defense.
I want to express my deepest gratitude to the incredible six: Spiros Antonatos, Michalis Polychronakis, Demetres Antoniades, Manos Athanatos, Elias Athanasopoulos and Christos Papachristos, for their help and guidance, their friendship, the never-ending (but always entertaining) debates, the drinking, and for making the lab a fun place.

I would also like to thank all the past and present members of the Distributed Computing Systems Lab for their friendship and contributing (in one way or the other) in the completion of this dissertation: Stamatis Volanis, Thanasis Petsas, Aris Tzermias, Giorgos Vasiliadis, Antonis Papapadogiannakis, Manolis Stamatogiannakis, Antonis Krithinakis, Lazaros Koromilas, Panagiotis Papadopoulos, Michalis Diamantaris, Panagiotis Ilia, Symeon Meichanatzoglou, and anyone else I might be forgetting...

My warmest regards to my Italian friend, Fede, for his help in completing this work, his humour, and for going out of his way to welcome me in his home. It hasn't gone unnoticed.

I would also like to thank all my friends for their support, love, tolerance, and all the great moments that we have shared...The list is too long to expand...You know who you are...

I would like dedicate this work to my father George, my mother Karen, and my sister Maria, for their endless love and support throughout the years. For accepting me as I am, allowing me to find my own voice, and embracing me in all my strangeness. I wouldn't have made it this far without you.

Finally, Eliza (TSI-TSI), words can't adequately describe my luck for having met you...
I thank you for many things but, most of all, for making everything worthwhile...

Περίληψη

Οι υπηρεσίες κοινωνικής δικτύωσης αποτελούν τις πιο δημοφιλείς ψηφιακές υπηρεσίες, καταλαμβάνοντας την πλειοψηφία του χρόνου που ξοδεύουν οι χρήστες στο Διαδίκτυο. Αυτές οι υπηρεσίες έχουν εξελιχθεί σημαντικά απο την μορφή που κατείχε η πρώτη γενιά τους, και προσφέρουν μια εκτενής συλλογή λειτουργιών όπως αλληλεπίδραση μεταξύ χρηστών, ανταλλαγή περιεχομένου, διαδικτυακά παιχνίδια και υπηρεσίες καθολικής σύνδεσης (single sign on). Αυτές οι υπηρεσίες έχουν πολύ σημαντική επίδραση στο Διαδίκτυο, και έχουν μεταβάλλει αμετάκλητα την έννοια της ιδιωτικότητας στην ψηφιακή εποχή. Ένα φυσικό επακόλουθο της δημοτικότητας τους είναι και η στοχοποίηση τους απο κακόβουλους χρήστες με σκοπό το κέρδος.

Η τεράστια συλλογή προσωπικών δεδομένων και ενδιαφερόντων των χρηστών που έχουν συλλέξει αυτές οι υπηρεσίες, καθώς και η εμπιστοσύνη που δείχνουν οι χρήστες στα μηνύματα που λαμβάνουν απο άλλους χρήστες αυτών των υπηρεσιών, καταστούν τις υπηρεσίες κοικωνικής δικτύωσης ιδανικό εφαλτήριο για την μετάδοση κερδοφόρων εξατομικευμένων επιθέσεων. Οι επιθέσεις σε αυτά τα δίκτυα μπορούν να βασιστούν στην πραγματογνωμοσύνη επιθέσεων απο πιο.παραδοσιακά μέσα (π.χ., spam στο ηλεκτρονικό ταχυδρομείο), και να ενσωματώσουν νέες τεχνικές για την δημιουργία σύνθετων και πολύπλοκων επιθέσεων. Η διαρκής εξέλιξη αυτών των υπηρεσιών και η συνεχής ενσωμάτωση νέων λειτουργιών εισάγει νέες ευπάθειες (vulnerabilities) που μπορούν να εκμεταλλευθούν οι επιτιθέμενοι.

Η έρευνα για την ασφάλεια σε υπηρεσίες κοινωνικής δικτύωσης επιβάλλει μια επιθετική προσέγγιση όπου οι ερευνητές "αναλαμβάνουν το ρόλο" του επιτιθέμενου όταν εξερευνούν τα αμυντικά μέσα αυτών των υπηρεσιών. Η "κλειστή" (proprietary) φύση τους περιορίζει την εκτέλεση τους στα ελεγχόμενα πλαίσια ενος εργαστηρίου, και απαιτεί μια black-box προσέγγιση καθώς οι εσωτερικοί μηχανισμοί τους είναι άγνωστοι. Αυτό έχει ως αποτέλεσμα οι ερευνητές να πρέπει να αλληλεπιδρούν με τις πραγματικές υπηρεσίες και τους χρήστες τους. Μόνο τότε μπορούν να προβλέψουν τεχνικές που μπορεί να υιοθετήσουν μελλοντικά οι επιτιθέμενοι, και να αναπτύξουν αποτελεσματικές αμυντικές τεχνικές που θα εμποδίσουν τις πραγματικές επιθέσεις.

Σε αυτή την εργασία επιδεικνύουμε οτι με την χρήση λειτουργιών απο διάφορες ηλεκτρονικές υπηρεσίες με τρόπους για τους οποίους δεν έχουν σχεδιαστεί, μπορούμε να "χτίσουμε" και να εξαπολύσουμε καινοτόμες επιθέσεις. Τα αποτελέσματα απο τα πειράματα μας αποκαλύπτουν τον ευπαθή σχεδιασμό των υπάρχοντων αμυντικών μηχανισμών που χρησιμοποιούν οι υπηρεσίες κοινωνικής δικτύωσης, και την αδυναμία τους να προστατέψουν τα κεφάλαια τους απο τους επιτιθέμενους. Τα χαρακτηριστικά των.επιθέσεων μας και τα πειραματικά αποτελέσματα μας, καθοδηγούν τον σχδιασμό και την υλοποίηση καινοτόμων αμυντικών μηχανισμών.

Προσδιορίζουμε τα εξής στοιχεία ως κεφάλαια για τις υπηρεσίες κοινωνικής δικτύωσης που πρέπει να προστατεύονται απο κακόβουλους χρήστες: (i) οι πληροφορίες των χρηστών, (ii) οι λογαριασμοί (accounts) των χρηστών και (iii) οι ενέργειες των χρηστών. Αναλαμβάνουμε τον ρόλο του επιτιθέμενου και εξαπολύουμε επιθέσεις που παρακάμπτουν τους αμυντικούς μηχανισμούς (αν υπάρχουν) που έχουν ως σκοπό να προστατεύουν αυτά τα κεφάλαια. Πρώτα εξερευνούμε διάφορες τεχνικές για συλλογή και συσχετισμό.προσωπικών δεδομένων χρηστών που μπορούν να χρησιμοποιηθούν για εξατομικευμένες επιθέσεις. Στη συνέχεια, επιδεικνύουμε την αποτελεσματικότητα των επιθέσεων ενάντια σε μηχανισμούς πιστοποίησης χρηστών που χρησιμοποιούν φωτογραφίες. Τέλος, διεξάγουμε εκτενή πειράματα για να εξερευνήσουμε τους αμυντικούς μηχανισμούς υπηρεσιών κοινωνικής δικτύωσης για την ανίχνευση ενεργειών απο κακόβουλους χρήστες.που κάνουν χρήση λειτουργιών που βασίζονται στην γεωγραφική θέση του χρήστη. Σε κάθε περίπτωση βασιζόμενοι στα πειραματικά μας αποτελέσματα, σχεδιάζουμε μηχανισμούς για την μείωση ή πρόληψη (αν είναι εφικτό) των επιθέσεων μας.

Επόπτης: Καθηγητής Ευάγγελος Π. Μαρκάτος

Contents

List of Figures

Chapter 1

Introduction

"*A social network service is an online service, platform, or site that focuses on building and reflecting social networks or social relations among people.*" - Wikipedia

Early social networking services appeared in the '90s in the form of generalized online communities such as Geocities [37]. These services evolved into what we have come to perceive today as social networking sites, with the emergence of Friendster [35] in 2002 and the sites that followed, such as Myspace, Orkut and Facebook. This second generation of social networking services or online social networks (OSNs) have become the fastest growing web services and have attracted the interest of hundreds of millions of users. Facebook has exceeded 1.2 billion active users with almost 700 million users using the service daily [3], while Twitter has more than 200 million monthly active users [78]. The mass adoption of social services has led them to become one of the most prominent online activities, as they take up the greatest share of web users' online time [74]. Online social networks are not all restricted to recreational activities, but can focus on professional activities (e.g., LinkedIn), the dissemination of information (e.g., Twitter), or the promotion of music (e.g., Myspace). Overall, users are able to interact with each other, chat, share articles, play games and upload content. Facebook users upload 350 million photos each day, making it the largest photo-sharing service online [9]. In the attempt to utilize these new services to the fullest, people are disclosing a vast amount of personal information when creating their digital counterparts for interacting with other users.

This abundance of user information and the interconnections between users have made OSNs an attractive target for the Internet miscreants. Users tend to show a great amount of trust to online communication and interactions and adversaries attempt to exploit that trust when deploying attack campaigns, which have proven to be more successful than those deployed through other means [16]. Every person participating in an OSN is vulnerable to a series of threats ranging from identity theft [43] to monetary loss [77]. Even users that are cautious are vulnerable to indirect threats that first exploit their online contacts that don't follow safe practices [100, 232].

Furthermore, with the widespread adoption of smartphones that offer Internet connectivity on the go, a new piece of sensitive information has become available: a user's location. This has greatly influenced social networks and other web services and led to the development of rich location-based functionality. These location-based services (LBS), that are information and entertainment services accessible with mobile devices, have the ability to make use of very

accurate geographical information through the GPS readings of modern mobile devices. User location data is becoming an integral part of social networks and web services that build upon that information to offer novel applications [240].

Apart from the rich and novel functionality offered within the confines of each service, online social networks offer other types of functionality that are reforming the world wide web, and the way people access and use services. For example, social networks are becoming the de-facto solution for single sign-on platforms, as a large number of popular services allow users to sign in using their Facebook [25] or Google+ [49] accounts. Such cross-site operability is rapidly reforming the Internet landscape, introducing privacy concerns [175], as well as functionality that can be exploited by attackers for delivering more effective attacks.

As the functionality of online social networks continues to evolve, and new capabilities are introduced, adversaries are bound to evolve their offensive methodologies and adapt to the latest techniques. Thus, security research in OSNs must follow an adaptive and pre-emptive approach. By approaching these services from a *malicious perspective*, researchers attempt to identify potential vulnerabilities of the systems and novel attack techniques for bypassing existing security mechanisms. Exploring the possible ways in which such services can be exploited and attacked, can provide the vital insight for designing more effective defense mechanisms that will hinder actual incidents. Their proprietary nature often prohibits the deployment of such systems within a controlled setting and, thus, researchers are required to interact with the actual service in order to evaluate their security properties. Thus, it is crucial to design experiments in an ethical manner and minimize the impact on actual users or the service providers.

1.1 Assets of online social networks

The attacks targeting online social networks that have been seen in the wild or demonstrated by researchers, are often intricately designed and may consist of multiple phases. Any user interactions with an OSN can be considered as "assets" for the social networking services, and should be protected from malicious individuals. Specifically, we identify the following assets:

- *User accounts:* a very valuable asset that can be highly profitable to attackers are accounts that belong to legitimate users. Once a user account has been compromised, it can be employed for conducting various malicious actions, which can result in the most effective personalized attacks as all actions originate from a legitimate source.

- *User information:* the most apparent asset of online social networks is the immense amount of (personal) user data that they have accumulated. This information is valuable to attackers as it can be used to craft personalized attacks that are far more effective than traditional generalized attacks. These campaigns can range from spam propagation to industrial espionage [76].

- *User actions:* the final asset is the collection of activities from the users within the social networking service. Various types of functionality within the OSNs, as well as aspects of their business model, are based on user actions. For example, social networks that also employ location-based functionality can be used to deploy recommendation systems. Actions originating from fake accounts or compromised accounts, can pollute the system and have a significant impact on the intended functionality. Thus, it is crucial for social

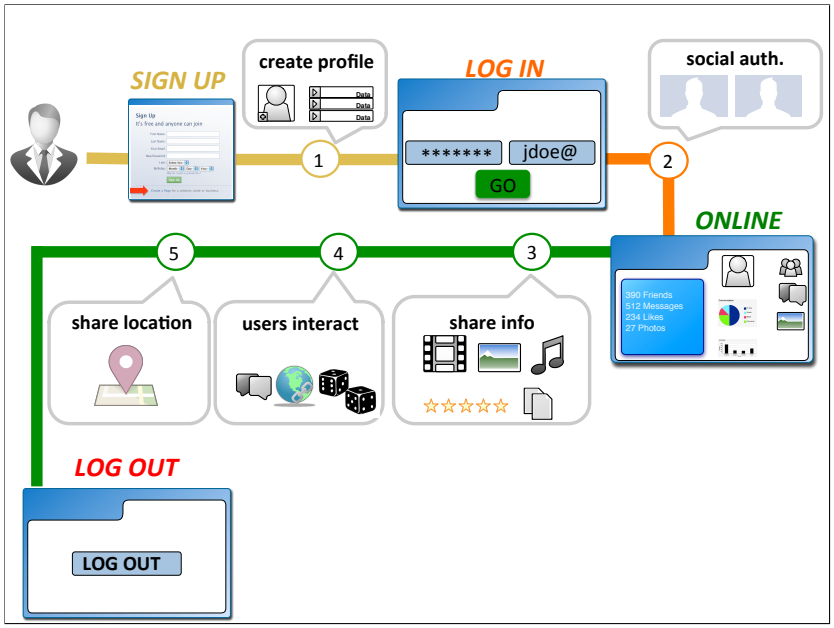


Figure 1.1: Typical user interactions with a social networking service.

networking services to deploy defense mechanisms for detecting fake accounts or their actions that diverge from legitimate behaviour.

In Figure 1.1 we illustrate typical interactions of a user with an OSN. Upon signing up with the service, the user creates his digital representation, i.e., fills in the data to create his social profile (step 1). Next, to log into the service, the user may be required to solve a Social Authentication challenge (step 2). We explore the security characteristics of Social Authentication and expose its weaknesses in Chapter 3, and design a robust SA mechanism in Chapter 4. Once the user is online, he can share more information and interests, publish content, upload photos etc. (step 3). All this data can be harvested by adversaries and used to craft personalized attacks against the user. We explore the efficiency of such techniques in Chapter 5. This data, combined with that from step 1, can used for cloning profiles and deploying attacks that use stolen identities, as we demonstrate in Chapter 6. The user can also interact with other users through messaging, exchanging content or playing games (step 4). In Chapter 7 we build an extensive social forensics toolkit that collects and analyzes user information and interactions. Finally, the user can also take advantage of location-based functionality offered by such services (step 5). We conduct experiments regarding fake location attacks and detection mechanisms in Chapter 8.

Overall, we explore all three categories of assets, and deploy novel attacks that target them. In each case, we design and deploy attacks that target the asset, which also reveal any existing defense mechanisms deployed by the service to protect the assets. We don't attempt to exploit implementation vulnerabilities (e.g., through buffer overflow attacks) but to reveal the ineffective or incomplete design of their defense mechanisms at the application level. Subsequently, based on the attack technique and the experimental results, we identify the deficiencies of existing mechanisms and design defenses that can protect the services and their users from our attacks.

1.2 Challenges of Security Research in OSNs

Exploring the security aspects of online social networks presents a series of interesting challenges. On one hand, the large-scale nature of such services requires efficient and accurate experimentation methodologies as well as a robust infrastructure. Consider that miscreants are known to capitalize on popular events that are expressed through viral behavior on OSNs such as Facebook and Twitter. For example, during the night of the 2012 U.S. presidential election, 31 million Tweets were posted online at a peak rate of 372,452 Tweets per minute [230]. Studying and analyzing such content in search of SPAM or malware campaigns, would be a daunting task. Keeping up with the rate of user-generated content also places significant burden on the network connection both in terms of bandwidth as well as latency. Moreover, maintaining such content for subsequent analysis mandates a large amount of storage space and processing power. On the other hand, the particular nature of security research presents both ethical and legal issues. From the standpoint of the OSN, a researcher might seem like an attacker and from the standpoint of a researcher, probing the service to identify weaknesses might mean producing the tools for the actual attackers.

The explosive growth rate of social networks has created the first digital generation consisting of people of all ages and backgrounds. However, the lack of technical literacy among the majority of users has resulted in a naive approach where the caution demonstrated in social interactions of the physical world has disappeared, as that develops over time and is passed on through generations. As such, security research in social networks, has to take into account the human aspect as adversaries may resort to social engineering (see Section 2.2) to trick users into divulging critical information or bypassing security mechanisms. Thus, when exploring attack and defense techniques, researchers must also investigate the possible outcome if adversaries incorporate social engineering techniques as well. As such, the goal of research is also to highlight dangerous user practices and educate the public by promoting safer usage of these social services.

1.3 Thesis statement and contributions

Online social networks are the most prominently used online services, with ever-increasing rich functionality, that have exposed users to unprecedented threats. Securing online social networks presents a series of challenges, and is a perfect example of the perpetual arms race between security researchers and attackers. However, apart from attacks against the online social networks or their members, user information and profiles have been used to craft attacks against high value targets not associated with these services. These targeted attacks utilize OSN assets to deploy "stepping-stone" attacks that allow them to gain access even to internal systems and networks that are heavily protected. The success of these attacks relies on their personalized nature which exploits the weakest aspect of security mechanisms; the human factor. Thus, it is crucial to develop more effective defense mechanisms for hindering the malicious acquisition or usage of OSN assets.

In this dissertation we demonstrate that *the functionality and inherent structure of information sharing in social networks can be misused for constructing attacks that prove to be effective while remaining sufficiently practical. Our empirical results (i) reveal the deficiencies of OSN defenses and (ii) guide the design of improved defense mechanisms.*
In summary, the contributions of this dissertation are the following:

- We perform the first empirical evaluation of Facebook's Social Authentication (SA) mechanism and demonstrate its weaknesses when employed as the second factor of a two-factor authentication scheme. Our attack takes advantage of the inherent nature of information sharing in social networks and is very effective, even with minimal effort from the attacker. Specifically, we design and implement an automated, modular system that leverages public photos and face recognition, which manages to defeat 22% and significantly assist in 56% of the SA tests. We show that readily available face recognition services offer a very accessible and effective alternative to building a custom face recognition system, which can be utilized by adversaries. Finally, we also demonstrate an effective attack scenario against SA, which previous work has overlooked, that achieves 96.7-100% success rates.

- We conduct the first user study, with 91 participants, that explores the ability of humans to identify their acquaintances in photos taken under realistic, non-ideal conditions. Our results assert our assumption that humans are far superior in identifying faces compared to state of the art software, as they are able to solve 99.1% of the SA tests containing people not identified by software. Based on the insights derived from our experimental results, we design a secure, yet usable SA mechanism, that relies on a novel tag selection and transformation process. We experimentally evaluate our proof-of-concept implementation and demonstrate its robustness against automated attacks. After applying the transformations, face recognition software fails to detect even a single face, and 3 template matching methods pass less than 1% of the challenges while requiring 3 orders of magnitude more processing effort than against non-processed photos. We discuss the benefits of employing our revisited Social Authentication mechanism as a user-gnostic CAPTCHA service and analyze its strengths against traditional CAPTCHA-breaking attacks.

- We evaluate the efficiency of several techniques for harvesting email addresses, which is the first step for mining information to create personalized attacks, with metrics like number of addresses per query or traffic volume per email. We find that using social networks in conjunction with search engines is the most efficient method to harvest large numbers of email addresses. We also demonstrate the effectiveness of a novel method for creating personalized attacks that misuses social networks as oracles, which maps 43.4% of the harvested email addresses to the users' Facebook profiles. Our study regarding the amount of personal user information publicly available, illustrates the extent to which users are vulnerable to identity theft, and the feasibility of personalized attacks. For example, we found that almost half the Facebook profiles reveal their current location and interests, while all LinkedIn profiles contain the user's location and connections. Finally, we implement a tool that detects cases of identity theft in LinkedIn, and conduct the first case study regarding cloned profiles which found duplicates of 7.5% of the accounts.

- We create an extensive social forensics toolset for crawling a wide range of major social and communication services. Our toolset handles all three phases of forensics analysis: (i) collecting data, (ii) correlating data across services, and (iii) visualizing user activities. This toolset can assist law enforcement agencies in utilizing data regarding online user activities for solving criminal cases, and also educate users on the risks of privacy leakage. We conduct a small case study to demonstrate the efficiency of our correlation process

that associates disjoint sets of information from different services, which can also be misused for creating personalized attacks. We are able to correlate 75.4% of the Facebook and 85.6% of the Twitter profiles to user accounts from other services.

- We create a tool for the systematic exploration of server-side components employed by location-based services (LBS) for identifying clients providing fake locations. We use it to conduct the first extensive case study that measures the effectiveness of existing defense mechanisms, and the respective threshold for each heuristic. We reveal that it is possible to check into a place from as far as 200m away, while traveling at speeds up to 932 mph between check-ins. To stress our experimental findings, we develop an adaptive attack algorithm that takes advantage of the threshold heuristics to maximize the impact of fake-location attacks while remaining undetected. Based on the results of our study in Foursquare and Facebook Places, we argue that anomaly detection-based heuristics are not sufficient for capturing clients that misbehave. To that end, we design and implement a system that can be deployed at LBS venues to ensure user presence during the check-in process, and evaluate its performance. Using commodity NFC hardware, and a check-in protocol based on delegation and asymmetric cryptography, our system is able to eliminate a range of attacks, while completing the check-in process in approximately 105 ms.

1.4 Organization of Dissertation

The rest of this dissertation is organized as follows. Chapter 4.1 provides some background information about various attack techniques that have been deployed in online social networks, and defense approaches that have been proposed.

Chapter 3 explores social authentication, a security mechanism that attempts to prevent adversaries from compromising *user accounts*, even if they have acquired user credentials. We first demonstrate how attackers can automatically break this security mechanism. In Chapter 4 we revisit the concept of social authentication and design a more secure mechanism, based on the insight of a user study and our previous attack technique.

In Chapter 5 we present our research regarding the techniques that adversaries can employ for mining and correlating *user information* from social networks, for deploying effective personalized spam and phishing campaigns.

In Chapter 6 we explore how adversaries can take advantage of user data mining and correlation techniques for deploying identity theft attacks. We create a tool for detecting clones profiles, and conduct a study regarding publicly available personal information that can be leveraged for such attacks.

Chapter 7 presents our work in social forensics, i.e., how social networking user data can be collected, correlated and visualized for benign purposes. We build a toolset that can be employed by law enforcement agencies for identifying clues that can assist in solving crimes.

In Chapter 8 we focus on location-aware functionality that has been incorporated in major social networks. Specifically, we explore how adversaries can misuse certain services for gaining profit, through fake *user actions*. After demonstrating our novel attack technique, we also design and implement a novel defense system that can be incorporated by any location-based social network, for validating a user's location.

Chapter 9 presents the related work to the attack and defense methodologies proposed in this dissertation. Finally, in Chapter 10 we summarize the contributions and results of this

dissertation, and outline research directions that can be explored in future work.

1.5 Publications

Parts of the work for this dissertation have been published in international refereed conferences, and workshops:

- Iasonas Polakis, Stamatis Volanis, Elias Athanasopoulos, Evangelos P. Markatos. The Man Who Was There: Validating Check-ins in Location-based Services. In Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC). December 2013.

- Iasonas Polakis, Marco Lancini, Georgios Kontaxis, Federico Maggi, Sotiris Ioannidis, Angelos D. Keromytis, Stefano Zanero. All Your Face Are Belong to Us: Breaking Facebook's Social Authentication. In Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC). December 2012.

- Georgios Kontaxis, Iasonas Polakis, Sotiris Ioannidis, and Evangelos P. Markatos. Detecting Social Network Profile Cloning. In Proceedings of the 3rd IEEE International Workshop on SEcurity and SOCial Networking (SESOC 2011), co-located with the IEEE International Conference on Pervasive Computing and Communications (PerCom). March 2011.

- Iasonas Polakis, Georgios Kontaxis, Spiros Antonatos, Eleni Gessiou, Thanasis Petsas and Evangelos P. Markatos. Using Social Networks to Harvest Email Addresses. In Proceedings of the 9th Workshop on Privacy in the Electronic Society (WPES 2010), co-located with the ACM Conference on Computer and Communications Security (CCS). October 2010.

Chapter 2

Background

The aim of this section is to provide basic background knowledge regarding the various types of attacks that have been deployed against online social networks in the past, and the techniques employed in each case. We present attack techniques that have either been seen in the wild or demonstrated by researchers, and categorize them depending on the overall approach of the attack, or a specific technique that we wish to emphasize. We also present certain defense mechanisms that have been proposed by researchers.

2.1 Data Mining and Correlation

We first identify the challenge that arises from user's participation in a social network, in regards to being targeted by attackers. We believe that the visibility of a user's participation in a network may offer enough information to attackers to make him the target of sophisticated personalized attacks. In this section we focus on a specific aspect of this *fundamental challenge* posed by online social networks; specifically, the *public availability of personal information* and how it can be "mined" by adversaries for nefarious purposes.

An adversary may use the extracted data to re-create (a partial view of) the network's social graph[1]. While it is easier to collect this information, compared to the complete information available in user profiles, various uses of this data have been published that could be used for malicious purposes; detecting communities of users with similar interests [199], identifying well-connected individuals [238], inferring private information from a user's friends [185, 246], and facilitating the efficient collection of user profiles via targeted attacks [119].

As the vast amount of data that social networks have acquired can be considered their major asset, one would expect OSNs to employ various mechanisms for hindering third parties from data mining activities. In practice, however, this is not the case and a multitude of research papers present their findings based on the data extracted from OSNs.

In [192] the authors perform a large-scale measurement study of four popular social networks, namely, Flickr, YouTube, LiveJournal, and Orkut. They crawl large fractions of the social graphs and collect data from over 11 million profiles. With Orkut being the exception, the social networks studied by the authors are mainly used to publish, organize and locate content. In these social networks, friendship is not bi-directional, and follows an approach where a user "likes" or "follows" another user without it being mandatory for the second user to reciprocate the "friendship". In effect, these social networks create graphs with directed

[1] The graph consisting of users (vertices) and their friendship links (edges).

edges, that lead to an interesting graph analysis by the authors. Results confirm that these networks follow a power-law distribution as does the degree distribution of the Internet [132]. Furthermore, they are scale-free [184] and small-world [88].

Bonneau et al. [106] explore different techniques for expanding the data one can extract from user profiles. Instead of only extracting the personal information publicly available they propose 3 techniques to gain access to information viewable by a specific group of users (e.g., the users in an account's friend list). First, an adversary can create a series of fake profiles and issue a large number of friend requests towards target accounts. Once a request is accepted, the adversary now has access to all the information available in the user's profile. Furthermore, the adversary will have access to the data of the user's friends that allow second degree friends to view their profile. Second, an adversary can compromise a large number of random accounts using different techniques (e.g., malicious applications, phishing etc.) to extract all the information available. Finally, an adversary can take advantage of regional/network settings, which allows users from the same network (e.g. a specific university) to view information on user profiles that is not viewable by other users. Gao et al. [139] take advantage of this to harvest data from 3.5 million users, as well as 187 million wall posts these users received.

One of the first such studies focusing on Facebook was performed in [146]. The authors mined the information contained in the Facebook profiles of 4,540 students of CMU, which constituted the whole Facebook CMU population at the time. Their findings show that 87.8% of the users revealed their birth date, 39.9% listed a phone number (with 28.8% disclosing a cellphone number), and 50.8% listing their current residence. From a random sample of 100 students, they manually verified the validity of the information contained in the profiles. 89% of the profiles contained a seemingly real name, 3% a partial name, and 8% a fake name. 61% of the profiles contained an image that allowed the direct identification of the user. 98.8% of the users kept the default privacy setting that made their profile searchable by everyone, and found that only 3 users had profiles that were not viewable by any user. The study was conducted in 2005 when Facebook was mostly used by university students and faculty, which explains the much larger percentages of information revelation compared to the previous two studies. The authors also attempt to identify the privacy implications for users that reveal personal information to the public.

Stalking. Information such as current residence and class schedule will allow a potential stalker to determine the whereabouts of a user. 860 profiles contained such information (since the study was conducted after the end of the semester, in normal cases the numbers would be higher), while 77.77% of the profiles listed an AOL instant messaging (AIM) account. AIM allows users to add "buddies" to their list without knowledge of or confirmation from the buddy being added. Thus, the adversary can track when the user is online.

Demographics re-identification. US citizens can be re-identified using a combination of their 5-digit ZIP code, gender, and date of birth [222]. 88.8% of the profiles reveal their gender and birthday. 50.8% list their current residence, for which ZIP codes can be easily obtained. Overall, 45.8% of the users list their birthday, gender, and current residence.

Face re-identification. With the use of face recognizing software, adversaries can link public social network profiles to profiles of other sites that typically host anonymous profiles (e.g., dating sites).

Social Security Numbers and Identity Theft. When a person's hometown and birthday are know, coupled with information publicly available, an adversary can calculate a large portion of a user's social security number and resort to social engineering to recover the rest.

A study performed by Rabkin [206] attempted to assess the security properties of personal knowledge questions that are used for fallback authentication. The author argues that since such mechanisms owe their strength to the hardness of an information-retrieval problem, in the era of online social networks and the vast availability of personal information their security is diminishing. In this study 12% of the sampled security questions from online banking sites was *automatically attackable*, i.e, the answers to such questions could be found on a user's OSN profile.

Another technique that has been used by adversaries for data mining in social networks, is the creation of malicious applications [58, 59]. To install the application the user must grant it access to all profile information. Once this has been done, the application owner can harvest all the information contained in the user profile. A different use of malicious applications was demonstrated by Athanasopoulos et al. [94], where a social networking service was used as an attack platform. Specifically, all users that installed the application participated in a Denial-of-Service (DOS) against a victim host. A seemingly innocent application that presented users each day with a new photograph from National Geographic, contained Javascript code that also generated HTTP requests towards a victim. While their experimental evaluation was not able to demonstrate a significant attack rate, an application with hundreds of thousands of users would be able to deploy a DOS attack with a significant impact.

2.2 Social Engineering

No one should underestimate the impact that the human factor has on security. Any chain is as strong as its weakest link, and that is also the case with computer security. As such, adversaries often employ various techniques of social engineering to bypass or break security mechanisms by manipulating users. An accurate description of social engineering has been given by Kevin Mitnick[2], arguably among the most famous figures in this context:

> "social engineering is using manipulation, influence and deception to get a person, a trusted insider within an organization, to comply with a request, and the request is usually to release information or to perform some sort of action item that benefits that attacker."

However, this attack vector is not limited to people within an organization. It is also employed against end users in various attack scenarios such as personalized spam and phishing campaigns. The term "phishing" was coined in 1996, when attackers used to refer to a compromised account as phish. At the time, phishers used to trade compromised accounts as a form of electronic currency. Today, phishing has evolved into a more sophisticated threat, with many targets. A typical phishing attack usually entails the adversary posing as someone the user trusts (e.g., a friend, a boss, an administrator, a web service) and requiring them to divulge some information (e.g., a password) or complete an action (e.g. click on a link). Lots of recent incidents highlight the effectiveness of social engineering attacks. Accordingly, the findings of this year's RSA advanced persistent threat (APT) summit [208] designate social engineering as the #1 threat vector.

Most Internet users have come across cases of social engineering, such as in cases of spam emails originating from friends' email addresses [224] which have been compromised. In other

[2] `http://www.time.com/time/magazine/article/0,9171,2089344,00.html`

cases, the emails originate from different addresses and just masquerade their origin to appear to have been sent by a friend. In both cases, the attackers goal is to exploit the implicit trust users show to communication from their online contacts.

Nowadays, with the seemingly universal adoption of online social networks, and the abundance of personal information released, users are unwillingly and unknowingly aiding attackers in launching social engineering attacks. Thus, users of such social services are bound to become the main target of personalized spam campaigns, that incorporate user information to become more convincing.

2.3 Impersonation Attacks

In this section we refer to the practice of adversaries impersonating other users for nefarious activities. The impersonation can take place both when communicating with other users as well as when the adversary contacts the social networking service. We identify several categories of impersonation attacks.

2.3.1 Password Stealing

Even though stolen credit card numbers have been a valuable target for many years, in the recent years attackers have shifted their attention to user credentials ranging from email services to social networking sites. According to Shulman [216] the demand for stolen credit card numbers has taken a plunge, a fact that is reflected by their very low *going price*, as one can purchase them for as little as six cents a piece in bulk . On the contrary, demand for social networking site credentials is high, with the price being determined by several factors, such as the popularity of the specific site or the amount of friends or followers the account has. For example the credentials for a Facebook account may fetch a higher value than a less-popular social application devoted to some niche community, while a Twitter profile with hundreds of followers is far more valuable than one with merely a few dozen. According to the author, OSN credentials can yield over 1,000 times more money than a stolen credit card number. We identify two ways of stealing user credentials; a *direct method* and an *indirect method*.

Direct password stealing

Here we present direct password stealing, where adversaries use different techniques to obtain user credentials for social networks. One method is to lure victims to phishing pages that resemble the login page of a social network, and trick them into entering their credentials [23]. Another technique is to install malicious software on a user's machine, that either operates as a keylogger and collects passwords typed by the victim, or breaks the browser password manager to obtain saved credentials.

Stone-Gross et al. [220] describe the "hijacking" of the Torpig botnet, which used a malware program designed to harvest information from the compromised machines. To do so, Torpig exploits the password manager functionality offered by most web browsers and email clients. During a period of 10 days during which the authors were in control of the botnet, they received 54 thousand credentials for email accounts (which may also be valid for social networking sites, as users tend to reuse passwords across multiple domains) and almost 298 thousand unique credentials for web sites (used on 368 thousand domains), most of which were for web mail and social networking sites [172].

Another example is Facebook Hacker, a tool that allows attackers to steal credentials for Facebook as well as other sites [21]. This tool also exploits the browser's password manager to harvest site credentials. Other malware and phishing scams as well have targeted Facebook credentials [24, 71].

Indirect password stealing

The *interconnection* of web services is a trend that is being widely adopted by social networking services and poses a major security threat. By interconnecting social networking services, when one account is compromised then a series of other services that have been interconnected will be compromised. We refer to *indirect* password stealing as the cases in which an adversary that steals the credentials for a non-social networking service will also gain access to a social networking service account. Before being deprecated, MSN messenger [61] allowed a user to connect his Hotmail account to several social networking services such as Facebook, Myspace and Twitter. Therefore, an adversary that stole a user's email password, would also be able to perform several actions from the user's social network account. While the adversary may not be able to gain full access to the account with this technique, being able to post messages on behalf of the user can lead to the propagation of phishing or spam campaigns.

Furthermore, social networking services have promoted single sign-on platforms, where users can register or sign into other services using their social network credentials. Thus, if an adversary is able to compromise a social network profile, they also gain access to a large number of other online services (regardless if the user uses those services).

2.3.2 Session hijacking

Huber et al. [159, 160] present the friend-in-the-middle (FITM) attack, an active eavesdropping attack that relies on hijacking user sessions. In this attack, the adversary first gains access to the communication between the social networking service and the user either through installing malware on the user's machine or monitoring an unencrypted wireless network. The adversary then clones the HTTP headers that contain the authentication cookies and can interact with the OSN to acquire information or post malicious requests on behalf of the user. Upon hijacking the session, the adversary can proceed with one of the following scenarios.

Friend injection. The adversary can inject his profile in the victim's contact list. Once in the contact list, the attacker can gain access to the information available on the profiles of the victim's contacts that is viewable to second degree friends. Furthermore, he can also post messages that will be seen by the victim and its contacts (e.g., to propagate spam).

Application injection. By adding a third-party application controlled by him, the attacker can automatically acquire all the information contained in the victim's profile. The application could be removed so as not to be noticed, but could also be left as users tend to install a plethora of application, and allow the future extraction of new data added.

Session hijacking. Friend-in-the-middle (FITM) attacks [161] enable sophisticated social engineering attacks, such as context-aware spam and social phishing, where the attacker impersonates the victim to his contacts and utilizes information extracted from their profiles. A Firefox extension that demonstrates the FITM attack was developed by Eric Butler [27]. The extension uses a packet sniffer to intercept all unencrypted cookies towards certain popular social networking sites, such as Facebook and Twitter. Intercepted identities are displayed, and the user can login to a social network as one of those identities.

In a case study [133] Felt demonstrated an XSS attack that enabled her to hijack a user's Facebook session. By discovering a vulnerability in the parsing of a specific tag in Facebook's Markup Language (FBML) she was able to create an application that pushes malicious code on a user's profile page. When a user visits the profile, the browser renders that code and fetches Javascript code from the attacker's server. The Javascript code accesses the session information, and allows the attacker to hijack it and, thus, impersonate the user.

2.3.3 Social Profile Cloning

As the majority of users are not familiar with privacy issues, they often expose a large amount of personal information on their profiles that can be viewed by anyone in the network. In [103] Bilge et al. demonstrate an attack of profile cloning, where someone other than the legitimate owner of a profile creates a new profile in the same or different social network in which he copies the original information and then sends friend requests to the user's contacts. By doing so one can create a fake profile impersonating the legitimate owner using the cloned information. Since users may maintain profiles in more than one social networks, their contacts, especially the more distant ones, have no way of knowing if a profile encountered in a social networking site has been created by the same person who created the profile in the other site.

The usual assumption is that a new profile, claiming to be related to a pre-existing contact, is a valid profile; either a new or secondary one. Unsuspecting users tend to trust the new profile and actions initiated from it. This can be exploited by attackers to lure victims into clicking links contained in messages that can lead to phishing or drive-by-download sites. Furthermore, a cloned profile could be used to send falsified messages in order to harm the original user. Their attack system, called ICloner, consists of four main components.

Crawler. This component is responsible for crawling a social networking site and collecting the contact lists of users, and information from profiles that users have chosen to make public.

Identity matcher. This component analyzes the information in the database and tries to identify profiles in different social networks that correspond to the same person.

Profile creator. This component use the information from the identity matcher to create accounts in a social network where the victims have not registered yet, or to duplicate an existing profile inside the same network.

Message sender. This component logs in the created accounts and automatically sends friend requests to the contacts that are friends with the user of the legitimate profile cloned. Depending on the targeted social networking site, CAPTCHAs might need to be solved to issue certain actions such as creating accounts or sending friend requests.

CAPTCHA analyzer. This component relies on techniques designed to automatically break the CAPTCHAs displayed in a series of OSNs, with a success rate that makes automated attacks feasible in practice.

For their experiments they extracted contact list information for over 5 million user profiles and complete user information from more than 1.2 million users. Their first experiment was to evaluate if people were willing to accept friend requests from people they already had in their contact list and compare it to acceptance rates for strangers. The friendship acceptance rate for the forged profiles was over 60% for all forged accounts (in one case, being as high as 90%). The acceptance rate from unknown users was constantly below 30%, except for one test account that achieved a 40% acceptance rate.

The second experiment was to evaluate the trust users show in these new accounts depending if they belong to people they already had in their contact lists and complete strangers. In

both cases, they sent non-personal messaged that contained a suspicious link with the target's full name in it. For both types of accounts targets clicked on the link in 50% of the cases.

2.4 Social Spamming

As online social networks have attracted hundreds of millions of users, a natural consequence is to also attract spammers. While many spammers leverage social networks to propagate phishing schemes, which we analyze in Chapter 2.5, here we focus on spammers that use social networks to propagate spam messages.

Webb et al. [239] injected 51 fake (honeypot) accounts in Myspace and utilized bots that monitored incoming friend requests. Once a honeypot received a friend request, the bot responsible for that profile downloaded the profile that sent the friend request, and marks it with a timestamp. All friend requests are denied for two reasons. First, to be able to monitor the frequency with which an account will receive requests from a specific spam profile and, second, to avoid classification of their accounts as spam by Myspace. Results showed, however, that while several honeypots received friend requests from the same spam profile, none of the honeypots received more than one request from a specific profile. Furthermore, spam bots send messages in bursts and do not follow a stealthy approach. Next, they clustered the profiles based on the HTML content, so as to identify duplicate or very similar profiles. From the 1,487 spam accounts they had collected only 226 (15.2%) had the same (or nearly the same) HTML content as one of the remaining 1,261 profiles. Next, they clustered the profiles based on the content of the "About me" field which usually contains the deceptive text. This clustering generated 637 unique clusters, which means 850 (57.2%) of the profiles had the same (or nearly the same) "About me" content as one of the remaining 637 profiles. The spam messages point people to web pages that mostly advertise pornographic content, while one of the pages sold male enhancement pills.

A similar approach was followed by Stringhini et al. in [221] to collect information regarding spammers in 3 major social networks, namely Facebook, Twitter and Myspace. 300 *honey-profiles* were created on each OSN by mixing data collected by random profiles from Facebook and Myspace. These profiles followed a passive approach and did not send any friend requests, but accepted those received. During their one year study, their decoy profiles received 4,250 friend requests and 85,569 messages. They manually classified the profiles that sent the user requests and found that 542 belonged to spammers, the majority of them on Twitter. The authors classify spam profiles in the following categories.

Displayer. These profiles do not post spam messages. The information that points to spam URLs is contained on the profile and users must visit it to view the content. All Myspace spam profiles belong to this category.

Bragger. These spam profiles don't send spam to other profiles but rely on status updates in Facebook and tweets in Twitter, to distribute spam content. 163 Facebook and 341 Twitter spam profiles belonged to this category.

Poster. These bots send direct messages to the victims, and all of them were found in Facebook and propagated spam through wall posts (which is the most effective method as it is also seen by a victim's contacts).

Whisperer. These bots send private messages to their victims. 20 bots were in this category, and all were found on Twitter. An interesting finding was that regarding how OSNs fight spam. Facebook spam profiles have an average lifetime of 4 days, Twitter spam profiles 31 days,

Category	Fraction of spam
Free music, games, books, downloads	29.82%
Jewelery, electronics, vehicles	22.22%
Contest, gambling, prizes	15.72%
Finance, loans, realty	13.07%
Increase Twitter following	11.18%
Diet	3.10%
Adult	2.83%
Charity, donation scams	1.65%
Pharmaceutical	0.27%
Antivirus	0.14%

Table 2.1: Breakdown of spam categories for spam on Twitter, based on tweet text.

while none of the Myspace spam profiles were deleted during the experiments. Furthermore, the authors use machine learning techniques to detect "bragger" and "poster" spammers. To do so, they rely on six features; the fraction of accepted friend requests, the ratio of messages containing URLs, the similarity amongst a user's messages, the probability that a profile's friends were selected from a list (calculated based on the similarity of contacts' first names), the number of messages sent and the number of friends the profile has. Their automatic classifier was trained with 1000 Facebook profiles (173 spam, 827 legitimate) and then tested on 790,951 profiles. The classifier detected 130 spam profiles 7 of which were false positives. For Twitter, the training set consisted of 500 profiles, and from the 135,834 crawled profiles 15,932 were classified as spam. Upon manual inspection of 100 random profiles, they found 6 to be false positives.

A large scale study was conducted by Grier et al. in [145] to explore the existence of spam in Twitter. Their dataset consisted of over 400 million public tweets that contained 25 million URLs. Over 2 million tweets were identified using three popular blacklists [42, 51, 82] as either spam, phishing or malware. 95% of the blacklisted URLs directed users towards spam while the remaining 5% were malware or phishing. A very interesting finding is that the vast majority of Twitter accounts that send spam messages, are legitimate accounts that have been compromised by attackers and also exhibit normal user behavior. According to the findings, 50% of the spam uses random terms and cannot be characterized based on its content. The remaining 50% can be broken down as seen in Table 2.1. We can see that almost one third of the characterized spam lured victims by advertising free downloads. Another interesting fact is that spammers leverage several Twitter-specific features to attract potential customers.

Call outs. Spammers use the "@" symbol to mention a specific user in their tweet and personalize it, so as to attract the specific user mentioned. 3.5-10% of the spam tweets are personalized.

Retweets. Certain spam tweets are re-posted either by spammers or legitimate users. Roughly 1.8-11.4% of the spam tweets use this feature.

Tweet hijacking. Spammers retweet messages from other users after appending spam URLs. This is used mainly for phishing and malware tweets (23% of all retweeted phishing and malware tweets), which try to exploit the trust users show in the user that posted the original tweet.

Trend setting. In an effort to create a trending topic, spammers create a flood of tweets containing a specific hashtag with 14% of the total trends being used only by them.

Trend hijacking. This constitutes the most popular technique for spammers, where they exploit the popularity of certain topics and incorporate them in their messages to attract victims.

Benevenuto et al. [101] use machine learning techniques to detect spammers in the Twitter network. By incorporating terms of trending topics, and using URL shortening services such as tinyURL [79] to obfuscate the destination, spammers are able to trick users into visiting their web pages. To distinguish spammers from non-spammers, the authors rely on the following characteristics; the fraction of a user's messages (tweets) that contain a URL, the fraction of tweets that contains spam words, and the average number of hashtags (used to denote a topic) in tweets. By using these metrics the authors evaluate the feasibility of applying a supervised learning algorithm that can detect spammers. They use a labeled data collection to test the effectiveness of their method. In their experiments their method correctly classifies 70% of the spammers and 96% of the non-spammers. The spammers that were incorrectly marked as non-spammers present a dual behavior, i.e., while they follow a non-spamming most of the time, in some occasions they send messages that are considered spam. This coincides with the previous study, where the authors report that the vast majority of spammers use legitimate accounts that have been compromised, and thus present a dual behavior. They authors also apply their method to identify spam tweets instead of spammers while using new metrics based on the tweet content. While this approach bypasses the dual behavior problem and increases its efficiency by detecting 78.5% of spam, it also misclassifies 7.5% of non-spam messages as spam.

Irani et al. [163] coin the term *trend-stuffing* to describe attackers incorporating trending topics in tweets to attract victims. They propose an approach to automatically discover trend-stuffing in tweets. First, for each trending topic, a model is built describing the tweet content so as to distinguish trend-stuffing tweets from legitimate tweets based on the content. Second, for tweets that contain URLs, a meta-model is created to describe the contents of the webpage. The intuition is that the web page content from spam URLs will be irrelevant to that of web pages from legitimate tweets.

Another large study by Gao et al. [139] studied spam in Facebook, that propagates through "wall posts". As wall posts remain on a user's profile page until explicitly removed and can be seen by all the contacts that can view the profile, they are ideal for the propagation of spam campaigns. The authors collect 187 million wall posts sent over a period of 1.5 years. To identify spam campaigns, a graph is created from the wall posts that contain URLs. Each node represents a wall post, and two wall posts are connected if they contain the same URL or similar text. After creating the graph, to identify spam campaigns, one has to only identify which connected subgraphs propagate spam campaigns. The authors claim that spammers exhibit a *distributed* and *bursty* nature (*many* accounts in the cluster send messages in a short period of time), and use these metrics to identify clusters of spam campaigns. Overall, they are able to identify 297 "malicious" clusters that contain 212,863 wall posts. After using a multi-step validation process they are able to identify 199,782 spam wall posts. According to the authors, URLs in spam messages follow one of three different formats. Simple hyperlinks [3] are the most popular format (135,962 posts) due to the convenience for the target user that only has to click them. However, their major disadvantage is the fact that they can be easily detected. Plaintext URLs [4] are rarely used (only in 13,361 posts) due to the inconvenience for the target

[3] e.g., $< a\ href =$"..."$>\ http://www.spamsite.com < /a >$

[4] e.g., *spamsite.com*

of having to copy and paste the URL in the browser's address bar. Obfuscated URLs [5] are able to evade automatic detection techniques, however the target must first comprehend and then reconstruct the URL (used in 50,459 posts). Next, the authors wanted to characterize the domains the spam URLs pointed to. The larger number of distinct domains (8,609) belonged to blogs, probably due to the ease of registering new accounts in such domains. To isolate individual spam campaigns out of the full set of malicious posts, wall posts were classified by identifying characteristic strings in each campaign along with manual classification. 10 campaigns were identified overall. Three types of campaigns were the most popular; campaigns that offered free gifts, campaigns where the spammer lured the victim by saying that someone liked them, and campaigns that described a product (usually drugs). A very interesting finding was that 70.3% of the wall posts, directed the victims to a phishing site, and 35.1% to a page that had malware.

2.5 Social Phishing

Phishing attacks attempt to trick the targeted victims into revealing sensitive information by impersonating a trustworthy entity, such as a site administrator. This information could be the credentials for signing into a e-banking site or a social networking profile. In certain cases, phishing campaigns attempt to persuade victims into transferring money to a bank account by impersonating a friend in trouble or a potential business associate.

Traditionally, phishing campaigns propagate through emails and are impersonal, i.e., they address the victim using generic terms such as "user", "customer" or "subscriber". This, however, may alert victims of the illegitimate origin of the email. Personalized phishing follows a different approach. It is based on the use of personal information of the victim which the attacker has harvested from some external source. In social phishing, a variation of targeted phishing, the emails are crafted in a way so as to look like they originate from a friend or a relative of the potential victim. By compromising accounts, adversaries can impersonate the victims and send messages to their contacts. Thus, attackers can take advantage of the trust users exhibit towards acquaintances or family. With the widespread adoption of social networking services by millions of users, attackers have found a fertile "playground" ideal for such attacks.

Several cases of phishing campaigns propagating through Facebook or other OSNs that tricked users into revealing their passwords have been documented by the electronic press [22, 64, 81]. However, one of the attacks documented went even further and attempted to trick users into sending them money. This variation of the traditional Nigerian (or 419s) scam [218] is a very good example of how information found in social networks can be used for malicious purposes. In this case [53] the attacker used a compromised Facebook account to impersonate the owner of the account and ask the user for money due to an emergency situation.

In an effort to quantify the effectiveness of targeted phishing (also known as spear-phishing [50]) compared to traditional phishing schemes, Jagatic et al. [166] conducted a very interesting experiment. First they crawled several social networking websites so as to infer acquaintance relationships amongst a set of students from their university. The next step was to send a spoofed email to a number of recipients that seemed to originate from a friend of theirs. The email contained a link that redirected them to a phishing site which asked them to login with their university credentials, while the domain name was completely different to that

[5] e.g., www.\tspam site\t.\t \tco\m (remove spaces)

of their university. 72% of the targeted users entered their valid university credentials. At the same time, a control group received the same email address originating from a fictitious person with a university email address. In this case 16% of the targets revealed their credentials. While this percentage is higher compared to those of previous experiments (probably due to the valid university email address), it is still much lower than the one achieved in the case where a friend was impersonated. This demonstrates the effectiveness of using personal information available in social networks to craft highly-effective personalized attacks.

Huber et al. [157] attempt to automate social engineering tasks in an OSN environment and develop the ASE (automated social engineering) bot. Initially, they present a high-level description of their attack cycle.

Plan. The attacker defines the initial parameters for the bot, such as the victim Facebook accounts, the target organization etc.

Map & Bond. The bot maps the organization and bonds with targeted victims. Potential victims are selected based upon certain criteria (e.g., male, single) and the bot initiates communication through Facebook.

Execute. Bot carries out attack since the victim's trust has been gained. Can direct victim to malware, or trick him into revealing confidential information.

Recruit & Cloak. Delete account to eliminate traces, or approach victim's contacts for future attacks.

Evolve/Regress. If the attack was successful, move on to another attack (e.g., use stolen credentials). Otherwise, fall back to simpler attack. To evaluate the effectiveness of their implementation of an ASE bot, they conducted two experiments. In the first experiment they targeted 5 organizations, the employee's of which the bot would have to search for in Facebook and match against certain criteria; single males that also belonged to Facebook's Sweden network. The second experiment attempted to evaluate the chatting functionality of the bot. This is the most important aspect of the bot, as it must trick users into believing it is human so as to create trust before the attack is launched. This experiment was set up as a classic Turing test, in which two Facebook profiles were created that would be used to chat with the test subjects. One profile (named Julian) would be handled by the authors while the other (named Anna) by the bot. The subjects had to answer whether they believed they were chatting to a bot or a human. Results showed that test persons concluded that Anna was a chatbot with 85.1% probability on average while all test subjects agreed that Julian was human and not a chatbot.

2.6 Compromising User Privacy

Here we present attacks that aim to compromise the privacy of OSN users. The techniques presented here are not only used by adversaries with the intent of harming the users, but also from advertisers that wish to personalize their ads and customize them to the preferences and interests of each user. Nonetheless, we will refer to the individuals that resort to such techniques as adversaries, as they compromise users' privacy.

Social networks are the most popular and time consuming online activities with an average Facebook user spending more than 3 hours a day on the site [74]. With users being attracted to OSNs, among other, for the ability to "socialize" with a large, geographically dispersed set of friends, as well as meet new people, users tend to befriend a much larger set of people than they would in the real world. With an average Facebook user having 190 online friends,

many of whom are merely "cyber-acquaintances" [235] and posting a plethora of personal information that all of them can access, social networks are leading to the age of unprecedented public availability of personal information. However, users do not comprehend the dangers of revealing personal information to online buddies many of whom they have never met in the real world [84]. While social networks provide security mechanisms to block access to certain personal information, studies have revealed that users do not comprehend issues of online privacy. Therefore, a challenge is to educate users on matters of online privacy so as to comprehend that the exposure of sensitive information is potentially dangerous.

2.7 Privacy settings and public information

Default settings for Facebook and Twitter allow everyone to view a user's name, friends and pages he is a fan of. A study conducted by Gross et al. [146] revealed that only 0.06% of the users hide the visibility of information such as interests and relationships, while in [179] the authors report that 99% of the Twitter users that they checked retained the default privacy settings. Attackers that harvest this publicly available information can use it to craft personalized attacks that are far more effective than traditional attacks.

Krishnamurthy and Wills [180] examine popular OSNs to provide a characterization of potential privacy leakage. Their viewpoint is of the type of information disclosed by users in their profiles, and the ability of other users to access it. The authors present the default privacy settings for controlling access to each type, as well as the available options in each case. For a series of social networking sites that they crawled, they found that at least 80% of the users retained their default privacy settings. They also studied Facebook privacy settings across different regional networks. A surprising finding is that users in smaller networks tend to share less private information compared to those who belong to networks of a larger size. Also, users seem to care more about protecting their profile info than their list of friends. Finally, they recorded the set of servers contacted during a session of typical actions for 11 OSNs. Results show several third-party domains receive information about what the users are doing.

In a subsequent study [181], the authors define *Personally identifiable information* (PII) as "information which can be used to distinguish or trace an individual's identity either alone or when combined with other information that is linkable to a specific individual". In this very interesting study they describe how third-party servers can exploit the PII leakage of social networks so as to link it with user actions inside these networks or even elsewhere on the Internet. Their methodology is based on the analysis of HTTP headers, and detecting whether a user's ID [6] is leaked to third parties. The authors demonstrate that most users can have their PII linked with tracking cookies or leaked through the HTTP *Referer* header or the Request-URI. According to their results, 11 of the 12 OSNs they tested, leaked the user's OSN ID. Furthermore, in two cases pieces of PII were directly leaked to third-party domains. To defend against PII leakage, the authors first identify the parties involved in the transactions; the user, third-party aggregators, the OSN, and any external applications accessed via the OSN. To prevent leakage, a series of measures has to be taken. The Referer header can be blocked [7], aggregators can filter out any PII-related headers that arrive at their servers and OSNs can strip any visible URI of userid information or create session-specific

[6] OSNs use a unique identifier to recognize each user, and store information about him.
[7] Firefox allows direct blocking, and add-ons allow per-site control.

values. Similarly, applications can strip the id or create internal mappings.

Another important aspect of privacy in social networks is presented in [227]. The authors consider the risk to user privacy from content uploaded by friends, since no existing privacy mechanisms enforce policies over such content. The lack of joint privacy controls over content can reveal personal information about a specific user. The authors formalize multi-party privacy and create a language that reveals undesired exposures by existing security controls. Each user has a different *exposure policy* that specifies which users are allowed to view specific content. To identify unwanted exposure, the set of users that are allowed to view the content based on the privacy policy of the uploader are calculated. If the content is viewable by users that should not be allowed based on the privacy policies of the users that are referenced in the content, then a privacy conflict exits. Based on a dataset of 83K Facebook profiles, the authors find that an average private profile has over 80 references publicly exposed by friends with weaker privacy requirements. Furthermore, the authors demonstrate that based on conversations and references from a user's friends, as well as their common interests, one can predict the user's personal attributes. While this has also been demonstrated in the past [99, 151, 193, 249] the authors refine this technique and differentiate friends based on the frequency of interactions amongst them. By recognizing a user's closest friends, which have a higher chance of sharing common interest and attributes with the user, they are able to predict certain user attributes with a very high accuracy (e.g., 84% for religious views).

2.8 Attacks based on social graphs

By modeling users as vertices and friendship links as edges, we can create the graph of a social network. In this section we present attacks that use social graph information to compromise users' privacy.

In certain social networks such as Linkedin or Facebook, link information between two users may be considered private as it might reveal a connection that the user may not want revealed. Korolova et al. [176] present the *link privacy attack* threat, where an adversary aims to reveal the complete link structure of a specific "neighborhood" in the network, and provide a theoretical and experimental analysis. They also introduce the *lookahead* metric which defines the extent to which links are made visible to a user. For example, the social network has a lookahead of 0 if a user can see only see the other users he links to; it has a lookahead of 1 if a user can see exactly the friends that he links to as well as the friends that his friends link to. A very interesting result, is that the number of user accounts that an attacker needs to subvert in order to obtain a fixed portion of the link structure of the network decreases exponentially with the increase in lookahead provided by the network owner.

Zhoe et al. [250] demonstrate that by knowing part of a targeted user's neighborhood graph, i.e.,some of the neighbors of the user and how these neighbors are connected, an adversary can identify the target in an anonymized dataset and compromise the target's privacy. They also consider another definition of graph anonymity; a graph is k-anonymous if for every node there exist at least k-1 other nodes that share isomorphic neighborhoods. In this case the neighborhood of a node is defined by its immediate neighbors and the connections between them.

Frikken and Golle [138] study the problem of assembling pieces of graphs owned by different parties privately. They propose a set of cryptographic protocols that allow a group of authorities to jointly reconstruct a graph without revealing the identity of the nodes. The

graph thus constructed is isomorphic to a perturbed version of the original graph. The perturbation consists of addition and or deletions of nodes and or edges. Moreover, their methods involve cryptographic protocols that are computationally demanding.

Backstrom et al. [95] describe a set of active attacks that allow an adversary to learn whether an edge (friendship) exists or not between a pair of nodes, using only an anonymized copy of a social network. In the first case, their attacker initially selects a set of users whose privacy he wishes to compromise. The next step involves the creation of "sybil" (fake) accounts which are injected in the social network and attempt to create a link with each of the targeted accounts. Based on the number of links created, the adversary is able to recover the newly created subgraph and with it the profiles of the targeted users. The second attack uses a smaller number of random links between the fake and target accounts, but uses a more complex algorithm to retrieve it. In a similar vein, Narayanan et al. [197] are able to de-anonymize the users of an anonymized social graph using the structure of an auxiliary social network that has a number of overlapping users. Their approach, does not require the creation of fake accounts and works even when the overlapping number of users is small. Nonetheless, their algorithm requires the *a priori* knowledge of a set of seed accounts which are present in both social networks.

Puttaswamy et al. [204] present the social intersection attack that targets social content-sharing applications. In this attack, an adversary can identify the original owner of an anonymized shared data object. In this attack, "two or more compromised users in nearby social circles can periodically perform an intersection of their friend lists, as well as of the shared content they obtain from the social application. The common content observed by all attackers is likely to have come from a common friend". According to their analysis of real data from seven popular social networks, 70% of the users can be uniquely identified if 2 of their friends are compromised. To defend against this type of attack, the authors propose the addition of latent-edges to the social graph in such a way that the system achieves k-anonymity [223]. That is done by connecting the user to 2^{nd} degree friends (friends of friends). The process is the following: "The node first selects a subset of its neighbors. Then it builds a clique with the members of this subset. Finally, it connects the clique members with all the non-clique members in the neighborhood. Latent or virtual edges are added in the process.". They continue with a series of optimizations to reduce latent edges, and then theoretically prove that the StarClique graph structure they create has minimal connectivity necessary to provide k-anonymity against one-hop colluding neighbors.

2.9 Using collateral information

In this section, we refer to work that demonstrates how the use of collateral information can be used to reveal the identity of an OSN user.

A practical attack to de-anonymize social network users was presented by Wondracek et al. [241]. The attack attempts to exploit information about the social network groups the user is a member of so as to uniquely identify the user (or greatly reduce the number of possible candidates). In the threat model the authors describe, an adversary controls a malicious website which the targeted victims visits. Upon visiting the site, the attacker first conducts a history stealing attack [164, 168], in which he probes the browser history of a victim for certain URLs that reveal group pages on a social network that the victim has visited (and may also be a member of). Once the attacker identifies the set of groups the target is a

member of, he uses previous knowledge of group memberships to find users that belong to that set of groups. Thus, the attacker can reveal the target's identity or, at least, trace him down to a small set of candidates. The authors conducted an experiment, targeting the Xing social network. First, they downloaded the membership information of 6,574 public groups containing more than 1.8 million unique members, and 108 private groups with 404,331 group members. Subsequently, they calculated the group fingerprint of each user, and found that for 42.06% that use groups it is exact (i.e., only a single user in the social network is a member of exactly these groups). For 1 million of the users, the attacker can trace the target to a set of less than 32 candidates, for 90% of all users, the candidate set is reduced from initially 1.8 million to less than 2,912 users. In an initial real world experiment with volunteers, the authors were able to de-anonymize 15 of the 26 subjects, while the remaining 11 had no indication of group interaction. In a larger experiment with 9,969 subjects, the authors were able to de-anonymize 1,207 individuals which corresponded to a little over one third of all the subjects that had at least one Xing-specific link in their browsing history.

In [146] Gross et. crawl the Facebook profiles of members of the Carnegie Mellon University network. They argue that based on the information available on a profile (which is viewable by any Facebook user that belongs to the same network) combined with information that is publicly available on the Web or other sources, an adversary can infer the social security number (SSN) of a victim. Furthermore, the profile information can be used to identify the user in an anonymized data source, e.g., hospital discharge data.

2.10 Identifying user location

Krishnamurthy and Wills [183] explore the problem of privacy leakage taking into account a new aspect; users accessing OSNs from mobile devices. Access through mobile devices comes with a very important feature, i.e., the information concerning the user's location. Newer OSNs (referred to as mobile OSNs or mOSNs) such as Foursquare and Loopt, have been created specifically for users to access them through mobile devices and reveal their location, and also take advantage of interconnection features of traditional OSNs to present users with an integrated social networking experience. The authors are interested in *combination leakage*: "are there pieces of information that are on traditional OSNs that when combined with new features in mobile OSNs result in privacy leakage?". From 13 mSONs three of the mOSNs always make a user's checked-in location available to all other mOSN users and three more make it available by default, while two make it available to a user's friends by default. From the mOSNS, eight allow users to connect posts to Facebook, and ten with Twitter. Therefore, information that is visible to a specific set of users on one site can be viewed by a different set of people on a different site. For example, a post including a user's current location on a Foursquare profile that the user has connected to a Facebook account, also becomes visible on that user's Facebook Wall which is visible to all Facebook users by default. Finally, results show that in certain cases, the user's current location is leaked to a third party.

The risk of user location exposure has also been explored by Freudiger et al. in [137]. They focus on location-sharing services (LSSs), which usually comprise of two components; "a localization component, with which users obtain their location and a visualization component, with which users render their location on a map and that of their friends". Third-parties running LSSs can collect user location information that is correlated with the user's online identity and profile them over time. A simple scheme to protect users when sharing their

location relies on an asymmetric cryptographic scheme; location information is encrypted with a secret and the secret is encrypted with the public keys of each user. To protect users when contacting a localization component, the authors propose the use of caching and dummy queries. By caching all access points in areas of interest, users can completely avoid contacting localization components. When the needed information is not available (due to memory size restrictions posed by mobile devices), users can create k-1 queries to the localization server (following the principle of k-anonymity). Even so, users are open to attacks that attempt to link user accesses over time to reveal the dummy queries. A similar approach is proposed for the case of online maps that visualize user location.

2.11 Defense Mechanisms

Here we describe certain defense mechanisms proposed, so as to protect OSN users from from the attacks presented in the presented in the previous Chapters. We also present social network designed to enhance user-privacy. An important aspect of online social networks are third party applications that can extend existing OSN functionality or present new activities, such as games and quizzes. As these applications gain access to a user's data upon installation, they pose a great threat to user privacy [26].

Baden et al. [97] argue that by using exclusive shared knowledge for identification, two friends can verify the true identity of each other in social networks. This can enable the detection of impersonation attacks in such networks, as attackers that impersonate users will not be able to answer questions. Once a user's identity has been verified, public encryption keys can be exchanged. Furthermore, by using a web of trust one can discover many keys of friends-of-friends and verify the legitimacy of user profiles that they don't know in the real world and don't share any secret knowledge.

Persona [96] is an online social network designed to enable users to define their privacy. Users are able to define different groups to which their contacts are assigned, manage each groups membership, and mandate access to resources. The system allows fine-grained privacy settings, as each contact can belong to a set of different groups (e.g., a specific user could be regarded as a "co-worker", "friend" and "neighbor"). The system relies on attribute-based encryption (ABE) [102] to enforce such logic, as messages are encrypted using attributes (e.g., "neighbor" OR "family") and only contacts with secret keys that have the attributes can decrypt the messages.

In [134] Felt et al. study the 150 most popular Facebook applications and conclude that almost all of them could retain their functionality while using a limited interface that would allow them to access an anonymized social graph for user data. Specifically, over 90% of the tested applications have unnecessary access to private data as they only access the public data, and 94% just display that information. The authors propose a simple solution to this problem, privacy-by-proxy. Initially the authors extend FBML the markup language used by Facebook. They add tags that abstract user data and handle it without allowing applications to access private data. Each user is identified by the application, through a unique per-application ID that is consistent across sessions. Thus, an application could request a user's list of friends and would receive a list of application specific user IDs, i.e., an anonymized view of the user's social graph. However, when a user with the appropriate privileges attempts to access user data, the tags are substituted by real data prior to the user viewing them.

xBook is a framework proposed by Singh et al. [217] that provides a hosting service for

social network applications and enforces information flow within the framework so as to prevent untrusted third-party applications from leaking users private information. The core concept is to allow applications to access the information they need so as to retain their functionality and also prohibit these applications from passing the information to third-party entities unless the user has approved it. All applications are hosted on a trusted platform and xBook operates as a mediator for all communication. The authors describe 3 ways of information being leaked, and how their design prevents them. In the first case, an application can share user information with any third party. Since all communication with external entities is mediated by xBook, this is prevented by design. In the second case, an application can pass information of one of its users to another user. As each user is assigned a separate instance of an application, and communication between instances is mediated, this case of leakage is also prohibited. In the third case, an application can recreate the social graph of all its users. This is prevented by not allowing any single component having direct access to the data of all its users; a component can only access an anonymized view of the data set.

Anderson et al. [90] propose a system architecture for online social networks that enables users to protect their private information from other users as well as the OSN operator. Their system assumes a model of an untrusted central server and *smart* clients. As applications will run in a secure sandbox, their access to information or other applications will be mediated by the user's client by exposing an API. An interesting aspect of user privacy highlighted by the authors, is that if a third party can learn the amount of information posted by a user it also knows the amount of information potentially hidden from it. Thus, user content is structured in the form of discrete blocks that contain data and hidden links that point to other blocks. This is accomplished through cryptography.

Xu et al. [245] an early warning system that detects worms that propagate through OSNs. Their system relies on two characteristics exhibited by such worms; they propagate following social connections (friendship links) and generate passively noticeable worm activities (e.g., messages, status updates). Their system relies on monitoring only a small number of users, which are selected based on their high degree of connectivity (i.e., they have many friends/followers). They employ decoy accounts to monitor these users and collect evidence of worm infections. Each decoy account is assigned several friends to mimic normal users. To detect worm activities, when a decoy receives a message from a user, it checks if the other decoy account that monitors that user (2 decoy accounts monitor each user) has received a *similar but not identical message*. If this is the case, then the message is the result of worm propagation with high probability, since OSN worms try to personalize their messages. If the decoy accounts receive an identical message or update, an extra verification process is needed since it could be a benign message, e.g., a group message. In such cases, the system calculates the similarity between this event and the other events received from all the deployed decoy accounts, and if the similarity is high (e.g., over 90%) then with high probability the event is evidence of worm propagation. This can lead to false positives but, according to the authors, can be solved by white-listing events that point to popular domains. In their evaluation using real social graph data from 1.8 million Flickr users, when monitoring 500 users they are able to detect worm propagation when less than 0.13% of the total user accounts are infected.

Chapter 3

User Accounts: Breaking Social Authentication

Studies [216] have shown that traditional underground economies have shifted their focus from stolen credit card numbers to compromised social network accounts, which are sold for the highest prices. Gao et al. [140] found that the vast majority of spamming accounts in OSNs are not dummy profiles created by attackers, but legitimate, existing user accounts that have been compromised. Additionally, new Facebook phishing attacks use compromised accounts to steal personal information [165].

As a standard method for strengthening the security of online user accounts, high-value services such as online banking, and recently Google services, have adopted two-factor authentication where users must present two separate pieces of evidence in order to authenticate. The two factors are such that the risk of an adversary acquiring both is very low. Typically, the two factors consist of something the user knows (e.g., a password) and something the user possesses (e.g., a hardware token). Physical tokens, however, are inconvenient for users, who may not always carry them, and costly for the service that deploys them.

In 2011 Facebook, in an effort to combat stolen account passwords, introduced its so-called Social Authentication (SA), a second authentication factor based on user-related social information that an adversary "half way around the world" supposedly lacks and cannot easily trick the owners into divulging. Following the standard password-based authentication, if Facebook deems it necessary, users are presented with photos of 7 of their friends and are asked to identify them. SA appears to be more user-friendly and practical as (i) users are required to identify photos of people they know and (ii) they are accustomed to tagging photos of their friends—thus implicitly providing the necessary labeled dataset for Facebook.

In this chapter we identify the vulnerable nature of SA and empirically confirm a series of weaknesses that enable an adversary to carry out an effective automated attack against Facebook's SA. The key of SA is the knowledge a user has about his online social circle, whereas an attacker trying to log into the account with stolen credentials lacks. Facebook acknowledges that its heuristics and threat model do not cover the case of friends and family (i.e., anyone inside a user's online social circle) hacking into one's account. The intuition behind our research is that any stranger who obtains a user's password can gain enough data to defeat the SA mechanism.

To this end, we initially conduct a series of experiments to validate our assumptions about the access that an adversary might have to such information. The core of this chapter is the

design and implementation of an automated, modular system that defeats Facebook's SA mechanism. The general principles of our approach allow it to be extended and applied to any photo-based SA system. Initially, during a preparatory reconnaissance phase we obtain a victim's list of friends and the photos accessible from his OSN profile. This includes crawling the publicly-accessible portion of the victim's social graph and (optionally) performing actions that bring us inside the restricted part of the social circle, such as issuing friendship requests to the victim's friends. We then process the collected photos using face detection and recognition software to build each friend's facial model. An attacker is highly unlikely to be familiar with the friends of a victim—at least under the threat model assumed by Facebook—and there lies the security of recognizing one's friends as a security mechanism. However, by acquiring accurate facial models of a victim's friends we are in possession of the key to solving SA challenges. When the SA test is triggered, we lookup the identity of the depicted friends and provide an answer.

At a first glance, it might seem that our attack only affects Facebook users that leave their friends list and published photos publicly accessible. According to Dey R. et al. [122] (2012), 47% percent of Facebook users leave their friends list accessible by default. However, an attacker can always attempt to befriend his victims, thus gaining access to their protected information. Such actions may achieve up to a 90% success rate [103, 107, 196, 231]. That way, the set of vulnerable users may reach 84% of the Facebook population. At the same time, our experiments show that 71% of Facebook users expose at least one publicly-accessible photo album. Similarly, an attacker has very good chances of getting access, through online friendship requests, to profiles with private photo albums. Moreover, even if user A's photos are protected from public view and A does not accept friend requests from unknown people, user B might have a photo of A in which A is tagged (i.e., their face framed and labeled with his real name and Facebook ID). If user B has their photos public, A's tags are implicitly exposed to crawling. Overall, dynamics of OSNs such as Facebook, make it very hard for users to control their data [188, 219] and thereby increase the attack surface of threats against SA. We show that anyone can gain access to crucial information for at least 42% of the tagged friends used to build SA challenges that will protect a user's profile.

Under such minimal attack-surface assumptions we manually verify that our implemented SA breaker, powered by a face recognition module, solves 22% of the real SA tests presented by Facebook (28 out of 127 tests), in less than 60 seconds for each test. Moreover, our attack gives a significant advantage to an attacker as it solves 70% of each test (5 out of 7 pages) for 56% of the remainder tests (71 out of 99 tests). Note that we obtain this accuracy in real-world conditions by relying solely on publicly-available information, which anyone can access: We do not send friendship requests to the victims or their friends to gain access to more photos. Furthermore, our simulations demonstrate that within a maximized attack surface (i.e., if a victim, or one of his friends, accepts befriend requests from an attacker, which happens in up to 90% of the cases), the success rate of our attack increases to 100%, with as little as 120 faces per victim for training, and takes about 100 seconds per test.

A recent study [173], provided a formal analysis of the social authentication weaknesses against attackers within the victim's social circle. We expand the threat model and demonstrate in practice that any attacker, inside and outside the victim's social circle, can carry out automated attacks against the SA mechanism in an efficient manner. Therefore we argue that Facebook should reconsider its threat model and re-evaluate this security mechanism.

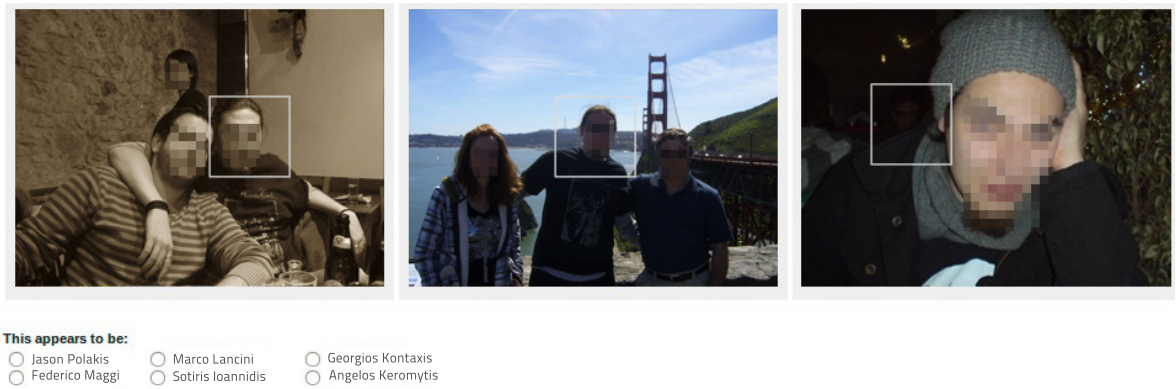


Figure 3.1: Example screenshot of the user interface of a Facebook SA page. The screenshot is synthetic due to copyright reasons, but is an exact replica of a real-world Facebook SA page. Faces have been pixelated for privacy reasons.

3.1 Social Authentication

We hereby describe the nature of Facebook's SA in terms of functionality and heuristics. We go beyond a general description and evaluate its behavior under real-world conditions. Facebook's SA was announced in January 2011 and, to the best of our knowledge, is the first instance of an authentication scheme based on the "who you know" rationale: a user's credentials are considered authentic only if the user can correctly identify his friends.

3.1.1 How Social Authentication Works

After the standard, password-based authentication, the user is presented with a sequence of 7 pages featuring authentication challenges. As shown in Fig. 3.1, each challenge is comprised of 3 photos of an online friend; the names of 6 people from the user's social circle are listed and he has to select the one depicted. The user is allowed to fail in 2 challenges, or skip them, but must correctly identify the people in at least 5 to pass the SA test.

3.1.2 Requirements for Triggering

Based on our analysis, Facebook activates the SA only for the fraction of accounts that have enough friends with a sufficient amount of tagged photos that contain a human face.

Friend list. SA requires that the user to be protected has a reasonable number of friends. From our experiments we have concluded that, in the case of Facebook, a user must have at least 50 friends. To obtain this information, we created 11 distinct dummy profiles and increased the number of friends of these accounts on a daily basis, until we managed to trigger the SA (detailed in §3.3.3).

Tagged photos. The user's friend must be tagged (placed in a labeled frame) in an adequate number of photos. Keep in mind that since these are user-submitted tags, Facebook's dataset can get easily tainted. People often erroneously tag funny objects as their friends or publish photos with many friends tagged, several of whom may not actually be present in the photo.

Faces. SA tests must be solvable by humans within the 5 minute (circa) time window enforced by Facebook. We argue that Facebook employs a face detection algorithm to filter the dataset of tagged people to select photos with tagged faces. From our manual inspection of 127 instances of real SA tests (2,667 photos), we have noticed that Facebook's selection process is quite precise, despite some inaccuracies that lead to SA tests where some photos contain no face. Overall, 84% of these 2,667 photos contained at least one human-recognizable face, and about 80% of them contained at least one face such that an advanced face detection software can discern—in this test, we used `face.com`. To validate our argument on the use of face detection filtering, we repeated the same manual inspection on a different set of 3,486 photos drawn at random from our dataset of 16,141,426 photos (detailed in §3.3.1). We then cropped these images around the tags; hence, we generated a SA dataset in the same manner that Facebook would if it naively relied only on people's tagging activity. Only 69% ($< 84\%$) of these photos contain at least one recognizable human face, thus the baseline number of faces per tag is lower in general than in the photos found in the real SA tests. This confirms our hypothesis that Facebook employs filtering procedures to make sure each SA test page shows the face of the person in question in at least one photo.

Triggering. Facebook triggers the SA when it detects a suspicious login attempt, according to a set of heuristics. Our experiments reveal that this happens when (i) the user logs in from a different geographical location, or (ii) uses a new device (e.g., computer or smartphone) for the first time to access his account.

3.1.3 Advantages and Shortcomings

The major difference from the traditional two-factor authentication mechanisms (e.g., confirmation codes sent via text message or OTP tokens) is that Facebook's SA is less cumbersome, especially because users have grown accustomed to tagging friends in photos. However, as presented recently by Kim et al. [173], designing a usable yet secure SA scheme is difficult in tightly-connected social graphs, not necessarily small in size, such as university networks.

The number of friends a user has may also influence the applicability and the usability of SA. In particular, users with many friends may find it difficult to identify them, especially when there are loose or no actual relationships with such friends. A typical case is a celebrity or a public figure. Even normal users, with 190 friends on average[1], might be unable to identify photos of online contacts that they do not interact with regularly. Dunbar's number [127] suggests that humans can maintain a stable social relationship with at most 150 people. This limit indicates a potential obstacle in the usability of the current SA implementation, and should be taken into account in future designs.

Another parameter that influences the usability of SA is the number of photos that depict the actual user, or at least that contain objects that uniquely identify the particular user. As a matter of fact, feedback [165] from users clearly expresses their frustration when challenged by Facebook to identify inanimate objects that they or their friends have erroneously tagged for fun or as part of a contest which required them to do so.

Finally, in certain cases, Facebook currently presents users with the option to bypass the SA test by providing their date of birth. This constitutes a major flaw in their security mechanism. Obtaining the victim's date of birth is trivial for an adversary, as users may reveal this information on their Facebook profile.

[1] `https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859`

3.1.4 Threat Model and Known Attacks

Throughout this chapter we refer to the people inside a user's online social circle as friends. Friends have access to information used by the SA mechanism. Tightly-connected social circles where a user's friends are also friends with each other are the worst scenarios for SA, as potentially any member has enough information to solve the SA for any other user in the circle. However, Facebook designed SA as a protection mechanism against strangers, who have access to none or very little information. Under this threat model, strangers are unlikely to be able to solve an SA test. We argue that any stranger can position himself inside the victim's social circle, thereby gaining the information necessary to defeat the SA mechanism. Kim et al. [173] suggest that the progress made by face-recognition techniques may enable automated attacks against photo-based authentication mechanisms. At the same time, Dantone et al. [120] have demonstrated that social relationships can also be used to improve the accuracy of face recognition. Moreover, Acquisti et al. [85] went beyond the previous approach and presented a system that can associate names to faces and, thus, de-anonymize a person solely by using a picture of his or her face. Although no scientific experimentation on real-world data has been made to measure the weakness of SA, these studies suggest that the face-to-name relation, which is the security key behind SA, may be exploited further to demonstrate that the scheme is insecure. Our intuition that attackers can overcome the limitations of Facebook's perceived threat model has been the motivation behind this work.

3.1.5 Attack Surface Estimation

In our attack model, the attacker has compromised the user's credentials. This is not an unreasonable assumption; it is actually the reason behind the deployment of the SA. This can be accomplished in many ways (e.g., phishing, trojan horses, key logging, social engineering) depending on the adversary's skills and determination [123]. Statistically speaking, our initial investigation reveals that Facebook's current implementation results in 2 out of 3 photos of each SA page (84% of 3 is 2.523) with at least one face that a human can recognize. This makes SA tests solvable by humans. However, our investigation also reveals that about 80% of the photos found in SA tests contain at least one face that can be detected by face-detection software. This rationale makes us argue that an automated system can successfully pass the SA mechanism. To better understand the impact of our attack, we provide an empirical calculation of the probabilities of each phase. In other words, if an attacker has obtained the credentials of any Facebook user, what is the probability that he will be able to access the account? What is the probability if he also employs friend requests to access non-public information on profiles? To derive the portion of users susceptible to this threat, we built the attack tree of Fig. 3.2. We distinguish between a casual and a determined attacker, where the former leverages publicly-accessible information from a victim's social graph whereas the latter actively attempts to gather additional private information through friendship requests.

Friends list. Initially, any attacker requires access to the victim's friends list. According to Dey et al. [122] $P(F) = 47\%$ of the user's have their friends list public—as of March 2012. If that is not the case, a determined attacker can try to befriend his victim. Studies have shown [103, 107, 196, 231] that a very large fraction of users tends to accept friend requests and have reported percentages with a 60–90% chance of succeeding (in our analysis we use 70%, lower than what the most recent studies report). Therefore, he has a combined 84% chance of success so far, versus 47% for the casual attacker.

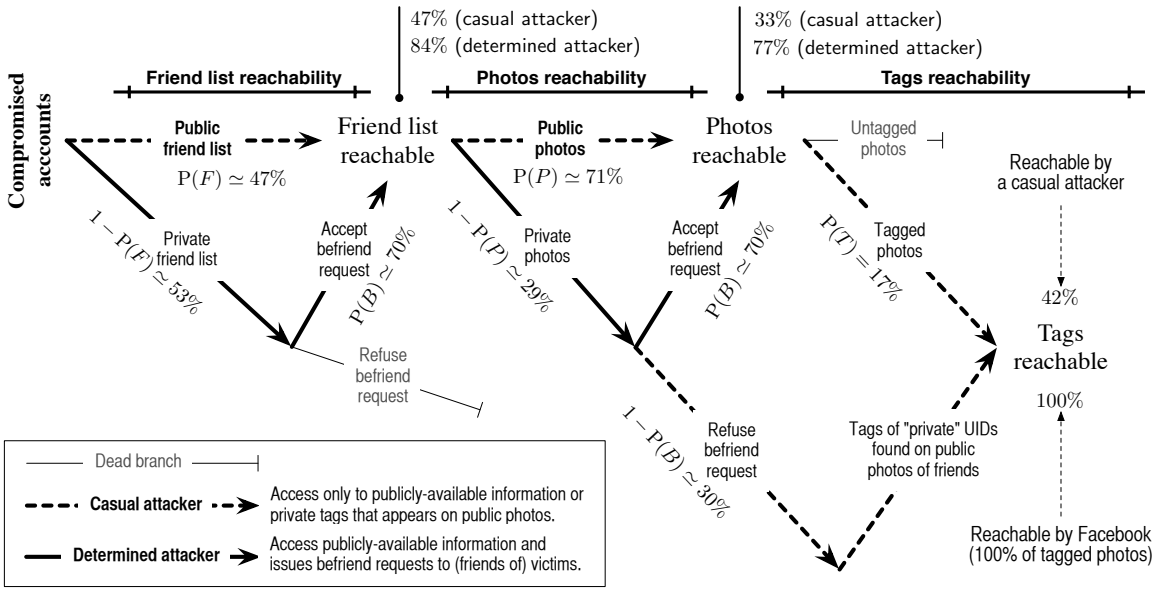


Figure 3.2: Attack tree to estimate the vulnerable Facebook population. Not all the branches are complete, as we consider only the events that are relevant to the case study.

Photos. Ideally the attacker gains access to all the photos of all the friends of a victim. Then with a probability of 1 he can solve any SA test. In reality, he is able to access only a subset of the photos from all or a subset of the friends of a victim. Our study of 236,752 Facebook users revealed that $P(P) = 71\%$ of them exposed at least one public photo album. Again we assume that a determined attacker can try to befriend the friends of his victim to gain access to their private photos with a chance of $P(B) \simeq 70\%$ to succeed, which is a conservative average compared to previous studies. At the end of this step, the determined attacker has on average at least one photo for 77% of the friends of his victim while a casual attacker has that for 33%. This is versus Facebook which has that for 100% of the friends with uploaded photos.

Tags. The next step is to extract labeled frames (tags) of people's faces from the above set of photos to compile $\langle \texttt{uid}, \texttt{face} \rangle$ tuples used by Facebook to generate SA tests and by the attacker to train facial models so as to respond to those tests. By analyzing $16,141,426$ photos from out dataset, corresponding to the 33% of friends' photos for the casual attacker, we found that 17% of these photos contain tags (hence usable for generating SA tests), yet only the 3% contain tags about the owner of the photo. This means that by crawling a profile and accessing its photos it is more likely to get tags of friends of that profile than of that profile itself. The astute reader notices that Facebook also has to focus on that 17% of photos containing tags to generate SA tests: Facebook will utilize the 17% containing tags of all the photos uploaded by a user's friends and therefore generate SA tests based on 100% of the friends for whom tags are available, whereas an attacker usually has access to less than that. In the extreme case, having access to a single friend who has tagged photos of all the other friends of the target user (e.g., he is the "photographer" of the group), the attacker will acquire at least one tag of each friend of the user and will be able to train a face recognition system for 100% of the subjects that might appear in an SA test. In practice, by collecting the tags from the photos in our dataset we were able to gather $\langle \texttt{uid}, \texttt{face} \rangle$ tuples for 42% of

the people in the friend lists of the respective users. Therefore, assuming that all of a user's friends have tagged photos of them on Facebook, a casual attacker is able to acquire this sensitive information for 42% of the tagged friends used by Facebook to generate SA tests. As we show in §3.3.3, with only that amount of data, we manage to automatically solve 22% of the real SA tests presented to us by Facebook, and gain a significant advantage for an additional 56% with answers to more than half the parts of each test. We cannot calculate the corresponding percentage for the determined attacker without crawling private photos (we discuss the ethical reasons for this in §3.4). However, we simulate this scenario in §3.3.2 and find that we are able to pass the SA tests on average with as little as 10 faces per friend.

Faces. Finally, from the tagged photos, the attacker has to keep the photos that actually feature a human face and discard the rest—we can safely hypothesize Facebook does the same, as discussed in §3.1.2. We found that 80% of the tagged photos in our dataset contain human faces that can be detected by face-detection software, and Facebook seems to follow the same practice; therefore, the advantage for either side is equal. Overall, our initial investigation reveals that up to 84% of Facebook users are exposed to the crawling of their friends and their photos. They are, thus, exposed to attacks against the information used to protect them through the SA mechanism. A casual attacker can access $\langle \texttt{uid}, \texttt{face} \rangle$ tuples of at least 42% of the tagged friends used to generate social authentication tests for a given user. Such information is considered sensitive, known only to the user and the user's circle, and its secrecy provides the strength to this mechanism.

3.2 Breaking Social Authentication

In this section we first provide a high level description of our system built to break social authentication tests. We then proceed with a detailed description of certain design and implementation dilemmas we faced, and our solutions.

Our approach applies to any photo-based SA mechanism and can be extended to cover other types of SA that rely on the proof of knowledge of "raw" information (e.g., biographies, activities, relationships and other information from the profiles of one's social circle). We focus on Facebook's SA, as it is the only widespread and publicly-available deployment of this type of social authentication. Our attack consists of a preparation phase (steps 1-3), which the attacker runs offline, and an execution phase (step 4), which the attacker runs in real-time when presented with the SA test. Fig. 3.3 presents an overview of our system's design.

Step 1: Crawling Friend List

Given the victim's UID, a crawler module retrieves the UIDs and names of the victim's friends and inserts them in our database. As discussed in §3.1.5, casual attackers can access the friend list when this is publicly available (47% of the users), whereas determined attackers can reach about 84% of the friend lists by issuing befriend requests. We implement the crawling procedures using Python's `urllib` HTTP library and regular expression matching to scrape Facebook pages and extract content.

Step 2: Issuing Friend Requests

An attacker can use legitimate-looking, dummy profiles to send friendship requests to all of the victim's friends. As shown in Fig. 3.2, this step can expand the attack surface by increasing

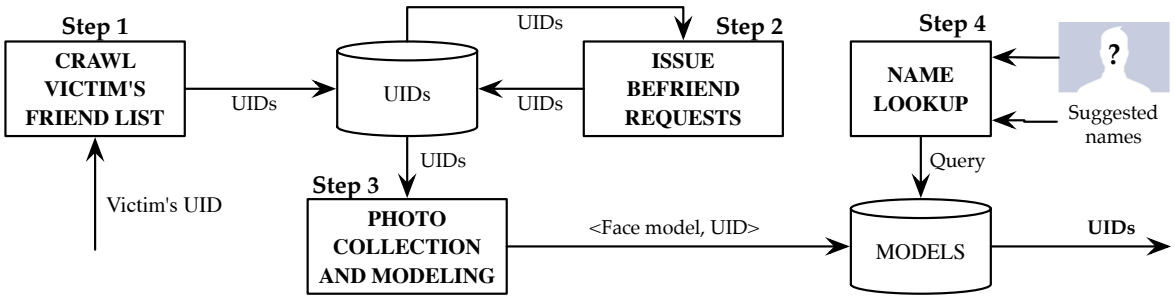


Figure 3.3: Overview of our automated SA-breaking system. It operates in four steps. In **Step 1** we retrieve the victim's friend list using his or her UID. Then, in **Step 2** (optional), we send befriend requests, so that we have more photos to extract faces from and build face classifiers in **Step 3**. In **Step 4**, given a photo, we query the models to retrieve the corresponding UID and thus match a name to face.

the reachable photos. We implement a procedure that issues befriend requests via the fake accounts we have created for our experimental evaluation. Even though we do not collect any private information or photos of these users for our experiments, we need an adequate number of friends in our accounts to be able to trigger the SA mechanism. We select users for our requests, based on the friends suggested by Facebook. Also, as shown by Irani et al. [162], to achieve a high ratio of accepted friend requests, we create profiles of attractive women and men with legitimate-looking photos (i.e., avoiding the use of provocative or nudity photos). In addition, we inject some random profile activity (e.g., status messages, like activities). If Facebook triggers CAPTCHA challenges at some point, our system prompts a human operator to intervene. However, Bilge et al. [103] have demonstrated the use of automated systems against the CAPTCHA countermeasure. Moreover, to hinder spammers, Facebook limits the number of friend requests each profile is allowed to issue in a short period of time and enforces a "cooldown" period of two days on misbehavior. To overcome this obstacle and still have profiles with an adequate amount of friends, we spread our friend requests over a period of one week. We also noticed that for profiles that have education and employment information and send requests to people within these circles, Facebook enforces more relaxed thresholds and allowed us to send close to 100 requests in a single day. In addition, the method described by Irani et al.[162] allows to increase the number of friends passively as opposed to requesting friendships explicitly.

Step 3: Photo Collection/Modeling

- **Photo collection.** We collect the URLs of all the photos contained in the albums of the target's friends using the same screen-scraping approach that we described in Step 1. We then feed the collected URLs into a simple module that does the actual download. This module stores in the database the metadata associated with each downloaded photo: URL, UID of the owner, tags and their coordinates (in pixels).

- **Face Extraction and Tag Matching.** We scan each downloaded photo to find faces. Subsequently, we label each face with the UID of the nearest tag found in the adjacent 5%-radius area, calculated with the euclidean distance between the face's center and the

tag's center. Unlabeled faces and tags with no face are useless, thus we discard them. We save the selected faces as grayscale images, one per face, resized to 130×130 pixels.

Step 4: Name Lookup

When Facebook challenges our system with a SA test, we submit the photos from the SA test to the classifier, which attempts to identify the depicted person and select the correct name. We detect the faces in each of the 7 photos of an SA page and extract the 150 principal components from each face's 130×130 matrix. Then, we use the classifier to predict the class (i.e., the UID) corresponding to each unknown face, if any. If, as in the case of Facebook, a list of suggested names (i.e., UIDs) is available, we narrow its scope to these names. Then, we query the classifier and select the outcome as the correct UID for each unknown face, choosing the UID that exhibits more consensus (i.e., more classifiers output that UID) or the highest average prediction confidence.

3.2.1 Design Phase Details

Here we detail the decisions we took when devising the design of the experiments. In our case, we needed to traverse public parts of Facebook's social graph, such that we could collect photos of users which we had previously befriended using a series of dummy accounts. We then needed to analyze the collected photos to produce a dataset of labeled faces, which we would supply to the face recognition algorithm to build models for the recognition system.

The aforementioned dummy accounts were treated as the victim accounts in our experimental scenario, where we assumed the role of the attacker. In this scenario, the attacker knows the password for the accounts, but lacks the social information to solve the SA challenges presented by Facebook. Thus, face recognition software is employed to overcome the absence of that knowledge.

Data Storage. When the experiments mandate the management of large amounts of data, the selection of the appropriate type of database to be used is driven by two factors: *scalability* (both up and down) and *flexibility*. We decided to implement our system upon a lightweight, non-relational database such as MongoDB or CouchDB. Relational databases such as MySQL and SQLite are not optimized for handling non-transactional (e.g., purely tabular), big data repositories with evolving schema (e.g., new attributes). In addition, OSN are well represented with graphs data structures, which are not natively supported by relational databases. Last, and most importantly, queries in non-relational databases can scale easily thanks to MapReduce. MapReduce has been extensively used in many researches with great benefits (e.g., [149]).

During the phase of data modeling, one must take into account that the principal difference from relational, SQL-like databases is the ability to store and retrieve great quantities of data, and not the relationships between the elements. Thus, JOIN operations are not supported, and data needs to be de-normalized using explicit references between documents for emulating relationships. Even though non-relational databases cannot, necessarily, give full ACID guarantees, at the same time they do offer a distributed, fault-tolerant architecture and the possibility to scale horizontally. These characteristics fit the prerequisites that stem from managing large amounts of data, where performance and real-time responses are more important than consistency.

Furthermore, apart from being suitable for the management of large volumes of data,

non-relational databases are also very flexible as there is no restriction for the mandatory application of a fixed schema. This results in the ability to change the structure of the collected data even after an experiment has started, without the need of rebuilding the entire database to make the old data consistent with the new structure.

In practice, among all non-relational databases, we chose *MongoDB*[2], a document-oriented database, where data is handled in *collections* of *documents*. To draw a comparison between this concept and that of the SQL style, we could say that collections are like tables, and documents can be considered records. While every record in a table has the same sequence of fields, documents in a collection may have fields that are completely different. Additionally, the format of the responses (JSON) returned from the services in our experiments, perfectly matched the native data type of *dictionaries* in Python. In addition, JSON is the data-exchange format adopted by many web-service APIs (including Facebook's). Also, in cases of multiple institutions collaborating on the same project, MongoDB offers two methods of cooperation: *replication* and *sharding*. The first one occurs through groups of servers, known as replica sets, and ensures redundancy, backup, and automatic failover. The latter distributes a single logical database system across a cluster of machines.

File Storage. The next design decision was about the type of file storage to be used. The available options in our case were a typical filesystem versus GridFS, a system for storing an unlimited number of arbitrarily-large files directly into MongoDB. The machine we chose for the experiments was already equipped with an ext3 formatted drive. The problem, however, is that the maximum number of i-nodes per directory is 32,000 in ext3, and that could pose serious limitations on the data we were about to gather and its organization on disk. Even though this limitation can be overcome using an ext4 filesystem or modifying some internal parameters by rebuilding the ext3 filesystem, this was not an optimal choice because folder indexing would have taken a considerably large time when the folder was accessed. While caching would have surely reduced this overhead, given the amount of data to be saved in files, it would not have solved the problem completely. Therefore, we decided to rely on GridFS, which also allowed to easily reallocate the underlying database on a new, larger drive in case more space was needed. GridFS works by breaking large files into multiple chunks: it saves the chunks in one collection (`fs.chunks`) and metadata about the file in another collection (`fs.files`). When a query for a file is submitted, GridFS queries the chunk collections and returns the file one piece at a time. While a filesystem can be seen as the simplest and fastest way, GridFS presents other advantages as well: data replication facilitates load balancing among distributed servers, millions of files can be co-located in a single logical directory without any performance impact and it has increased portability as file management is independent of the application technologies.

Face-recognition software. When designing our experiments, we identified two type of experiments that we needed to conduct. The first one demanded a more versatile approach towards the selection of algorithm parameters, while the second one was more demanding in terms of face-recognition accuracy. As such, we ended up building a custom solution as well as relying on an existing cloud-based service.

Custom solution. The major advantage of a custom solution is the versatility in parameter tuning, as every aspect of the algorithm can be rigorously tested with different values. This was very important for specific experiments where we needed to measure the correlation between the size of the training dataset (i.e., number of faces), and the accuracy in recognizing a face.

[2]http://www.mongodb.org

Furthermore, a custom solution allows to conduct multiple offline tests without incurring any limitations from the service provider. Finally, no network latency is present, which greatly affects the experiments' duration when dealing with large amounts of data.

Nonetheless, a custom solution also presents some disadvantages. First and foremost, it takes time and effort to refine it so as to be comparable to state-of-the-art systems. Specifically, the custom solution was effective when simulating a scenario of an attacker that has obtained a fairly large set of photos by infiltrating the victim's social circle with a dummy account. However, in the scenario of an attacker that only relies on a small amount of publicly-available data, the accuracy of our custom solution was not as satisfactory. For this set of experiments, we needed a more accurate process, and explored the possibility of employing a cloud-based service that provides a more accurate face detection algorithm. Also, the computational power requirements are not negligible, as these algorithms are computationally intensive and have long execution times on commodity machines.

Existing cloud-based service. The major advantage of a state-of-the-art solution is the far greater accuracy compared to a custom solution. Development of the face-recognition system is effectively being outsourced to the service, which offers a production-ready tool for researchers to use—although, as mentioned, with limited tweaking options. In terms of available resources, depending on the service and its usual load, researchers may be able to significantly increase their processing capabilities as opposed to utilizing only their local means. By building upon an existing service, no development time is needed for designing and implementing an algorithm that will be far less accurate (unless developed by computer-vision experts), and can be better allocated on other core tasks. Another advantage of a cloud-based solution is that the REST constraints, to which many services adhere nowadays, ensures scalability. The most limiting disadvantage of using an existing service, is the restriction of API usage. As the number of API requests per hour is limited, conducting a large number of experiments will last longer than using a custom solution where an infinite number of experiments can be conducted without any restrictions.

3.2.2 Implementation Phase Details

Computing resources. All our experiments were conducted on a single machine with a 8-core Intel(R) Xeon(R) E5440 @ 2.83GHz processor with 8 GB of RAM. The machine was used for the entire duration of the experiments and for all their phases, including crawling, offline tests and API hosting. However, it was not fully dedicated to our project as its resources had to be shared with other projects. Regardless of this relatively-small amount of computing resources, we managed to collect an ample dataset, and conduct the experiments on it.

Facebook Crawling. One of the most important aspects of research in OSNs is the collection of data. The massive user base mandates the retrieval of large amounts of data that will consist a large enough sample to accurately reflect the various properties of the network. As such, the implementation of the crawler was integral to our experimental process. The first dilemma we encountered was to decide whether our crawler would use Facebook's public APIs to collect the information we needed, or if we would build an entirely custom solution that did not rely on the API. Our first concern was whether a strict rate limit applies for the API usage. However, Facebook allows 100 million API calls per day[3], which is large enough for our intended experiments.

[3]https://developers.facebook.com/policy/

On the other hand, for our experiments we wanted to collect the data in the manner of an attacker, who collects any data left publicly available by Facebook users. Therefore, we implemented our solution so as to mimic the behavior of a "normal" user browsing the website. Thus, for every user, we retrieved the actual page that contains the list of friends, followed by the albums and photos. The entire solution was implemented in `Python` and every single web request was issued through the `urllib2` library, which impersonated the HTTP User Agent of a popular Web browser.

The main problem with our tool was that some steps in the crawling procedures (i.e., album and photo retrieval) were much slower than others, which resulted in them becoming a severe bottleneck of the system. To overcome this obstacle, we built our crawler in a completely modular and asynchronous fashion. We built four standalone modules, each consuming data from an input queue and inserting tasks in an output queue that was in turn processed by the following component or saved into the database.

The first module was the *Friend Collector*. It took a list of user IDs (UIDs) as input and browsed the Facebook profile of each one. It retrieved all the data of the user's contacts (name and UID) using regular expressions created specifically for that page's structure. If the number of friends was too large and they did not fit on one page, the module performed multiple requests (just like a browser would have done) to get all of them in multiple passes. Every bit of information was saved into our database and marked as *non-crawled*; at the same time the new UIDs were put in the input queue, so that they would eventually be processed and their friends would be retrieved as well. At the end of this step the initial UID was placed in the output queue, ready to be handled by the other modules.

The second module was the *Album Collector*. Each UID in the input queue was used to reconstruct the URLs of their photo album pages, which were subsequently scraped, and the exact URLs of all the albums were saved into the database and put in the output queue. The user with the respective UID was then marked as *crawled*.

This same structure was used by the *Photo Info Collector*. It took as input the queue of album URLs and crawled them one by one, saving into the database the real URL of the photo as well as all the tags each photo contained (coordinates and related UID) and placed the URLs in an output queue.

The last module was the *Photo Downloader*, which downloaded every image it found on its input queue and saved it into the database using the MD5 hash as the key, to avoid duplicates. Once we started the crawling procedure we understood two key factors. First, this crawling process was much faster than relying on the public APIs. Second, it was crucial to follow a pipeline design and effectively distribute resources among the modules, as there was no way we could efficiently retrieve all the data following a sequential process of each user. Overcoming this issue was not trivial. In our initial experiments, our system was acquiring user information for a massive amount of users, while the number of downloaded albums and photos was, of course, significantly smaller. That resulted in our database being filled with potential targets for which we did not have any useful information for our experiments (i.e., photos and tags). For this reason we created an entire web application to keep the single queues monitored and be able to modify the behavior of the single modules at runtime: we could change the number of threads they used, change the request rate or even start and stop them at will, so as to deallocate resources when necessary. After a couple weeks of fine-tuning, we ended up putting as few resources as possible on the *Friend Collector* allocating no more than one thread and using a low request rate, while the greatest part of the resources was assigned to the downloading of album and photo information, with up to 32 threads per module.

Mimicking user behavior. The major asset of an OSN is the vast amount of data that OSNs have acquired. Consequently, they deploy various mechanisms for detecting and preventing automated crawlers from collecting that data. As aforementioned, during our experiments we conducted various actions on Facebook, such as creating test profiles, crawling the network to obtain friend lists and photo information (URLs and tags), and downloading photos. As these actions can lead to the account being suspended, any good crawling system should incorporate measures to avoid triggering such mechanisms. A very important measure is to refrain from "flooding" the OSN with a large amount of requests in short periods of time. In addition, by configuring the crawler to conduct other (automated) actions that resemble the behavior of a human user, we were able to perform our crawling experiments with a stealthier approach and avoid triggering the security mechanisms in most cases—triggering them occasionally is unavoidable. Specifically, we had a component that logged in as our dummy accounts, and mimicked certain user actions such as "liking" posts of other users and posting trivial status updates.

Network multi-presence. During our experiments, we needed a mechanism for triggering Facebook's SA mechanism. During manual inspection we found that the mechanism was triggered when logging in from geographical locations that had not been associated with the account in the past (i.e., from an IP address belonging to a different country). To add this functionality to our system, we resorted to ToR [125]. By enabling our system to access Facebook through the ToR network, we were able to automatically trigger the SA mechanism. Unfortunately, after a number of logins from a specific location, Facebook stopped triggering the mechanism. However, to bypass that restriction we periodically changed the ToR circuits. This demonstrates that the ToR network can be effectively used for experiments that don't relate to privacy matters, but require a virtual presence at dispersed geographical locations. The downside when using ToR is that the bandwidth is reduced substantially. This factor should be accounted for when planning for the time needed to complete experiments.

Face-recognition software. The face-recognition software was the core component of our data analysis phase. As such, we had designed various experiments for evaluating the efficiency of our attack, and exploring whether it poses a realistic threat. We built a custom solution, which presented the advantage of versatility as we could fine tune all algorithm parameters. In addition, as this solution lacked the accuracy of state-of-the-art solutions, we also resorted to using a cloud-based solution with much higher accuracy.

Using existing systems can greatly reduce implementation time and yield better results. If they are not modifiable (as with cloud-based services) one can resort to hybrid solutions of existing and custom-built components. Depending on the requirements of each experiment, the appropriate component can be used.

Custom Solution. To detect faces that needed to be labeled with the respective tag information found in each photo, we used a face-detection classifier part of the OpenCV[4] toolkit. Even though there are plenty of face-detection techniques available in the literature, which are more accurate than OpenCV, our goal was to demonstrate that current face-based SA offers only a weak protection. Even with simple, off-the-shelf solutions, an adversary can implement an automated attack that breaks it.

For the training part, which constructs a facial model of a user's faces, we used the `sklearn` library to construct a supervised classifier: We tested K-nearest-neighbors (kNN), tree and support-vector (with a radial-basis kernel) classifiers using a K-fold cross-validation

[4]http://opencv.org/

technique. We found that support-vector classifiers (SVC) yield the highest accuracy, but are computationally expensive. Therefore, we used kNN classifiers, with k = 3, as they provide a faster alternative to SVC with comparable accuracy.

Existing cloud-based service. We also investigate whether we can employ advanced face-recognition software offered as a cloud service. We selected `face.com`, which offered a face-recognition platform that allowed developers to build their own applications. The service exposes an API through which developers can supply a set of photos to use as training set, and then query the service with new unknown photos for the recognition of individuals. The service allowed developers to use up to two different sets of training data, referred to as "namespaces". Each set can hold up to 1,000 users, and we found no restriction on the number of photos that could be used to train a user. A restriction is set on API usage, with 5,000 requests allowed per hour. Such a usage framework may be restrictive for building popular applications with thousands of users but it is more than fitting for the tasks of an adversary seeking to defeat photo-based social authentication. Assuming the free registration, one may create a training set for up to 1,000 of a victim's friends (the max limit for Facebook is 5,000 although the average user has 190 friends). After that, one can register more free accounts or simply delete the training set when no longer necessary and reclaim the namespace for a new one.

We developed a module for our system that leverages the `face.com` API as an alternative, service-based implementation for conducting steps 3 and 4 of Figure 3.3. The photos are submitted to the service via the `faces.detect` API call to identify any existing faces and determine whether they are suitable for training the facial model classifier. The selected photos are then labeled with the respective UIDs of their owners with `tags.save`. Finally, the provided labeled dataset is used for training through the `faces.train` API call. Once this process has completed, `faces.recognize` can be used for submitting face recognition queries (step 4). The service returns a positive or negative face recognition within seconds. An interesting feature provided, is the ability to limit the face matching process to a specific set of users from the namespaces. We used this to narrow the search space for every Facebook SA challenge down to the six suggested names.

3.3 Experimental Evaluation

Here we evaluate the nature of Facebook's SA mechanism and our efforts to build an automated SA solving system. We first assess the quality of our dataset of Facebook users. We consider this a representative sample of the population of the online social network. We have not attempted to compromise or otherwise damage the users or their accounts. We collected our dataset as a casual attacker would do. Next we evaluate the accuracy and efficiency of our attack. In §3.3.2 we use simulation to play the role of a determined attacker, who has access to the majority of the victims' photos. In §3.3.3 we relax this assumption and test our attack as a casual attacker, who may lack some information (e.g., the victims may expose no photos to the public, there are no usable photos, no friend requests issued). More details on the capabilities of these two types of attacker are given in §3.1.5.

We implemented custom face recognition software for certain experiments. This was done for two reasons. First, we needed something very flexible that would allow us to perform as many offline experiments as needed for the experiments of the determined attacker. Second, as aforementioned, we wanted to show that even off-the-shelf algorithms were enough to break the SA test, at least in ideal conditions. However, superior recognition algorithms exist, and

	Total	Public	Private
UIDs	236,752	167,359	69,393
Not tagged	116,164	73,003	43,161
Tagged	120,588	94,356	26,232
Mean tags per UID:		19.39	10.58
Tags[9]	2,107,032	1,829,485	277,547
Photos	16,141,426	16,141,426	(not collected)
Albums	805,930	805,930	(not collected)

Table 3.1: Summary of our collected dataset.

we conducted exploratory experiments that showed that `face.com`, although less flexible than our custom solution, has much better accuracy. Therefore, we decided to use it in the most challenging conditions, i.e., to break SA tests under the hypothesis of the casual attacker.

3.3.1 Overall Dataset

Our dataset contains data about real Facebook users, including their UIDs, photos, tags, and friendship relationships, as summarized in Table 3.1. Through public crawling we collected data regarding 236,752 distinct Facebook users. 71% (167,359) of them have at least one publicly-accessible album. We refer to these users as public UIDs (or public users). The remaining 29% of UIDS (69,393) keep their albums private (i.e., private UIDs, or private users). We found that 38% of them (26,232 or 11% of the total users) are still reachable because their friends have tagged them in one of the photos in their own profile (to which we have access). We refer to these UIDs as semi-public UIDs (or semi-public users). Data about the remaining 62% of UIDs (43,161 or 18% of the total users) is not obtainable because these users keep their albums private, and their faces are not found in any of the public photos of their friends. The public UIDs lead us to 805,930 public albums, totaling 16,141,426 photos and 2,107,032 tags[5] that point to 1,877,726 distinct UIDs. It is therefore evident that people exposing (or making otherwise available) their photos are not only revealing information about themselves but also about their friends. This presents a subtle threat against these friends who cannot control the leakage of their names and faces. Albeit this dataset only covers a very small portion of the immense Facebook user base, we consider it adequate enough to carry out thorough evaluation experiments.

3.3.2 Breaking SA: Determined Attacker

The following experiment provides insight concerning the number of faces per user needed to train a classifier to successfully solve the SA tests. We create simulated SA tests using the following methodology. We train our system using a training set of $K = 10, 20, \ldots, 120$ faces per UID. We extract the faces automatically, without manual intervention, using face detection as described in §3.2. We then generate 30 SA tests. Each test contains 3 target photos per 7 pages showing the face of the same victim. The photos are selected randomly from the pool of public photos we have for each person, from which we exclude the ones used

[5]On 11 April 2012, our crawler had collected 2,107,032 of such tags, although the crawler's queue contained 7,714,548 distinct tags.

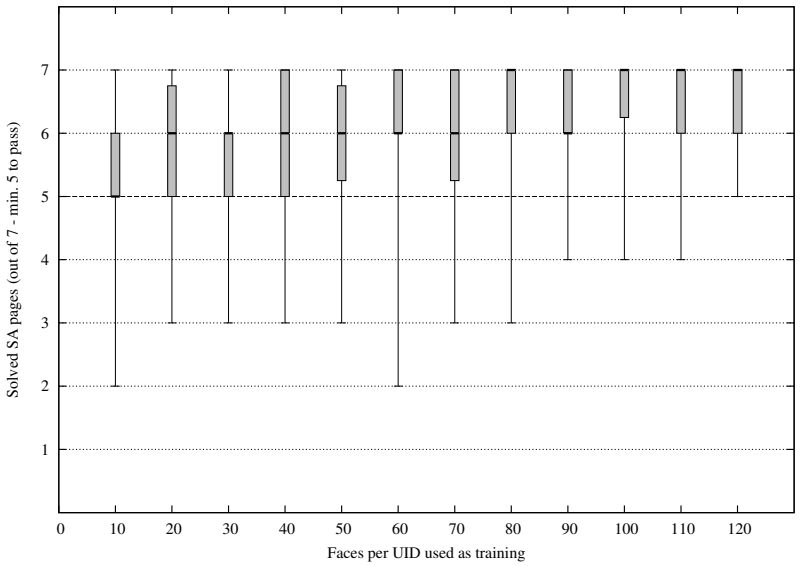


Figure 3.4: Percentage of successfully-passed tests as a function of the size of the training set. For each iteration, 30 randomly-generated offline SA tests were used.

for the training. For each page and K we record the output of the name-lookup step (step 4), that is the prediction of the classifier as described in §3.2, and the CPU-time required. Fig. 3.4 shows the number of pages solved correctly out of 7, and Fig. 3.5 shows the CPU-time required to solve the full test (7 pages).

For an SA test to be solved successfully, Facebook requires that 5 out of 7 challenges are solved correctly. Our results show that our attack is always successful (i.e., at least 5 pages solved over 7) on average, even when a scarce number of faces is available. Clearly, having an ample training dataset such as $K > 100$ ensures a more robust outcome (i.e., 7 pages solved over 7). Thus, our attack is very accurate. As summarized in Fig. 3.5, our attack is also efficient because the time required for both "on the fly" training—on the K faces of the 6 suggested users—and testing remains within the 5-minute timeout imposed by Facebook to solve a SA test. An attacker may choose to implement the training phase offline using faces of all the victim's friends. This choice would be mandatory if Facebook—or any other Web site employing SA—decided to increase the number of suggested names, or remove them completely, such that "on the fly" training becomes too expensive.

3.3.3 Breaking SA: Casual Attacker

In the following experiment we assume the role of a casual attacker, with significantly more limited access to tag data for the training of a face recognition system. At the same time we attempt to solve real Facebook SA tests using the following methodology. We have created 11 dummy accounts that play the role of victims and populate them with actual Facebook users as friends and activity. Then, we employ a graphical Web browser scripted via Selenium[6] to log into these accounts in an automated fashion. To trigger the SA mechanism we employ

[6] `http://seleniumhq.org`

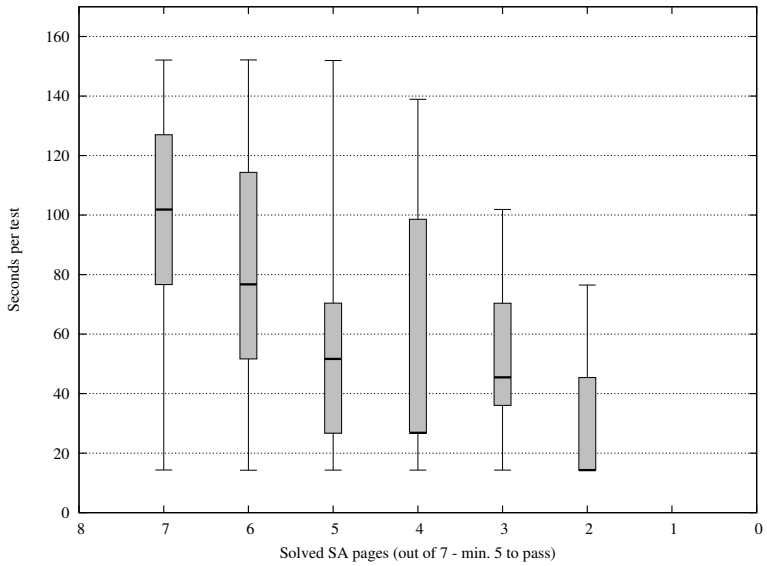


Figure 3.5: Time required to lookup photos from SA tests in the face recognition system.

Tor[7] which allows us to take advantage of the geographic dispersion of its exit nodes, thus appearing to be logging in from remote location in a very short time. By periodically selecting a different exit node, as well as modifying our user-agent identifier, we can arbitrarily trigger the SA mechanism. Once we are presented with an SA test, we iterate its pages and download the presented photos and suggested names, essentially taking a snapshot of the test for our experiments. We are then able to take the same test offline as many times necessary. Note that this is done for evaluation purposes and that the same system in production would take the test once and online. Overall, we collected 127 distinct SA tests.

We tried breaking the real SA tests using our module for `face.com`. Fig. 3.6 presents the outcome of the tests. Overall we are able to solve 22% of the tests (28/127) with people recognized in 5–7 of the 7 test pages and significantly improve the power of an attacker for 56% of the tests (71/127) where people were recognized in 3–4 of the 7 test pages. At the same time, it took 44 seconds on average with a standard deviation of 4 seconds to process the photos for a complete test (21 photos). Note that the time allowed by Facebook is 300 seconds.

We further analyzed the photos from the pages of the SA tests that failed to produce any recognized individual. In about 25% of the photos `face.com` was unable to detect a human face. We manually inspected these photos and confirmed that either a human was shown without his face being clearly visible or no human was present at all. We argue that humans will also have a hard time recognizing these individuals unless they are very close to them so that they can identify them by their clothes, posture or the event. Moreover, in 50% of the photos `face.com` was able to detect a human face but marked it as unrecognizable. This indicates that it is either a poor quality photo (e.g., low light conditions, blurred) or the subject is wearing sunglasses or is turned away from the camera. Finally, in the last 25% of the photos a face was detected but did not match any of the faces in our training set.

Overall, the accuracy of our automated SA breaker significantly aids an attacker in

[7] `http://www.torproject.org`

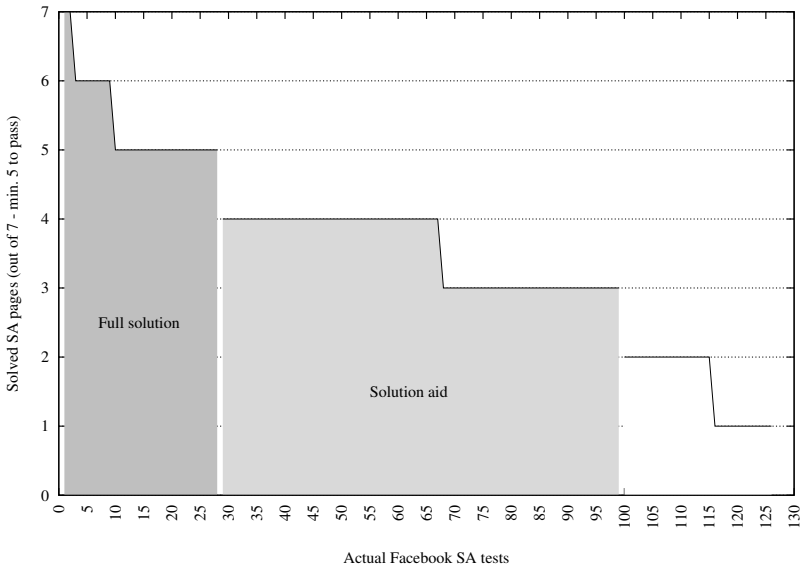


Figure 3.6: Efficiency of automated SA breaker against actual Facebook tests.

possession of a victim's password. A total stranger, the threat assumed by Facebook, would have to guess the correct individual for at least 5 of the 7 pages with 6 options per page to choose from. Therefore, the probability[8] of successfully solving an SA test with no other information is $O(10^{-4})$, assuming photos of the same user do not appear in different pages during the test. At the same time, we have managed to solve SA tests without guessing, using our system, in more than 22% of the tests and reduce the need to guess to only 1–2 (of the 5) pages for 56% of the tests, thus having a probability of $O(10^{-1})$ to $O(10^{-2})$ to solve those SA tests correctly. Overall in 78% of the real social authentication tests presented by Facebook we managed to either defeat the tests or offer a significant advantage in solving them.

After these experiments, we deleted all the photos collected from the real SA tests, as they could belong to private albums of our accounts' friends, not publicly accessible otherwise.

3.4 Ethical Considerations

In this chapter we explore the feasibility of automated attacks against the SA mechanism deployed by Facebook. As our experiments involve actual users, the question of whether this is ethically justifiable arises. We believe that research that involves the systematic exploration of real attacks is crucial, as it can reveal weaknesses and vulnerabilities in deployed systems, and provide valuable insight that can lead better solutions. This opinion is also shared among other security researchers [107, 167].

Nonetheless, we designed our experiments such that we minimize the impact of our research and preserve the privacy of these users. First, we never retained verbatim copies of sensitive information, besides the photos that we clearly needed for running the experiments. Secondly, our attack can optionally issue friend requests with the purpose of expanding the number of accessible photos. However, we issued friendship requests exclusively to reach the 50-friends

[8]Calculated using the binomial probability formula used to find probabilities for a series of Bernoulli trials.

threshold, required by Facebook to trigger the SA mechanism. We never took advantage of accepted requests to collect photos or other private information otherwise unavailable; we solely collected public photos. In particular, in §3.3.2 we simulated a determined attacker, by assuming he has obtained access to all the photos (both public and private) needed to launch the attacker under ideal conditions. We simulated these conditions using publicly-available photos.

3.5 Remediation and Limitations

Facebook has already devised some mechanisms that aim at hindering casual attackers and the practices presented in this chapter. We explain why these mechanisms are not very effective or have some drawbacks that make them impractical. We continue with some proposed modifications to SA to make it safer based on the insights we obtained through our experiments. In the next chapter we present our approach to redesign the social authentication mechanism.

3.5.1 Compromise Prevention and Notification

Facebook has recently deployed some security features that can help further defend against stolen credentials being used for compromising accounts. However, these mechanisms are opt-in and disabled by default. Therefore, users may not have them enabled, and will remain susceptible to the threat that we study in this chapter.

First, users can add certain devices to a list of recognized, trusted devices. Whenever a user logs in from an unrecognized device, a security token is sent to the owner's mobile phone. This token must be entered in the log-in form for the user to be successfully logged in. This security setting, called login approval, follows the traditional second-token authentication scheme and only works in combination with the recognized, trusted devices feature. This approach can deter our attack, because it implements a truly-strong, two-factor authentication: the adversary would either need physical access to the user's mobile phone to obtain the security token and successfully login, or to have compromised the mobile device as was the case of the Eurograbber malware [10].

Second, a user who fails to complete an SA challenge is redirected to an alert page, upon the next successful login, which reports the attempted login, and shows the time and place information. Unfortunately, if the adversary manages to solve the SA test in a subsequent attempt, he will be redirected to the notification page and the account owner will never see the alert. In addition to the default notification, users may enable an optional login-notification feature: Whenever their account is accessed, an alert message is sent via text or email message. This notification feature does not prevent an adversary from logging in and, therefore, does not prevent our attack, which takes less than one minute. Furthermore, if the adversary has compromised the email account—which is not an unrealistic assumption, as users may reuse their credentials across services—he can delete the notification email. If that is not the case, the adversary will still have access until the owner changes the password and terminates any illegal active sessions.

Moreover, these mechanisms present three additional drawbacks. First, users must link their mobile phone number to their Facebook account, which many may not wish to do. Second, and more importantly, users typically access their account from many devices some of which may be public (e.g., computers in libraries or their workplace). In this case, adding all

these devices to the list of trusted devices is both impractical and insecure, and users will not wish to receive alerts every time they log in from one of those machines. Finally, involving the cellular network may result in monetary charges, a factor which could seriously discourage users from opting in to the mechanism.

3.5.2 Slowing Down the Attacker

When the attacker is prompted with an SA challenge, he must solve a CAPTCHA before the actual SA test. Although this topic falls outside the scope of this thesis, it is worth noticing that solving a CAPTCHA is trivial and only takes a human a few seconds. In addition, as previous work [103, 107, 109] has shown, breaking CAPTCHAs automatically is feasible and, in many cases, easy. Furthermore, it is well known that adversaries can perform laundry attacks [93, 154] and crowd-source the solution of CAPTCHAs. In conclusion, CAPTCHAs may create a technical obstacle to automated attacks, but they should not be considered a definitive countermeasure.

The presence of suggested names in SA tests is the major disadvantage of the current implementation as it greatly limits the search space for adversaries. By removing suggestions, there is a high probability of face-recognition software returning multiple users with similar confidence scores. Also, the time needed for face recognition might increase for certain systems although, as we have shown, cloud-based face recognition systems are unlikely to be seriously affected. On the downside, it will be harder for users to identify their friends and the system will be less usable as one would have to manually type the exact names of his friends. Automatic "type ahead" features may lessen the burden, although they are still vulnerable to exhaustive enumeration.

3.5.3 SA revisited

Designing effective and usable CAPTCHAs [108] is as hard as designing effective and usable authentication schemes that exploit social knowledge [173]. The downside of CAPTCHAs is that they are either too easy for machines or too difficult for humans. However, we believe that SA tests can be more secure yet still solvable by humans, and we present our user study and design and implementation guidelines in Chapter 4.

Chapter 4

Revisiting Social Authentication

Even though SA is a promising approach, as it offers a user-friendly mechanism to strengthen the login process, we have demonstrated that users are vulnerable against adversaries that employ face recognition software. Here we demonstrate that SA is also vulnerable to an attack that previous work has overlooked. In this attack, the adversary first builds an offline collection of the photos uploaded by the victim and his online friends. When presented with a SA challenge, the adversary identifies the photos within the collection via image comparison, and uses the associated tag information to pass the challenge. Compared to the previous attack, this attack has an important advantage: the identification of photos within a collection is far more accurate than face recognition, and effective even if no faces are in the photo.

In this chapter we revisit the concept of SA with the goal of building a system that retains the usability of the existing mechanism, while being robust against these attacks. We conduct a user study that provides us with valuable information regarding a critical aspect of SA; the ability of users to identify their friends in photos taken under realistic, non-ideal conditions.

We create a social application that replicates the original SA setup deployed by Facebook, yet with multiple levels of "difficulty". It processes the photos with state of the art face recognition software and uses the output to categorize them as "simple", "medium" or "difficult", based on the quality of the faces found (if any). The original SA picks photos categorized as "simple". Instead, we leverage the "medium" and "difficult" categories to explore the ability of users to identify faces at strange angles, or to rely on visual clues and associate them with friends to pass the challenge. Indeed, users successfully pass over 99% of the "medium" and 82% of the "difficult" challenges, indicating their ability to identify their friends even when their faces are not present, based on secondary features (e.g., posture, hair), associative information (e.g., pets, objects relevant to hobbies), or memory retrieval (users remember having seen the photos). On the other hand, face recognition software fails to associate the depicted faces with the respective users.

Based on our experimental results we set the guidelines for designing a secure, yet usable SA scheme, that renders the aforementioned attacks ineffective. First, a secure SA must select photos categorized as having a high probability of containing a human face, yet with poor quality features that render facial models unreliable but can still be identified by human solvers (i.e., "medium" photos). Second, a secure SA must also transform photos in such a way that the original photos are useless even if the attacker uses sophisticated image comparison techniques instead of face recognition software.

Our prototype applies the following transformations: first it superimposes the selected

("medium") faces over the faces of a "background" photograph. The overlayed faces are transparent and blend in with the underlying faces to obfuscate any areas that can be matched to the initial photos. Then, a "perspective transformation" is performed on the photo, which hinders even highly resilient pattern matching approaches, like template matching. To demonstrate our robustness, we employ three template matching methods and simulate the offline collection attack. Two of the methods fail to pass even one of the challenges with two faces, and the third passes less than 1%, with the attack requiring 3 orders of magnitude more processing effort than against the non-processed photos. Furthermore, after our transformations are applied, face detection software fails to detect even a single face.

Finally, we discuss the applicability of our approach as a security service offered by an OSN to other websites. In our vision, this mechanism can be adopted by web services as a user-specific CAPTCHA service, or even by high value services (e.g., banking websites) as an additional security measure to two-factor authentication. We discuss its robustness against attacks that break traditional CAPTCHAs, like outsourcing attacks, and argue that it is a user-friendly and secure alternative to existing schemes.

4.1 Attacks against Social Authentication

The concept of photo-based authentication in OSNs was first presented in 2008 by Yardi et al. [248]. In 2010 Facebook deployed its SA application in an effort to prevent adversaries from using stolen credentials. In a nutshell, when a login attempt is considered suspicious, the system presents the user with a series of 7 pages, each containing 3 photos of a friend and 6 potential answers. The user is required to correctly identify the depicted friend in at least 5 pages for the login to be allowed.

Face Recognition Attack. In the previous chapter, we demonstrated that practically anybody can answer SA challenges, by collecting publicly available data and employing off-the-shelf face recognition software. By issuing friend requests towards the victim's friends, one is able to gain access to the photos used to create the SA challenges. These photos and their respective tags are used to train face recognition classifiers, which can identify the friends depicted in SA challenges with high accuracy. Our estimations show that 84% of Facebook users are susceptible to this attack.

Offline Collection Attack. SA challenges contain photos that are picked from the albums of the victim's friends, or photos that contain tag information identifying the victim's friends. Attacking SA does not need to rely on face recognition software, as more effective photo matching techniques can be used instead. After infiltrating the social circle, the attacker can create an offline collection with all the victim's friends' photos he can access, along with the tag information. When SA is triggered, the attacker identifies the presented photos within the collection, and uses the tag information to answer the SA challenge.

The advantage of this attack is its effectiveness even in cases where the presented photos contain faces that cannot be identified via face recognition. Regardless of the content (even if no faces are present), the adversary can pass the challenge if any of the photos are in the collection. Several image-analysis techniques could be used for this attack. We demonstrate the effectiveness of such an attack, with all our experiments having been conducted on a machine with an 8-core Intel(R) Xeon(R) E5440 @ 2.83GHz processor with 8 GB of RAM.

We build offline collections of varying sizes, and create 10 SA challenges from each collection. The collections are up to 40,000 photos, which is higher than the average number of photos a

Collection size	5K	10K	20K	30K	40K
Identified photos	20.8	20.9	20.5	20.3	20.9

Table 4.1: Avg. number of identified photos for offline collection attack.

user and his friends have (see Section 4.2.2). We employ a simple image comparison approach to identify the presented photos within our collection. To improve the performance of the attack, we crop the top left corners of the presented photos, and match them with the photos in the collection. In certain cases this might result in false positives (e.g., the top left corner of the photo has only black pixels), however it did not affect our success rate, as all challenges had at least 5 pages where all 3 photos were identified. Table 4.1 presents the results of our experiments, indicating the average number of photos identified correctly in each SA challenge (21 photos). In all experiments, we successfully identified at least 19 photos, and in all scenarios had an average of at least 20.3 photos (96.7%).

Our approach is quite efficient as we are able to identify the 21 photos within 10K photos in 2.9 seconds ($\sim$ 0.52 per photo in a 40K collection). The most time-consuming phase is loading the offline collection in memory, which takes on average 887.8 seconds for every 10K photos. This does not pose an obstacle as it can be done before attempting to log into the account and having to solve an SA challenge. It is worth noting that, as our goal is to demonstrate the applicability of this technique, we do not improve the performance of the attack by intersecting the suggested names and the tags after each photo identified within each page.This could further decrease times, as the tags from one photo might be enough to infer the answer.

Also, we do not explore the aspect of photo coverage the attacker can achieve, as previous work (e.g., [103, 107, 162, 231]) has extensively demonstrated the effectiveness of employing fake accounts to befriend users of OSNs, and have reported success rates of up to 90%.

Concluding Observations. Based on the characteristics of these attacks, we argue that a secure photo-based SA mechanism should not rely solely on (i) the secrecy of the photos, or (ii) the original content of the photos. First, our intuition is that the human capability of recognizing faces based on contextual and noisy information of an image should be exploited, so as to surpass the limitations of current state of the art face recognition techniques. Second, challenge photos should contain a certain amount of natural or synthetic noise, to hinder the automatic identification of the photos within a collection, even when sophisticated visual pattern matching techniques are employed. A similar criterion is adopted by reCAPTCHA [236], which is currently the most secure CAPTCHA service employed web-wide: it exploits the fact that scanned characters contain natural noise, which state of the art image analysis techniques cannot handle (especially with additional synthetic noise).

4.2 Measuring User Abilities

To design a secure SA that exploits noisy and unidentifiable faces, we need to verify our intuition that humans are capable of recognizing their friends based on contextual information in photos taken under natural conditions. Although previous work [144, 209] has already explored the ability of people to discern *human faces* or their features, we are, to the best of our knowledge, the first to focus on the ability of *recognizing friends* (as opposed to unknown faces), even under conditions where the faces may not be clear or even present at all.

Measurement Application. We created a Facebook app that replicates the SA mechanism and also allows configuration options to select the "quality" of the photos used. We asked users to identify their friends in several SA challenges, and to complete an additional questionnaire for each photo. We opted for a Facebook application for two reasons: first, they inspire trust in users as they are deployed within a sandbox and are governed by a series of permissions that clearly state the user data accessed. Second, a Facebook application enables direct access to some user profile data (e.g., their social circle). This enables us to respect user privacy and minimize collection of potentially sensitive information, since we use data stored on Facebook rather than having users upload information to our own infrastructure.

We also setup a site to direct users there and provide details about our study, the purpose of the experiments and what data participants would grant us access to.

IRB Approval. Once our study was ready to commence, we issued an IRB protocol request to the review board of our institution, where we clearly described the parameters of our study, and the type of data we would be gathering. After our request was approved we invited users to participate. To explore the usability of our approach in the context of a OSN with a massive heterogeneous user base, we desired a diverse set of participants.

Recruiting Users. We explored and experimented with the possibility of reaching human subjects through the Amazon Mechanical Turk (AMT) service [89]. However, asking Turks to install an app, or directing them to a URL outside Amazon to fill out a survey, explicitly violates the AMT terms of services. Our tasks were rejected by Amazon because of this purely technical incompatibility. The nature of our system, where challenges are crafted specifically for each user, prohibited us from taking advantage of such crowdsourcing platforms where a large number of diverse participants can be found. Therefore we resorted to recruiting users directly by sharing our app with the online contacts of our personal Facebook accounts, as well as by posting flyers around the university campus. We also offered prizes as an incentive for user participation. This allowed us to collect and analyze a significant amount of user data, regarding over 4 million photos, and over 1,000 solved SA challenges.

4.2.1 Measurement Workflow

Once a user installs our app, it collects the social graph and related metadata (i.e., friends, URLs of photos, and tags). Our app processes collected photos with state of the art face recognition software to categorize them as "simple", "medium" or "difficult", based on the quality of the faces found. Photos of each category are selected by our app to build challenges of increasing difficulty and measure the user's ability to solve them.

Step 1: Face Extraction. We use the `face.com` online service, which has since been acquired by Facebook [17], because its effectiveness when using photos taken under realistic conditions has been demonstrated [202], and it performs better than other state of the art solutions [225]. We focus on two specific metrics assigned to the detected faces:

Confidence: when detecting faces, `face.com` returns its level of confidence (i.e., percentage returned by the classifier) that the tagged area actually contains a face. Tags assigned a low confidence level have a high probability of not containing a face.

Recognizable: not all faces are suitable candidates for training (or being recognized by) a classifier: `face.com` returns a boolean value to indicate this; "true" when faces can be recognized or are suitable to be used as part of a training set, and "false" otherwise. That means that even if a face is present in the tag, for various reasons (e.g., angle, obstacles), proper face-classification features (e.g., haar, eigenfaces, fisherfaces) cannot be extracted.

(a) Simple (b) Medium (c) Difficult

Figure 4.1: Samples of photos drawn from each category. Faces were blurred for privacy reasons.

Step 2: Photo Categorization. Based on these metrics, our app assigns photos to the following categories:

Simple - Figure 4.1(a): photos containing tags that most likely frame a human face. This is our baseline category, as it replicates the existing SA mechanism, and provides a reference to compare to our approach. According to a study in [202], 80% of the photos presented in SA challenges by Facebook had a face in the tagged area that was detectable by software. Therefore, we select photos in which `face.com` has detected faces with high confidence (confidence $\geq 80\%$), and which have been classified as recognizable (recognizable = "true").

Medium - Figure 4.1(b): photos likely containing a human face, but which are bad candidates for training or be recognized by face classifiers. We select photos with a high "probability" of containing a face (confidence $\geq 80\%$) but have been classified as bad candidates for training/recognition (recognizable = "false"). Such photos contain tags that users will be able to identify, but software will fail to.

Difficult - Figure 4.1(c): photos classified with a confidence below 40%. This category is to measure how effective people are at recognizing their friends even if their face is not visible in the photo. This could be based on their posture, their clothes, visible objects, etc.

Step 3: Photo Description. After a user selects the name of each depicted friend, our app informs them if they were right or wrong, and requires them to answer 4 questions, per photo, describing: the content, the position and visibility of the user's face and other faces within the tagged area, and reasons why the photo was useful or not. The questionnaire can be found in Appendix A.

4.2.2 User Study Results

Our goal is to measure the users' ability to recognize their friends, and demonstrate that humans can solve this task in conditions where the automated attacks described in Section 2 would fail, as we show in Section 4.3.4.

Collected dataset and demographics. 141 users installed our app. From them, we collected a total of $4,457,829$ photos and $5,087,034$ tags and classified them in the 3 categories, as summarized in Table 4.2. However, 90 of them actually completed challenges, out of which

Type	Total	Mean
Photos	4,457,829	31,615
Tags	5,087,034	36,078
Simple	2,066,386	14,655
Medium	593,479	4,209
Difficult	820,947	5,822
Remaining	1,606,222	11,391

Table 4.2: Summary of the collected user data, and the tags that matched our selection criteria.

Type	Total	Passed	Success	Per User
Simple	362	358	98.89%	3.98
Medium	347	344	99.14%	3.81
Difficult	335	275	82.09%	3.68
Total	1044	977	93.58%	11.47

Table 4.3: Number of challenges taken from each category, and percentage of successfully passed ones.

79 were listed as male and 11 as female, from 6 different countries. Of the 82 that reported their age, 63 were between 20 and 30 years old and 15 were between 30 and 40. On average, users had 347 friends each.

Recognizing Friends. Table 4.3 presents the number of challenges (each containing 7 pages with 3 photos of the same user) per category, and the percentage of challenges that users were able to "pass" (i.e., recognize at least 5 friends correctly out of 7). Figures 4.2 and 4.3 present the performance of individual users in medium and difficult tests as well as individual test pages. We can see that while users might struggle to identify certain individuals in a test page, overall, the flexibility of a test (5 out of 7 correct answers required) yields surprisingly high and consistent results for "medium" difficulty challenges' with over 99% success rates. Thus, *even in photos with faces that cannot be identified by state of the art face recognition software, users' success rates are not impacted.* Moreover, in the case of difficult challenges, performance is more than acceptable with an 82% success rate.

Influence of the Social Circle Size. Figure 4.4 shows the number of friends that a user has and the success rate for solving SA challenges. Each point refers to the overall success rate of a user for all 3 categories, and the label indicates the total number of challenges that the user completed. As the number of friends increase, we expect users to score significantly lower. However, the results do not demonstrate such an effect, and no strong correlation is evident. We believe that the suggestion of names has a significant effect, because users can rely on content that can be associated to certain friends. *This result is quite encouraging, as it negates concerns regarding the applicability of SA for users with a large number of friends.* Here we only visualize users that have completed at least 3 challenges for reasons of visual clarity, without the overall distribution being affected in any way.

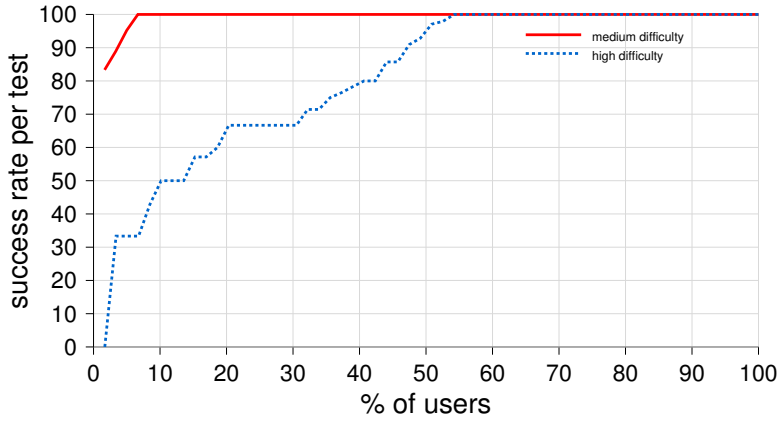


Figure 4.2: Recognizing friends: percentage of medium and difficult tests solved per user.

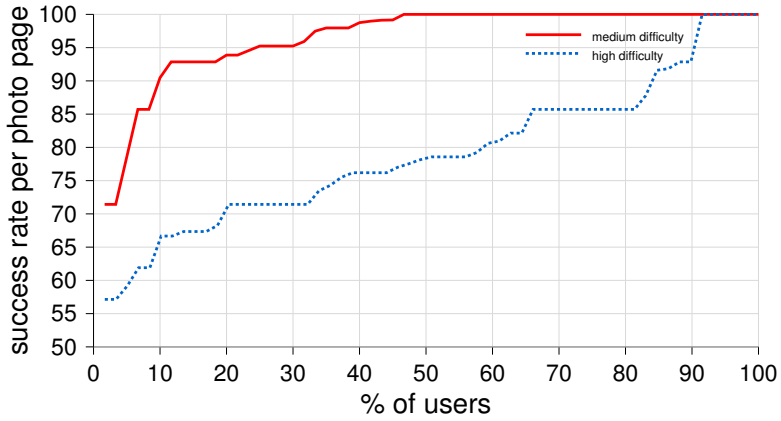


Figure 4.3: Recognizing friends: percentage of medium and difficult test pages (photo triplets of the same individual) solved per user.

Photo Content. Figure 4.5 shows the distribution of answers regarding the content of the photos, which offers an initial evaluation of the quality of the photo-detection process. As expected, for the "simple" and "medium" categories, the vast majority of photos (over 80%) are labeled as portraits, meaning that the focus of the photos are human faces. In contrast, in the "difficult" one they account for 37%. These numbers verify how accurate the face detection process of `face.com` is, as the confidence levels we have set in our photo categorization process(Section 4.2.1) are verified by users. About 10% of the remaining photos are landscapes, in which people may also be present but are not the main subject (e.g., large crowds). Photos of the "difficult" category are more evenly distributed.

Face Position. Figure 4.6 plots the distribution of answers about the placement of the friend's face with respect to the tagged area. Our tag selection process for the "medium" category returns photos that contain a clearly visible human face inside the tagged area in 49% of the cases (InClear) and an unclear face in 27.8% (InUnclear), cumulatively approaching the 80% confidence criteria we have set. The "simple" category contains a face in the tagged area

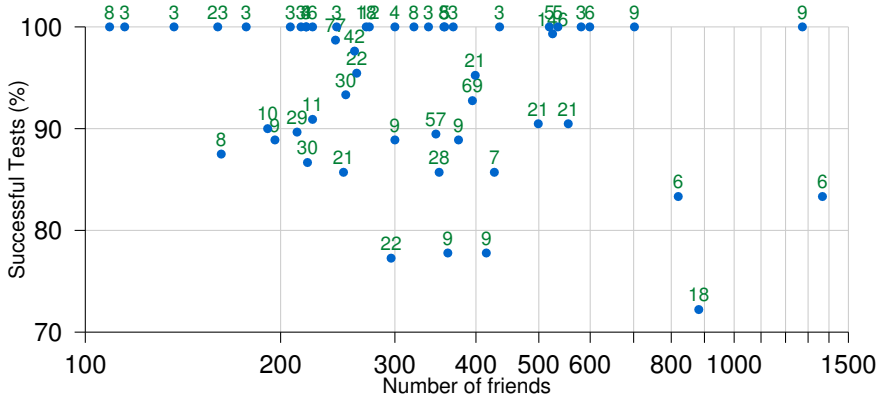


Figure 4.4: Correlation between number of friends and percentage of challenges successfully passed. Each point's label indicates how many challenges the user has taken.

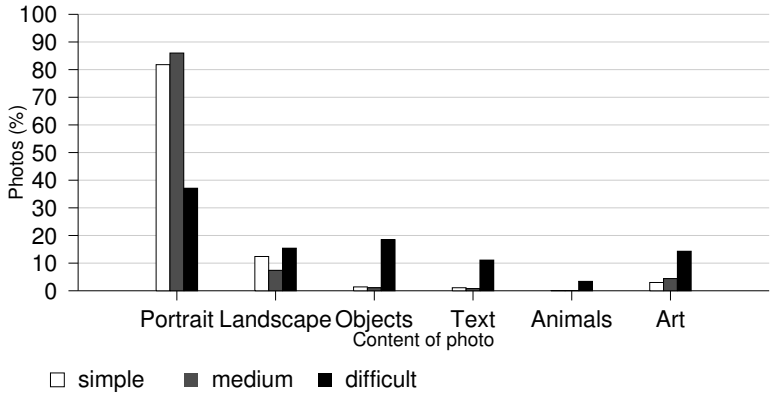


Figure 4.5: Photo content: distribution of photos regarding the type of their content.

in 80% of the cases, but has a slightly higher percentage (57.7%) of clearly visible faces. The "difficult" category, that selects photos with a low probability of containing a face, returns photos that do not contain the friend's face (Absent) in approximately half the photos.

Presence of Other Faces. Figure 4.7 shows the distribution of answers about other faces being visible in the photo, and their placement in regards to the tagged area. The "simple" and "medium" categories contain some other face in 83% and 77.5% of the photos with faces being outside the tag in 41% and 45% of the cases respectively. For the "difficult" category, 43.5% of the photos contain no human faces (Nobody).

Usefulness of the Photo. Figure 4.8 plots the distribution of photos regarding their usefulness. Users answered for about 70% of the "simple" and "medium" photos that the selected friend was present in the photo, which is less than the percentage of photos containing the friend's face according to Figure 4.6. This is due to users selecting other options such as "remembering the photo" or "content relevant to this friend", even though the friend's face is in the photo. An interesting effect of the selection process in the "difficult" category, where

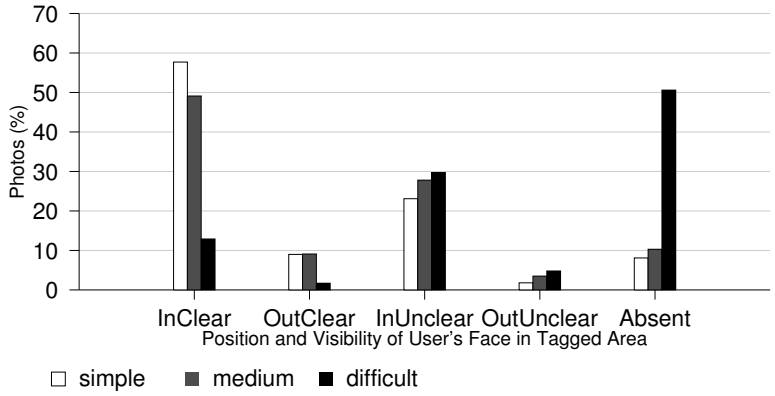


Figure 4.6: Face position: distribution of photos, based on the position of the friend's face in regards to the tagged area square.

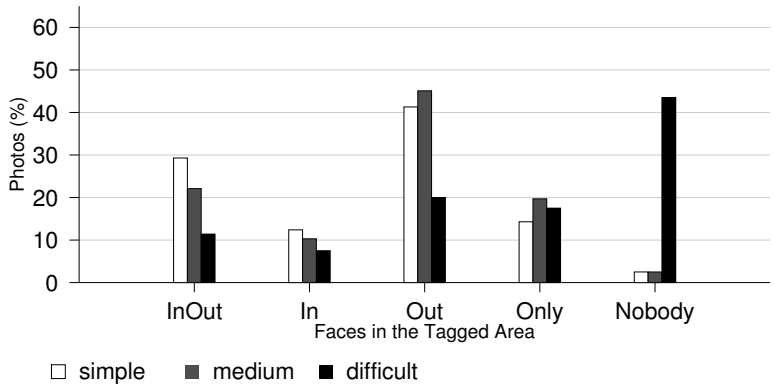


Figure 4.7: Presence of other faces: distribution of photos, based on the existence of other people's faces and their position in regards to the tagged area square.

photos have a low probability to contain the user's face, is users relying on other information to correctly select the friend. As seen in the figure, this category has a higher percentage of answers that rely on other types of visual clues and context for excluding (NoOneElse) or inferring (Relevant) suggested names.

Absolute Success Rate per Category. Table 4.4 shows the statistics for pages (each SA challenge contains 7 pages with 3 photos each) in which the users assigned all 3 photos to the same category. We present the percentage of pages in which the depicted friend was correctly identified, and the total number of pages in parentheses. An interesting result for portraits is that even in the "difficult" level, users identified their friends in 92.1% of the pages. *Thus, people can identify their friends in photos where software cannot even detect a face.*

In the "medium" category, users were successful in more than 97% of the pages, validating our initial intuition and the applicability of our approach. Even though the number of challenges for the other categories in the "difficult" level is not big enough to draw concrete conclusions, it is surprising that users achieved success rates of over 64% in all cases, and

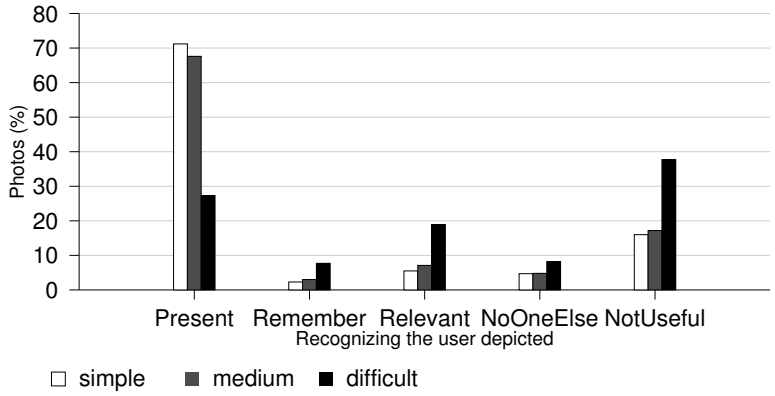


Figure 4.8: Usefulness of the photo: distribution of photos, regarding the reason they were deemed useful (or if they were not).

Type	Portrait	Landscape	Objects	Text	Art	Animals
Simple	97.4% (1133)	94.9% (59)	0% (1)	100% (1)	100% (2)	0% (1)
Medium	97.6% (1225)	90% (30)	0% (0)	0% (0)	75% (4)	0% (0)
Difficult	92.1% (267)	76.9% (26)	64.5% (31)	72.7% (11)	70.2% (37)	77.7% (9)

Table 4.4: Success rate per category: success rates (and total number of pages in parentheses) for cases in which all 3 photos were characterized to be of the same type, for each level of difficulty.

were even able to identify their friends in 77.7% of the pages that contained photos of animals. Thus, associative correlations assist users when presented with a photo of objects or pets.

Absence of Friend's Face. To analyze the ability of users to infer the correct answer even when the user is not present, in Figure 4.9 we break these numbers down based on the content. This is evident, as the cumulative ratios for the Landscape, Objects, Text and Art categories account for 44% and 55.5% of Relevant and Remember respectively. Thus, in almost half of the photos for which users rely on associative information or memory, the focus of the photos are not faces (Portraits).

In Figure 4.10 we focus on photos where the depicted friend's face was absent. When the users remember the photo, they are almost always able to correctly identify the friend (only in 1 case a user got the answer wrong). A very interesting result is the high success rate users achieve when selecting the depicted friend based on the relevance of the content, being 97.4% for the difficult photos. When users try to exclude suggested friends based on the content until they can select the correct answer (NoOther), they still achieve very high accuracy (albeit lower than the Relevant ones) with a combined 89.6% across the 3 difficulty categories. As expected, when the photo is not considered useful the success rates are lower. However, in the "simple" and "medium" categories, these photos belong to pages that were correctly answered in more that 82% of the cases, due to the high probability of the other photos containing a face.

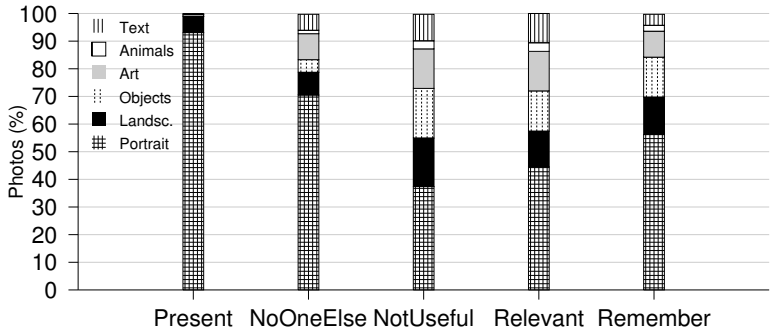


Figure 4.9: Usefulness of the photo: Correlation between content and usefulness.

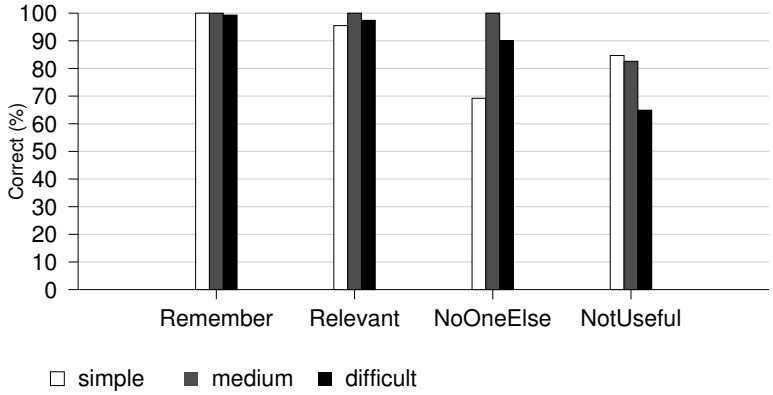


Figure 4.10: Usefulness of the photo: percentage of photos that were in pages answered correctly, while the depicted friend's face was Absent.

Total absence of faces. We wanted to explore whether the existence of other people in the photo might influence the results, i.e., even if the requested friend is not in the photo but other common friends are, the user might still be able to infer the correct answer. As we can see in Figure 4.11, where we focus on photos where the friend was absent, and no other faces were contained either, the results remain almost identical. However, the "simple" and "medium" photos that were flagged as NotUseful present a significant decrease (6.5%-16.3%). As can be seen in Figure 4.7, these categories have a much higher percentage of photos that contain other faces compared to the "difficult" category. Thus, even though photos might not contain the friend or other faces, the content can assist the user either in inferring the correct answer, or excluding suggestions until left with a plausible answer.

Exclusion and Inference. To explore the effect of other people's faces on users excluding suggestions or inferring the answer, we plot Figure 4.12, which provides interesting insight on the strategy users follow for identifying the depicted friend based on the presence of the friend's (columns with _F) and other users' (columns with _O) faces. Users tend to remember the photo when the depicted friend's face was either Unclear or Absent (Rememebr_F), as is the case for users inferring the correct answer (Relevant_F). Users also tend to remember

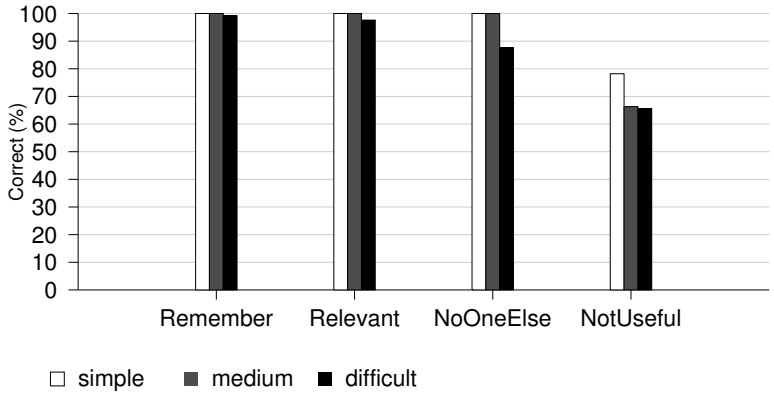


Figure 4.11: Usefulness of photo: percentage of photos that were in pages answered correctly, the depicted friend's face was Absent, and no other faces were contained.

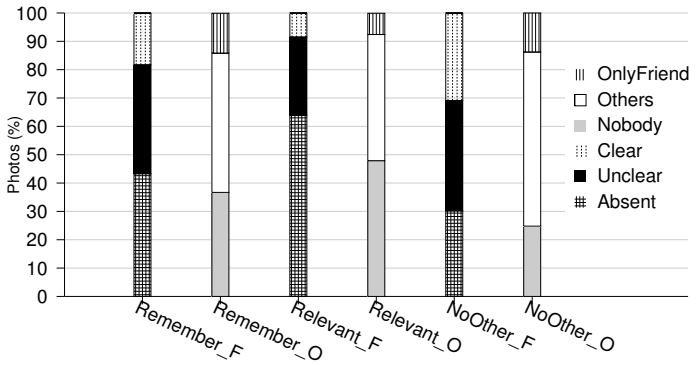


Figure 4.12: Breakdown of the photos that were useful for inferring or excluding answers, in regards to the friend's face (denoted by _F) and other people's faces (_O).

photos where other people (common friends) or no people are present at all (landscapes, pets and important objects). Furthermore, in Relevant_O we can see that users infer the correct friend almost equally from photos where Nobody (47.9%) is present (due to relevant objects and pets) or where Other (44.5%) people are present (people that the user knows are friends with the requested friend).

An interesting observation is that when it comes to excluding suggestions (NoOther_F), the absence of the friend's face (Absent) or its poor quality (Unclear) have a similar effect. However, the presence of other people's faces has a major impact, as it accounts for 61.4% of the photos in NoOther_O. We believe this is a strong indication that *when users are presented with photos of unknown individuals they tend to follow an approach of excluding suggested friends until left with a plausible answer.* If the users know the depicted people are acquaintances of the requested friend, they select the Relevant option. In the case of unknown individuals, they exclude suggestions of close friends and select less known contacts

that have a higher probability of being friends with people unknown to the user. Combined with the information that can be extracted from the other photos contained in a challenge page, this approach can be very effective, as users correctly answered 88.5% of all the pages that contained a photo for which NoOther friend matched.

4.3 Secure Social Authentication

Based on the results of our user study we proceed with establishing guidelines for creating a secure SA mechanism that is robust against the attacks we presented. The first phase deals with the selection of faces used, whereas the second deals with the form they are presented in.

4.3.1 Tag Selection

The first requirement for creating a secure SA mechanism is to establish the criteria for selecting the photos that will be used in the challenges. The goal is to filter out tagged faces that can be of use to adversaries that employ face recognition software, and select a subset of the remaining tags that have a high probability of containing a face.

Selection Criteria. We consider two characteristics for selecting tags for SA challenges. Our selection process creates a set of tags from photos that have been analyzed by face recognition software and have been assigned a low *recognizability* score and a high *confidence* level of containing a human face, i.e., "*medium*" tags. Even though we build our selection process using the `face.com` face recognition algorithm, our tag selection can be completed with any state of the art software. Once all the tags have been analyzed, the system will select the tags that could not be recognised by the software.

Stringent Tag Selection. Kim et al. [173] proposed that the selection of friends should be based on the social subgraph they belong to. For example, showing one friend from the subgroup "Work" and one from "Family", reduces the chances of the attacker belonging to all subgroups and, thus, possess the knowledge to pass the challenges. However, as aforementioned, previous work has shown that attackers can infiltrate the user's social circle with a high probability and gain information about most, if not all, social subgraphs.

The OSN can use several types of information to narrow down the dataset from which friends are selected to include in the SA challenges. Several metrics can be used to calculate the level of interaction, e.g.,the number and frequency of personal messages, wall posts, comments and likes exchanged between the users. In order to not aid the attacker by limiting the number of suggestions that seem plausible answers, friends from this narrowed set must have the same chance of being selected, and all suggestions must be from this set of friends. Based on the level of interaction we can alter the criteria thresholds of the photo selection process. The intuition is that when users are close, there is a much higher possibility of the user remembering the photo or having more contextual information for associating a photo without a face to a specific friend. Thus, after randomly selecting the friend, if the interaction level is high, the system selects photos from the "*difficult*" category. The times that a photo has been viewed by the user, or if the photo has been "liked", and how recent the photo is, can be taken into account when setting the threshold.

4.3.2 Photo Presentation

To defend against the offline collection attack, the tagged areas should not contain any areas identical to the original tagged areas of the depicted friends. This will prevent the attacker from identifying the photos in the collection and using the tag information to select the depicted users. In our approach, we will blend the faces that the user is required to identify with random faces on a "background" photo. The first step is to select N friends that we will present in the photo, and M tagged faces for each friend, which fulfill our photo selection criteria.

Tag Processing: If we simply overlay the tagged areas containing the faces onto a new photo, an adversary could still resort to the offline collection attack. To prevent this, we perform a sequence of transformations on the extracted areas. First, we rotate the face, an approach that has been demonstrated to impact face detection [144]. Second, we edit the tag's `alpha level` (A) to make it translucent and blend it with the underlying faces ($0 \leq A \leq 1$, where 0 is fully transparent). Thus, the tag will not contain any parts from the photos in their initial form, thereby hindering straightforward image comparison techniques.

Photo processing: Each challenge contains one photo. We select a photo that contains at least $N * M$ faces, and overlay the existing faces with the processed tags we created in the previous step. We then apply a perspective transformation to the image, which has been demonstrated to severely impair even complex feature or template matching techniques. According to Gauglitz et al. [142], "perspective distortion is the hardest challenge". The perspective transformation we perform, is variable by P, with P denoting the ratio by which the bottom of the photo is "compressed" from each side; e.g., for $P = 3$, the bottom is compressed from both the left and right by 1/3 of the photo. The user is presented with N drop-down menus, each one containing the name of one of the depicted friends, along with the names of $S - 1$ other friends. The user is required to correctly identify the N depicted friends. An example output, with $N = 2$, $M = 1$, $A = 0.6$ and $P = 3.2$, can be seen in Figure 4.13.

4.3.3 Prototype Implementation

We implemented a prototype of our approach, which comprises of a Facebook app for the data collection process, and a back end for the photo processing. We implemented the back-end in Python, using SimpleCV and OpenCV for the image processing, and SciPy and NumPy for matrix calculations. Our app relies on the Graph API to collect information about the user, his friends, tags, and photos. We require permission for accessing the user's social graph, friend list, and photo metadata. We do not collect the photos but only keep references (URL) to them, and fetch what is needed at runtime.

To create a SA challenge, the back-end first selects N distinct friends of the target user. For each friend, it finds M random tags of that friend, and fetches the corresponding photos. Then, it selects a random background photo. and searches it for faces. If any are found, the M random tags of the friend are placed on top of such faces. If no faces are found, the tags are placed randomly. The *tag processing* part randomly rotates and applies an alpha opacity filter that ensures that none of the original pixels of the tag are preserved. This is implemented with SimpleCV's `blit()` function, which takes the background photo, tag image, position of the overlay and opacity percentage as input, and returns a collage image. The rotation is implemented with `rotate()`, the perspective transformation is based on the `getPerspectiveTransform()` and `warpaffine()` functions of OpenCV.

Figure 4.13: An example output photo, with rotation, opacity, and perspective transformations performed.

Quality of tags: "medium" photos contain the face within the tag area in 77% of the cases, as seen in Figure 4.6. This is due to users tagging the photos and not centering the tag area over the user's face. Thus, we need a method to avoid mapping incorrect tag information to the extracted faces. We select 7,500 random photos and process them with face detection software to correlate faces and tagged areas. We calculate the distance between the center of the two squares and their position on a 2-dimensional Cartesian coordinate system. If multiple tags are present in the photo, we calculate the distance between all `(face, tag)` tuples, sort them incrementaly, and map each tag to the nearest unallocated face.

We present the results of our experiment in Figure 4.14. The center of the graph corresponds to the center of the detected face's area. Each point represents the center of the corresponding tagged area, and all distances have been normalized to the size of each photo. The three circles denote the areas that contain tags with a 5%, 10% and 20% normalized distance from the center of the faces area. The most dense area is around the center of the tag (meaning that many tags are usually pretty accurate) and tags are mainly skewed on the x-axis. 43.37% of the tags have the same center with the detected face. However, in certain cases the tagged area is at a distance of up to 90% of the size of the photo, which could be due to faces not being detected, or friends who are not actually in the photo being tagged and the tag being mapped to a different face. Facebook has recently implemented a tag-suggestion mechanism:

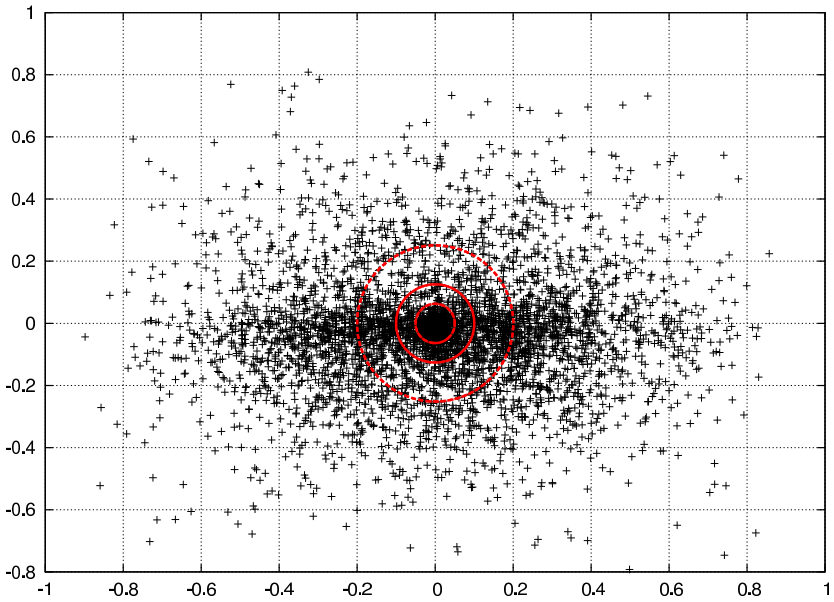


Figure 4.14: Distance between center of the user-generated tag and the center of detected face, for 7,500 photos (5, 10 and 20% distance boundaries are marked as red circles).

when the mouse hovers over a face in a photo, a perfectly centered tagging box appears which eliminates any errors on the user's side.

As our goal is to crop the area of a tag and superimpose that area on a new photo, we want to adjust the position of existing tags to coincide with the square created by face detection software, to improve any imprecise tags created by users. To avoid mapping incorrect tag information to faces, we use tags that have been mapped to faces within a distance of up to 5% of the photo's size. Thus, when extracting the tagged area, there is a high probability that it contains the corresponding user's face.

4.3.4 Security Evaluation

Offline collection attack. We evaluate the effectiveness of our approach against the offline collection attack as well, and not only face recognition attacks that focus on facial features. An attacker can rely on other features to automatically lookup a photo in a dataset, thus, obtaining the sensitive information required to solve the SA challenge. For our threat model, we assume the attacker has knowledge of our system, and has created a collection containing all the photos of the victim and his friends. We also assume he can apply the same categorization to photos as we do, and identify "medium" faces. Once presented with the challenge, the attacker will fetch all the "medium" tags for each of the suggested names. In our user study, we found that each user's friend has ∼12 "medium" tags. Thus, for a processed photo depicting one friend and with 6 suggestions provided, the attacker needs to only match one of the 72 tags with the photo to pass the challenge. Our attacking system employs 3 different *template matching* methods and also performs multiple rotations of the transformed photo to increase the detection rate. Specifically, we employ the normalized versions of the correlation coefficient algorithm (`CCOEFF_NORMED`), the cross correlation (`CCORR_NORMED`) and the squared difference algorithm (`SQDIFF_NORMED`).

To measure the impact of our transformations, we create two sets of 100 challenges each,

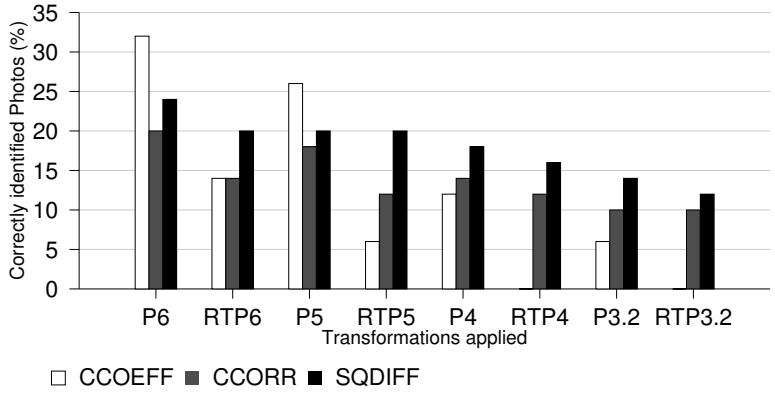


Figure 4.15: Percentage of photos where the tag was correctly identified, and the attacker could pass the challenge.

that contain one tag per photo; one set with tag transformations and one with tag and photo transformations. We also place the tags randomly in the photos, to explore the significance of the "background" photo. We maintain a constant 0.6 `alpha` level, and experiment by varying the perspective transformation ($P = 3.2, .., 6$), as shown in Figure 4.15. Results show that the perspective transformation (P) has a significant impact, but the tag transformations (RTP) contribute to the overall robustness. We achieve our best results when $P = 3.2$, with the highest scoring method (`SQDIFF`) identifying only 10% of the photos that have undergone all the transformations, compared to the 96.7%-100% success rate of non-transformed photos (see Section 4.1).

We manually inspected the identified tags and found them to be cases where they had been placed randomly on a position of the background photo with almost "no noise" (e.g., on a clear sky). Thus, we should strongly enforce a restriction of placing the tags on faces in the background photos, which would further decrease success rates. Apart from the impact on the success rate, the transformations require a significant increase of processing effort from the attacker. Attempting to match the tags to the transformed photo requires ∼319.6 seconds as opposed to ∼0.52 required for the attack against the non-transformed photos. This is more than the 5 minute window allowed by Facebook for a user to pass a SA challenge, which has 21 photos.

We repeat the experiment for $P = 3.2$ with 150 photos containing two tags ($N = 2$), where photos have to be compared to 144 tags (72 per depicted friend). `CCOEFF` and `CCORR` fail to identify both tags in any photo, and `SQDIFF` is only able to pass a single challenge. Processing a photo requires ∼ 626 sec. *Thus, we reduce the attacker's success rate to less than 1%, while requiring three orders of magnitude more processing effort on his part.*

We also explore the possibility of combining the results of the 3 methods for achieving better success rates, i.e., compare the output of each method and select the photo that receives the highest confidence out of the 3. This, however, is infeasible because in our experiments the "confidence" returned by `CCORR` is always higher than the one by `CCOEFF`, even when `CCORR`'s identification is wrong. Also, `SQDIFF` returns a differential result which is not comparable to the other two methods.

Face Recognition attack. Next we evaluate the robustness of our approach against face

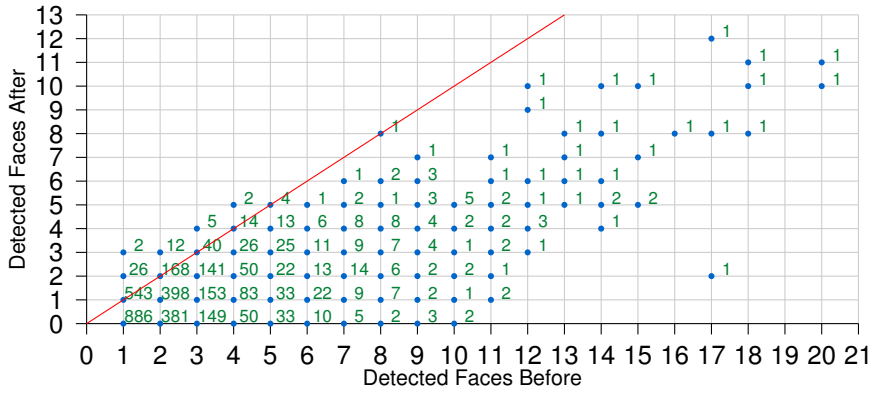


Figure 4.16: Faces detected before and after tag processing. The label of each point corresponds to the number of photos. The red line denotes the $X = Y$ axis.

`alpha level`	Faces	Transp.	Transp. & Persp.
0.8	176	11.3%	2.8%
0.7	180	8.9%	1.6%
0.6	180	4.4%	0%
0.5	175	4%	0%
0.4	180	2.7%	0%

Table 4.5: Face detection rate when applying transparency and perspective transformations.

detection and recognition. To explore how our tag selection and transformation process affects face detection, and by extension recognition, we calculate the number of faces detected in 3,487 "background" photos before and after we transform the tags and superimpose them on the background photo (no perspective transformation performed). We first detect the faces in the photo, then superimpose a transformed tag over every face and, finally, execute the face detection process again. We perform a conservative transparency transformation with an `alpha level` of $A = 0.8$. Figure 4.16 shows the detected faces before and after, and the label of each point indicates the number of photos with that (before, after) tuple. The red line denotes the $X = Y$ axis, upon which are the cases where the same number of faces are detected. Interestingly, even though the photos now contain double the number of faces, an extra face is detected only in 47 (1.3%) cases (points over the red line). Everything below the red line, indicates improvement as less faces are detected. Due to our tag transformations, no faces are detected in 43.6% of the photos, which significantly impacts face recognition, as faces have to be detected before compared to facial models.

While the rotation transformation increases the processing effort of the attacker, it cannot hinder an attacker as a stand-alone transformation, as the attacker can perform rotations upon the crafted photo to increase detection rates. Thus, we want to explore the combined impact of our two other transformations. To do so, for each experiment we create two versions of 250 transformed photos with 1 "medium" tag each. In the first version, we only apply the transparency transformation, and in the second both the transparency and perspective

transformation. We then manually remove any photos with tags that do not contain a human face, and use our face detection method to see how many faces are detected in the remaining tags. We test various `alpha levels` of transparency, with a constant value of $P = 3.2$ for the perspective transformation, as it had the best effect in our previous experiments. We present our results in Table 4.5. While the transparency transformation has a significant impact, by combining it with perspective transformation, *we are able to completely hinder face detection and, thus, recognition in all the photos, for* $A \leq 0.6$. Again, the detected faces had been placed on a "clear" background.

Security through distortion. Overall, our experiments demonstrate the robustness of our transformations against pattern matching and face recognition attacks, while creating photos that remain identifiable by humans. We achieve the optimal security results with $A \leq 0.6$, $N \geq 2$ and $P = 3.2$, with both attacks failing to pass even a single challenge, and tags remaining identifiable as can be seen in Figure 4.13.

4.4 Social Authentication as a Service

We discuss how an OSN can employ our secure SA mechanism to create a service for other websites. This can be employed as a user-gnostic CAPTCHA service, or as an additional security mechanism.

To outline the benefit of employing such a service in addition to a traditional two-factor authentication scheme, we describe the following scenario. An attacker steals a user's smartphone, which contains the credentials to an e-banking service and is also the device that receives the security token (via SMS or a token-generation app). Normally, the attacker will be able to complete any transaction as he possesses both tokens needed to pass the two-factor authentication. However, if the e-banking service employs this service for additional security, attackers that don't belong to the victim's social circle will fail to complete any transaction. Even if the device has an existing session with the OSN, they will not be able to pass the challenges (see outsourcing attacks below).

Usability Properties. A crucial metric for the applicability of a CAPTCHA scheme is based on the ability users have to successfully answer them. In our "medium" photo category, users correctly identified the depicted friend in 97.4% of the pages (which is lower than the 99.14% success rate for the complete challenges that require 5 out 7 pages to be passed). The authors in [108] followed an optimistic solving accuracy calculation to measure user success rates for image CAPTCHAs, and reported an 84–87% success rate, which is considerably lower than our rate. For the easiest category of text CAPTCHAs their results were comparable to ours, with 95–98%.

While we haven't conducted a user study to explore if our photo transformations will have a negative impact on the ability of users to identify their friends, previous work [150] demonstrated the ability of users to accurately identify heavily distorted versions of images they had seen before. We argue that users will not be impeded by our transformations, as they are not as distorting, users are familiar with the depicted subjects, and it is likely they will have seen many of the photos before.

Privacy. Deploying a user-gnostic CAPTCHA service may raise certain privacy concerns as it would result in Facebook acquiring information regarding websites visited by users. However, that information is also acquired through the "social plugins" or single sign-on services offered by many popular OSNs. These services have been widely adopted, and [175] reports that

more than 35% of the top 10,000 Alexa websites include the "Like" button.

Dataset. Another important feature of our system, is that it builds upon a dataset of photos and tag information that is readily available to the OSN. Thus, it doesn't face the challenge of creating a correct and extensive dataset as other image-based CAPTCHA schemes do [170, 233].

Security Properties. We discuss the effectiveness of our approach against typical CAPTCHA-breaking attacks.

Guessing Attacks. Our scheme allows automated bots to pass the challenge with a probability of $1/S^N$, where N is the number of friends depicted and S the number of suggestions per friend. The threshold adopted by previous work [251] is that bots should not have a success rate higher than 0.6%. By selecting 3 friends and providing 6 suggestions per friend, we are able to achieve an even lower probability of 0.46%. Furthermore, our system provides an extra level of security. As each CAPTCHA is created for a specific user, it is trivial to identify automated attacks that try to guess the answer. A legitimate user will request a new challenge when not able to identify the friends, without providing an answer, until presented with one he feels confident about answering. On the other hand, an automated script trying to guess the answer will continuously provide wrong answers until eventually guessing the correct one. Facebook can apply a penalty on accounts that exhibit such a behavior and ban them from the CAPTCHA service for a certain amount of time. Allowing a few failed or "bypassed" challenges, will be effective against automated attacks, without penalizing real users that might succeed after a few attempts.

Image Analysis Attacks. Various techniques from the field of image analysis have been demonstrated to be effective in automatically breaking CAPTCHA challenges. For example, OCR [103], support-vector machine classifiers [143] and projection-based segmentation algorithms [156]. We demonstrated the robustness of our tag selection and processing mechanisms against pattern matching and face recognition approaches.

Outsourcing/Laundering Attacks. Existing approaches create CAPTCHA challenges that are user-agnostic, i.e., are not created for a specific user. They are built upon the notion that virtually any human should have a high probability of successfully carrying out the tasks required to pass the challenge. However, these approaches are susceptible to outsourcing or "laundering" attacks [194], where adversaries relay the challenge to CAPTCHA-breaking services with human solvers that provide the correct answer. Our approach is robust against such attacks, as challenges are user-gnostic: they cannot be outsourced to human workers, as they wouldn't be familiar with the user's friends. Solving them would require the worker to first familiarize with the friends' faces. This, of course, is impractical as it would require too much time to pass a single CAPTCHA challenge (might not even be possible within the allowed time window).

4.5 Limitations

Re-processing Tags. Our tag-selection process relies on output from face recognition software, to select the subset of tags that cannot be automatically identified. To defend against advancements in face recognition that might overcome existing inefficiencies and result in more tags being identifiable, our system needs to employ state of the art software and periodically re-process the selected tags with the newest advancements. This is required for filtering out tags and maintaining a set of 'un-recognizable" tags, for creating the challenges. We must

note, however, that the tag and photo transformations play a more significant role in the overall robustness of our approach.

Processed tag collection. The attacker could create a collection of processed tags and compare those to the presented SA challenge. However, various characteristics of our approach render it robust against such a scenario. First, the completely random background photo, which blends in with the tags, introduces significant noise which can't be predicted by the attacker. Second, the placement of the tag on a photo significantly affects the result of its perspective transformation. Finally, as the transformations' values can be selected from a range, the attacker would have to create a massive collection of processed tags with various combinations of transformations and backgrounds. Even then, identifying the tag might not be feasible. Also, when comparing with just 72 tags, our experiments required over 5 minutes for a single processed photo with one tag. Thus, such an attack could not be completed within a realistic time window.

Chapter 5

User Information: Harvesting for Personalized Attacks

Privacy leakage is one of the biggest problems of social networking, as users tend to share a large amount of personal information and activities. This information is not limited to a name, date of birth, religion and marital status. Participation in events, friend lists, groups and organizations the user belongs to, preferences in music and food also reveal information about the life of the user. Articles revealing stories of employees losing their job or not getting hired due to information contained in their Facebook profile have gained wide attention [20, 63]. Even though in some networks users can fine tune their privacy settings (they have the option to share their personal information only with their friends, up to second degree friends, their network, everybody or nobody), information leakage still remains an important problem as many users do not always fully comprehend the implications of revealing personal information online [68].

Social networks have become a valuable resource for attackers. In earlier work it has been demonstrated that attackers can impersonate users in order to steal private information [103]. Privacy leakage attacks [247] can be exploited in many ways, such as revealing sensitive information for "high value" targets. One of the most sophisticated attacks based on harvested private information is personalized phishing. In traditional phishing schemes, emails contain generic terms, such as "Dear user", "Dear customer", "Hello subscriber" etc., which may be considered suspicious by many of the targets. Personalized phishing follows a different approach. The emails are crafted in a way so as to look like they originate from a friend or a relative of the potential victim. This type of email is far more convincing than the classic 419s scams [218] as it directly addresses the recipient and appears to be sent from someone the victim knows, thus, taking advantage of the implicit trust shown by the victim.

Social networks are an enormous and ever expanding pool of information that can be used as a stepping stone for personalized phishing campaigns. We demonstrate that even by retrieving the most basic information, i.e. the name of the user, we are able to harvest millions of email addresses. We present two different approaches to harvesting; *blind harvesting* that aims to gather as many email addresses as possible, querying for names retrieved from OSNs in the Google search engine, and *targeted harvesting* that aims to gather email addresses and correlate them to personal information publicly available on social networking sites.

Using the *blind harvesting* methodology we were able to harvest, on average, 45 emails per name for the Facebook names and 25 emails per name for the Twitter nicknames. Our results

show that this approach can harvest more addresses than traditional harvesting techniques in a highly automated, scalable way that requires little runtime and network overhead.

We present three *targeted harvesting* methodologies. The first uses the email-based search capability of Facebook. We collect names from highly populated Facebook fan pages and use the blind harvesting technique to search for email addresses. We then use the harvested email addresses in the Facebook search utility. If one or more profiles are returned, we check whether any of them have a matching name to the one collected from Facebook and map them to the email address in question. Such confirmation allows the use of personal information available in that profile to craft a personalized phishing email. This correlation technique can successfully link 11.5% of the harvested names with their actual email address. In order to improve the efficiency of the first technique, our second technique uses information from the Twitter network. By collecting <nickname,name> pairs from Twitter, we harvest emails with a prefix that is an exact match to the nickname and then search for them in the Facebook network. This technique can successfully correlate a user's profile with his email address for 43.4% of the profiles returned as part of this Facebook lookup. Our last technique relies on searching Google Buzz, (the predecessor of Google+) using the names of users collected from other social networks, to discover profiles and additionally crawl through their follower relations. Our experiments showed that 40.5% of the Buzz profiles we collected revealed the user's account name, which is also the user's Google mail account. Thus, by using our technique, one can harvest the actual email address of the targeted user and all the personal information that is revealed in their Google profile and Buzz profiles.

We argue that the mere participation in social networks opens users up to threats, and present novel attack techniques to highlight how attackers can take advantage of personal user information for deploying personalized attacks. We also highlight unsafe user practices that increase the vulnerability to such attacks. In section 5.5 we discuss several measures that can be employed to mitigate the harvesting of personal information from on line social networks.

5.1 Unsafe user practices

New technologies lead to new challenges. The massive adoption of online social networks by hundreds of millions of users around the world has led to the emergence of many challenges. Here we present why unsafe user practices in regards to the sharing of information in online social networks, can lead to a compromise of their privacy.

With users being attracted to OSNs, among other, for the ability to "socialize" with a large, geographically dispersed set of friends, as well as meet new people, users tend to befriend a much larger set of people than they would in the real world. With an average Facebook user having 190 online friends, many of whom are merely "cyber-acquaintances" [235] and posting a plethora of personal information that all of them can access, social networks are leading to the age of unprecedented public availability of personal information. However, users do not comprehend the dangers of revealing personal information to online contacts many of whom they have never met in the real world [84]. While social networks provide security mechanisms to block access to certain personal information, studies have revealed that users do not comprehend issues of online privacy. Therefore, a challenge is to educate users on matters of online privacy so as to comprehend that the exposure of sensitive information is potentially dangerous.

However, we argue that the mere participation of a user in a social network, renders them

vulnerable to attacks. We believe that the visibility of a user's participation in a network may offer enough information to attackers to make them the target of sophisticated personalized attacks. No matter how strict privacy settings may be introduced in the future, the names of almost all users will always be available to everyone. This is enough information for an attacker to use with a search engine and harvest email addresses faster and more efficiently than traditional harvesting techniques and, as shown in Section 5.3.2, map them to the owners' names. Alternatively, attackers can craft potential email addresses using variations of the user's name. Even though Facebook users can use the security settings to prevent their profile from appearing in search results, few users will use it. A social network where users cannot find other users is, by nature, not viable. Therefore, while names must be visible to all, their automatic extraction must be hindered, as we propose in Section 10.

Default settings for Facebook and Twitter allow everyone to view a user's name, friends and pages he is a fan of. A study conducted by Gross et al [147] revealed that only 0.06% of the users hide the visibility of information such as interests and relationships, while in [179] the authors report that 99% of the Twitter users that they checked retained the default privacy settings. Attackers that harvest this publicly available information can use it to craft personalized attacks that are far more effective than traditional attacks.

Overall, the attacks we present in this chapter cannot be prevented, but only mitigated. However, no matter how difficult it becomes for adversaries to extract user information, if users do not become aware of the potential threats, and treat any communication they receive with suspicion, these attacks will continue to plague users.

5.2 Harvesting email addresses

In this section we give a brief overview of the current methodologies used by spammers to harvest email addresses.

Web crawling. Email addresses of users are posted in various places on the Web. Personal web pages, blogs and forums are such examples. By crawling the web attackers can gather thousands of email addresses. However, this methodology suffers from low scalability as web crawling is a very time-consuming and bandwidth-demanding process.

Crawling archive sites. Attackers can narrow down their crawling to sites they know contain thousands of email addresses. For example, the Mailing List Archives site [57] hosts archives for thousands of computer-related mailing lists. The obfuscation used to prevent crawlers from extracting addresses is very simple to bypass, as addresses are written in the form "username () domain ! top-level-domain".

Malware. Attackers can instrument their malware code to collect addresses from the email clients of infected users or their instant messaging clients. Given the widespread use of email clients and popularity of instant messaging networks, this technique offers good scalability.

Malicious sites. Attackers can lure users and persuade them into registering at malicious sites, in exchange for access to desired content (e.g., adult content, software etc.).

Dictionary attacks. One can form email addresses by taking words from a dictionary. For example, the spammer can concatenate the word "john" with the domain "hotmail.com" and form the email address john@hotmail.com. Dictionary attacks can be classified into one of two types: blind attacks and search-based ones. Blind attacks try to guess email addresses by random concatenation of dictionary words and popular email domains. In this case, the

attacker would send spam to "john@hotmail.com" without any knowledge of the validity of the email address. This approach is not efficient and is limited to the dictionary size. Search-based attacks make use of Web search engines to validate the addresses acquired by the dictionary concatenation. The attacker now searches for "john@hotmail.com" and parses the results for email addresses. This approach is more efficient as it can return more addresses than expected. As an example, searching for "john@hotmail.com" can also lead to "other.john@hotmail.com" and "john@hotmail.de".

We describe a new approach to the way attackers can use information from social networks to perform more advanced search-based dictionary attacks. Instead of using words from a dictionary, an attacker can crawl popular social networks and use the collected user names or pseudonames as search keywords. This approach has two major advantages. First, it scales with the growth rate of social networks. While dictionaries are limited to few hundred thousand terms, the number of user names and pseudonames that can be found in social networks is in the order of hundreds of millions. Second, information from social networks can be used for personalizing spam campaigns. For example, attackers can use the full names of users in order to construct more convincing spam emails.

5.3 Using Social Networks for harvesting

Social networks provide a plethora of personal information. Users upload reports from their daily activities, political and religious status, events they have or will attend, photos, comments for other users and many more. Once a user has managed to become a friend with someone, he can extract various pieces of information that can be used for malicious actions.

Even though social networking sites cannot protect users from other malicious users that want to harvest personal information through social engineering tricks, they protect email addresses from automated harvesting. Before we describe how to use social networks as harvesting engines, we present the defensive measures taken by two popular social networking sites, Facebook and Twitter. Facebook does not reveal a user's email address to any user that is not contained in the user's friend list[1]. In case the harvester is in the list, the user's email address is presented as a GIF image to prevent automated extraction. Twitter, on the other hand, does not reveal a user's email address in any form. However, the personal information that is revealed includes the user's name, personal web page, location and a short bio description.

We identify and outline two different strategies that spammers may follow depending on the type of spam campaigns they wish to promote. First, we have spammers that propagate emails that contain advertisements for various products. This type of spammer will follow the *blind harvesting* approach which is the technique that will result in gathering as many email addresses as possible. Second, we have spammers that use spam emails to propagate scams, such as phishing campaigns. This type of spammer will opt to use the *targeted harvesting* technique that returns a much smaller number of results, but harvests information that can be used to craft very convincing personalized emails.

[1]This was the case in 2010, as Facebook no longer reveals the email to anyone, by default. However, in work conducted in 2013, we describe how to extract the email (see Section 7.2.3).

5.3.1 Blind harvesting

This technique aims to blindly harvest as many email addresses as possible in an efficient manner. The spammer does not care for personal information but simply wishes to gather email addresses. As shown by our results in Section 5.4.1, using social networks in conjunction with search engines is the most efficient method to harvest large numbers of email addresses.

We follow the same approach for both Facebook and Twitter to harvest email addresses. We initially crawl both networks to find names. As the structure and properties of the Facebook and Twitter networks differ, we have implemented two different crawlers for extracting names. An adversary might use the Facebook search utility to search for and harvest names. However a far more efficient way is to use Facebook fan pages. Users become fans of an artist or an activity, and anyone can freely browse all the names of a fan page. For example, at the time of this study, the fan pages of Madonna and Shakira (popular pop artists) had 1.3 and 1.7 million fans respectively, while Barack Obama had 8.8 million. Any attacker can visit a popular fan page, and will immediately have access to millions of names. In the case of Twitter we started from one initial account and then crawled the accounts the user follows, then the accounts they follow and so on. As we were interested only in the users' names and nicknames and not the actual tweets, this simple crawling is effective and fast for harvesting names.

Once the names have been harvested, they are used as terms in a search engine query. We used the Google search engine to locate email addresses. For each search term we query 8 different combinations ("term@hotmail.com", "term", "term@msn.com", "term@windowslive.com", "term@", "term at", "term@gmail.com", "term@yahoo.com") and for each query we retrieve the first 50 results. For scalability and efficiency reasons we do not open the URLs returned by the search engine. Instead, we parse the two-line summary provided in the results, for email addresses. This results in us missing a number of email addresses that may not be returned in the summary, however we remove a large overhead of having to parse the whole page. Our parser takes into account the various techniques used to hide email addresses from web crawlers, such as "username [at] domain".

5.3.2 Targeted harvesting

Attackers that rely on spam messages to propagate phishing schemes, can craft personalized phishing emails that are far more efficient than traditional techniques, by using personal information publicly available in social networks. Even though the blind harvesting technique can collect millions of email addresses efficiently, it presents a low probability of having these addresses matched to the name of their owners. The targeted harvesting approach links names to email addresses with a high probability, if not, absolute certainty. Furthermore, it enables the gathering of additional information that can render a targeted message much more convincing. Depending on the attack and the amount of personal information the attacker wants to collect, we describe three different methodologies for targeted harvesting.

Reverse lookup emails on Facebook. In the first case, we rely solely on the email-based search functionality of Facebook. Facebook allows users to search for other users based on their email address. We were surprised to find that even if the user has protected his email address through the privacy settings, and has made it visible only to him, his name will still show up in the search results when someone searches his email address. Only if the user disables his inclusion in public search results, we will not be able to find him using his email address. However, by default, Facebook includes users in search results. We collect names

from highly populated Facebook fan pages and use the blind harvesting technique to search for email addresses using Google. We then search for the harvested email addresses in Facebook and obtain the results. This way we have a pair of a user's profile and his email address (and any other information that is public), the basic information needed for a personalized phishing email. We can augment the collected information of the matched users by inviting them to become our friends. Once a user has accepted, we now have access to all the information posted in his Facebook profile. Our results from a series of initial experiments showed that 30% of the random invitations were accepted.

A major advantage of this technique is that it not only maps an email address to the owner's social profile, but also provides a technique for validating email addresses without the need of sending "probing" emails. When no profile is returned for a specific email address we cannot conclude if the email address is valid or not. However, when a user's profile is returned, we ascertain that the specific email is valid, since the user has entered it in his profile's contact information. Therefore, all the email addresses harvested using this technique are valid and eliminate the overhead of sending spam emails to many email addresses that are not valid. This is another advantage for spammers, since by eliminating all the emails that would be sent to invalid addresses and reducing the overall volume of the spam emails they send, they may be able to evade spam detection systems [242] that rely on the collection of a large number of spam emails.

Nickname-based Email Harvesting. In the second case we aim to use information that is available on Twitter in order to narrow down the search space of our first technique and improve its efficiency. This is done by using the nickname information available on Twitter. Many people tend to create a nickname that they consistently use across different domains and email providers. Our method crawls Twitter and collects name and nickname pairs. We then query Google using the nickname as a search term and extract email addresses that are an exact match (for example, if the nickname was "john_doe_1" we would only extract emails of the form "john_doe_1@domain.com"). This provides an association between a name and one or more email addresses. Next, we use the harvested email addresses as terms in the email-based search functionality of Facebook, exactly as in the first technique. Using this approach, one has to check much fewer email addresses than the first technique and, additionally, the success rate is higher as Twitter users will probably also have a Facebook account. The innovation of this technique is that it combines disjoint sets of personal information publicly available on different social networks and can be fully automated.

Site-aware Harvesting. In the third case, we employ Google's Buzz [39], a recently launched social networking service. In a nutshell, Buzz is a Twitter-like social networking service (based on follower/followee relations), along with content feeds and integration with other Google services (Gmail, Google Reader, Picassa, YouTube etc.). Each Buzz user has a Google profile page that contains basic information about him and his follower/followee relations. The Google profile page URL can either be based on the Google account username or a random long numeric identifier. The Google account username acts as a global identifier for all Google services, including the Gmail service. This means that if a user's Google profile URL includes his username and the user appears in the Buzz graph, then we automatically know his Gmail address. Thus, we can use the social graph of Buzz as a means to discover Gmail addresses. This approach has two major advantages. First, all harvested emails are valid. Second, and most important, for all collected email addresses we have the name of their owner, as we can extract it from the corresponding profile page. Moreover, since Buzz actually prompts the user to link and fetch content from other sites such as Twitter, Flickr, Google

Reader, YouTube, FriendFeed and LinkedIn, the attacker can enrich the amount and type of information assembled and utilized for the targeted spam campaign. We crawl Buzz profiles, through the Buzz search feature, by looking up names collected from Facebook and extract the follower/followee relations, wherever it is feasible. Additionally, references to unrelated profiles are returned by the search results as part of the indexed content. In the case where the user hides his relations, we are still able to process the profile contents, comprised of messages from and to other users. All names, that are rendered as clickable links to their respective profile pages, have their profile identifiers exposed. Even if Buzz decides to remove these links, effectively crippling the usability of the profile page, we could simply collect their names and look them up separately through the Buzz search feature.

5.4 Evaluation of harvesting techniques

Here we evaluate the proposed email harvesting techniques described in detail in Section 5.3. Furthermore we compare our techniques with the currently used approaches described in Section 5.2. Finally we perform a study regarding the use of harvested information in a spam campaign.

5.4.1 Blind Harvesting

We evaluate the use of our blind harvesting technique in comparison to current approaches. For obvious reasons we have omitted the malware and malicious site approaches from our comparison. Before proceeding to the analysis we first present and explain the comparison axes of our evaluation. We use three metrics:

- **Addresses-per-keyword ratio.** It is one of the most important metrics. A low ratio means that for each keyword queried the number of email addresses harvested is low. A high ratio means that the methodology can extract tens or hundreds of email addresses per keyword.
- **Traffic volume ratio.** Using search engines and sites for harvesting purposes requires downloading millions of pages. Downloading Gigabytes of data to harvest only a few email addresses decreases the scalability of the approach.
- **Automation.** Harvesting methodologies must be automated in order to be efficient. Although some approaches present high addresses-per-keyword ratio, they require manual intervention as they use information that does not expand and is located in multiple locations.

Address-per-keyword Ratio. Our first measurement evaluated the addresses-per-keyword ratio between our blind harvesting technique and four traditional harvesting methods: crawling archive sites, crawling the web for documents, a generic dictionary attack and a specialized dictionary attack. We crawled the MARC [57] and the W3C archive [83] sites to search for email addresses. For the document harvesting experiment, we only retrieved MS Word, Excel, Powerpoint and PDF documents as a step to narrow down our search space. For the generic dictionary attacks, we used keywords from an English dictionary [54]. For the specialized dictionary attack we used the 23,300 most popular English surnames [36]. For our harvesting techniques we extracted user names from Facebook and Twitter as well as user

	Dataset	Unique emails	Ratio
Facebook Names	82,383	3,706,493	1:45
Twitter Names	87,334	2,012,391	1:23
Twitter Nicks	31,358	784,099	1:25
Dictionary	146,973	3,630,071	1:24.7
Surnames	23,300	2,200,225	1:94
Documents	680,973	445,678	1:0.65
MARC	438,722	5,265	1:0.012
W3C	376,641	330,436	1:0.87

Table 5.1: A detailed listing of the dataset size and the number of unique email addresses harvested for each technique.

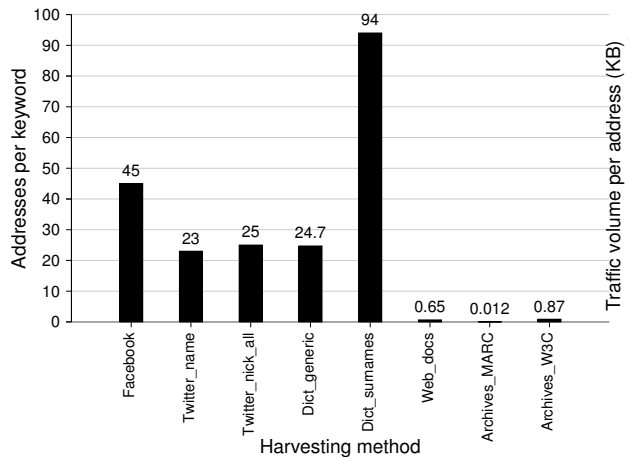


Figure 5.1: Ratio of unique email addresses per keyword for various email harvesting methodologies

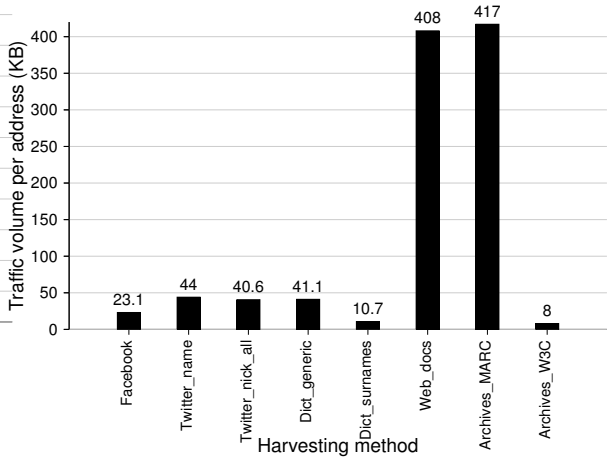


Figure 5.2: Ratio of traffic volume per email address for various harvesting methodologies

"nicknames" from Twitter. In all the experiments, we extracted all email addresses from the Google query results, and additionally evaluated the case where email addresses were an exact match to the Twitter nicknames.

The results are summarized in Figure 5.1. In the case of Facebook we extracted emails with a ratio of 1:45, i.e., we were able to harvest, on average, 45 unique email addresses per name queried. Using Twitter names, we achieved a ratio of 1:23, while a dataset of nicknames returned 25 addresses per query. The highest ratio observed was by the specialized version of the dictionary attack, which yielded 94 addresses per keyword. In fact, this methodology was expected to harvest a larger number, as it follows a similar approach but takes the most popular English names. However, this method suffers from scalability issues as described later in this section. The generic dictionary attack, contrary to the specialized one, achieved a lower ratio of 1:24.7. Crawling the web for documents returned 0.65 addresses per file downloaded. Finally, in the case of archive site crawling, the ratio for MARC and W3C archives is 1:0.012 and 1:0.87 respectively, where the ratio is defined as addresses extracted per page fetched. The low ratio for crawling sites is due to the download of structure pages, which are pages without any email address that contain hyperlinks to pages deeper in the site hierarchy. In fact, 96.7% of the MARC pages were structure pages as this site is deeply nested. The W3C

archive follows a more flat structure: 16.5% of the pages were structure pages. Ideally, if we exclude the structure pages, the ratios for the MARC and W3C archive become 1:0.4 and 1:1.05 respectively. Table 5.1 depicts the size of the aforementioned datasets, along with the count of harvested email addresses which produce the respected ratios.

Traffic Volume Ratio. Our second metric focuses on the cost per email address in Kbytes. The results are summarized in Figure 5.2. The traffic volume for the Facebook case is the number of names times the page size of Google results, that is 82,383 names times 130 Kbytes per Google result page times 8 (8 search combinations per name). The total traffic volume is around 79.8 Gbytes for approximately 23.1 Kbytes per email address. In the case we use names taken from Twitter, the ratio is 44 Kbytes per email address. When we use Twitter nicknames, the ratio drops down to 40.6 Kbytes per address. In the case of downloading office documents, the total volume of files was 181.6 Gbytes plus an additional 1.6 Gbytes for the Google queries, that is 408 Kbytes per email address. For the generic dictionary attack, we retrieved 142.3 Gbytes of search results which gives a ratio of 41.1 Kbytes per email address. For the specialized dictionary attack using popular surnames, we fetched 22.5 Gbytes of search results, that is a ratio of 10.7 Kbytes per email address. Finally, for the archive site crawling experiments, we downloaded 4.6 Gbytes, a ratio of 14.8 Kbytes per address in total. If we examine the two archive sites separately, the ratio for MARC is 417 Kbytes per address and for W3C is 8 Kbytes per address.

Automation. Our proposed harvesting technique is highly scalable. As we use information retrieved from social networks, our approach follows their growth rate. Therefore, our technique is fully automated as it expands, and no further manual intervention is needed for collecting more names that will be used as seeds. On the other hand, document crawling, generic dictionary attacks, and attacks based on surnames present very low scalability as the search terms are static, unlikely to change and have a limited dictionary size. Therefore, the process is semi-automated as customized crawlers have to be implemented for all new sites incorporated. Crawling mailing list archives presents medium scalability as we extract information from communities that expand, but that are interested in specific topics and expand with a much slower rate than social networks. This technique is also a semi-automated process, as most of the sites follow their own format to depict email addresses, and the appropriate regular expressions have to be written by hand.

Overall, while the harvesting technique that uses surnames presents a higher ratio for keywords per email and a smaller cost, it is not the optimal and most efficient one as it relies on a finite and limited dictionary that does not expand. On the other hand, while the *blind harvesting* technique exhibits a lower ratio and slightly higher cost, it has the advantage of being scalable, as it follows the expansion rate of social networks. *In the long run, we consider this to be the optimal solution for large-scale efficient harvesting.*

5.4.2 Targeted Harvesting

The second part of our evaluation focuses on our targeted harvesting techniques. Our experiment aims at measuring the effectiveness of these techniques for conducting personalized phishing campaigns. The results depict the percentage of names for which we can harvest at least one of their actual email addresses with each technique and therefore represent its effectiveness. We created two datasets containing randomly selected names from our databases. For reasons explained below, we selected names comprised solely of a first and last name, excluding middle names, dots or hyphens.

The first dataset contained 9000 names collected from a Facebook fan page. We used this dataset to evaluate our **first targeted harvesting technique**: for each name, we blindly harvested email addresses using the name as a search term in the Google engine and collected **any search results**. We then looked up the harvested email addresses using the Facebook search feature. If one or more profiles were returned, we checked whether any of them had a matching name with the one collected from the Facebook fan page and coupled with the email address in question. Overall, about 11.5% of unique names were associated with an email address that yielded a matching profile result from Facebook.

The second dataset was collected from crawling the Twitter network. For the **second targeted harvested technique** we wanted to measure the effectiveness of employing strict heuristics during the initial collection of email address through Google Search. For that matter, we included only **exact match results** of email addresses, i.e. only those whose prefix was identical to the Twitter username of the user being queried. Overall, using this strict Google search heuristic, we assembled 38986 <name,email> tuples, corresponding to 15627 unique names collected by our Twitter crawler. From those names, we selected 8,986 which did not contain middle names or special characters, just like in the first experiment. The reason for this filtering lies on the straightforward verification heuristic we employed; for each email address coupled with a name, we looked it up using Facebook search and, from any profile results returned, considered a match only if the name was exactly the same as the one in the dataset. Therefore, entries with middle names or special characters, having a larger possibility of being written differently across disjoint social networks, were excluded. The addresses were grouped by the Twitter nickname that resulted in their discovery. From the 8,986 users, 3,588 (39.9%) returned a Facebook profile and 1,558 (17.8%) were an exact match. Thus, 43.4% of the names, that returned a profile, had a user name that was an exact match to the Twitter profile name. By using a fuzzy string matching approach we could improve the success percentage. Let it be noted that names, that their harvested emails did not yield any Facebook results, may or may not be true positives of the targeted harvesting technique. As discussed in Section 6.4, additional OSNs could be employed to improve the query dataset. Also, in section 5.4.3 we present a study regarding the personal info collected from these profiles.

In comparison, the first and second methodologies, i.e., loose and strict collection of email address from Google search, may appear to be similarly effective with 11.5% and 17.8% of the names being a match. However, in the first case, a name is coupled with a much greater set of possible email addresses, requiring far more lookups in the Facebook than the second. In detail, in the first case, each name was coupled with an average of 104 email address, while, in the second case, only 4 address lookups took place for each name. Consequently, in the first case 0.2% of email address returned a profile result with a matching name, while in the second case the effectiveness climbed to 7%.

In regards to the **Google Buzz** approach, we used 1705 names and 850 of the most common English words (such as book, chair etc.) as search terms. We gathered a total of 59,680 Google profile URLs. 40.5% of the Google profile URLs (24,206 profiles) included the users' Google username, also used by default as their email address prefix, while the rest of the profiles were assigned random identifiers. This means that for each search term we gather approximately 22 Google profile URLs and around 9 valid Gmail accounts. As mentioned in section 5.3, all email addresses extracted from the profile usernames are valid Gmail accounts.

Label	Popularity
Current City	41.8% (667)
Hometown	38.8% (619)
Employers	24.9% (397)
College	24.5% (391)
High School	24.1% (385)
Relationship Status	21.0% (335)
Grad School	8.8% (140)
Birthday	3.9% (63)
Anniversary	3.4% (54)
Religious Views	2.5% (40)
Political Views	2.3% (36)

Table 5.2: Selected labels of personal information available on a Facebook profile page and their respective popularity among the matching profiles of the targeted harvesting evaluation.

5.4.3 Study of harvested personal info

Here we present a study based on the personal information publicly available in the Facebook profiles harvested from our second *targeted harvesting* technique. As mentioned in Section 5.4.2, 1,558 unique names were associated with a least one email address which yielded an exact-match profile match in Facebook, thus verifying the initial <name,email> association made by the Twitter crawler. Some of those names had more than one email addresses providing matching profiles. We investigated those cases and concluded that the profiles belonged to different people that shared the same name. Overall, 1,558 names led to 1,597 distinct profiles.

In Table 5.2 we present some selected labels of information, availabe on the Facebook profiles we harvested, which we consider to reveal personal information that can be exploited by attackers for targeted phishing attacks. For instance, one may use information about current employers or a person's studies to fake a workplace or college-related message. By adding such information, the email becomes more convincing and is therefore more likely to fool its recipient.

Subsequently, we proceed to examine the content of the Facebook profile, i.e., the page elements. We select the top 100 that appear more frequently among our dataset and apply a manual categorization. Table 5.3 summarizes the results. One may observe that items related to TV and cinema are the most common. An attacker could lure victims by crafting phishing messages to include references to such popular content. As shown by recent phishing campaigns [53], attackers use information regarding a victim's Facebook contacts, to impersonate their friends and trick them into giving them money. This type of attack could easily propagate to email phishing campaigns. To measure the feasibility of such attacks, we calculate the percentage of the harvested profiles which expose their respective friend lists. Overall, 72.6% of them, leak such information and the mean number of friends is 238.

5.5 Mitigation Techniques

In this section we provide a discussion on various measures that can minimize the public availability of personal information and hinder attackers from easily harvesting such information. While the defenses proposed can enforce users' privacy, we also refer to their potential negative

Category	Frequency
TV/Cinema	50%
Music	24%
Activity/Sports	10%
City/Travel	11%
Various	3%
Technology	2%

Table 5.3: Content categorization of the 100 most frequent items in a Facebook profile page.

impact on the functionality and expansion of social networking sites.

Server-Side Security. When proposing these measure, one must take into consideration that users of social networking sites are not restricted to "computer-savvy" people. In fact, the accessibility of such sites through mobile phones, smartphones and handheld devices allows the participation of people who do not even own a computer. For that matter, we consider the familiarity of users with computers to be minimal and their knowledge regarding information security and privacy matters to be negligible. Thus, even though educating the public to follow safe practices can improve the situation, it is our belief that effective privacy measures can only be enforced by the service and, therefore, propose only such. However, none of the solutions can completely hinder adversaries, but only mitigate the attacks.

Strict Privacy by Default. The first step that needs to be taken by social networking sites is to enforce strict default privacy settings. As shown by previous work [179], most users do not change default privacy settings and, thus, expose a large amount of information. For instance, Facebook's default settings reveal a person's real name, photograph, sex, relationship status, gender preferences, current city, hometown, biography, favorite quotations, current and previous employers, college and high school education, interests in music, books, movies and television and personal website. The e-mail address is not exposed but by searching for it, the person's profile will be returned. One should not be able to view any information from a user's profile other than his name if they are not friends in the specific networking site. If OSNs opt to hide all user information from third parties, attackers will not be able to harvest information for crafting personalized phishing attacks. On the other hand, features, such as email-address-based profile search, provide the necessary functionality for the social network to expand. Upon registration, a new user may use this feature to identify which of his e-mail contacts exist in the network and therefore instantly boost his networking degree.

Information-leakage Indicators. A variation of the first step is the preservation of standard privacy settings and the addition of indicators (e.g. icons, colors), that only the user can see, next to each profile field, illustrating information that is publicly available (e.g. any Internet user has access to it - colored red), available within the network (e.g. any Facebook user has access to it - colored orange), available within friends (e.g. any friend/contact of the user has access to it - colored yellow) and available only to the user himself (colored green). We believe that users will be very receptive to this concept as they will be able to instantly identify, through a glance at their profile, information that is exposed despite their will or knowledge. Nonetheless, social networking sites operate with the need for users to provide as much information possible about themselves. Such privacy indicators could scare the user into withdrawing a substantial amount of information.

Information rendered as Images. The next measure that can hinder attackers from harvesting names that can lead to email addresses, is to display names as images, just like the

way Facebook presents e-mail addresses. Displaying names as images raises the difficulty for extracting them, increases the error ratio on the attacker's side and does not break the users' experience. However, social networks can also provide a way for displaying names as plain text after verifying that the entity that issued the request is not a bot, e.g., by using CAPTCHAs. Unfortunately CAPTCHAs are not fool-proof. For instance, in [103], the authors were able to solve social networking site CAPTCHAs, including Facebook's reCAPTCHAs, through simple image processing techniques, combined with a dictionary and Google searches.

Automatic Tools Detection. Furthermore, OSNs should employ techniques that can detect accounts that are used by bots either to automatically issue friend requests for harvesting purposes or flood users with spam advertisements. Several services [11] are available and one is already being used by Twitter [75]. We believe this to be a major step in protecting users' privacy, since a large fraction of users accepts friend requests from unknown profiles. Therefore, all social networking sites must deploy such services.

Email Reverse Search. A major blunder on the side of social networking sites, is to allow users to search for profiles by using email addresses. By doing so, an attacker can easily map harvested email accounts to user profiles, and use the publicly available information to craft very convincing personalized phishing emails.

Use of nicknames. We believe that OSNs should exhibit the following behavior regarding the use of nicknames: if a user is logged in the site and is also in the contact list of the person using a nickname, he should be able to use the nickname directly (e.g., `facebook.com/nickname`). In any other case, the OSNs will prohibit its use, returning a "nickname not found" error. Instead of the nickname, a unique and random identifier will be used (e.g., `facebook.com/1309501319510`). This way, another user coming across this profile reference (e.g., in a fan page) will be unable to obtain the actual nickname and map it to an email address.

Chapter 6

Detecting Identity Theft

As the majority of users are not familiar with privacy issues, they often expose a large amount of personal information on their profiles that can be viewed by anyone in the network. As we demonstrated in the previous chapter, adversaries can easily harvest user information for personalized attacks. One category of such personalized attacks, is identity theft through *profile cloning*. In [103] the authors demonstrate an attack of profile cloning, where someone other than the legitimate owner of a profile creates a new profile in the same or different social network in which he copies the original information. By doing so, he creates a fake profile impersonating the legitimate owner using the cloned information. Since users may maintain profiles in more than one social networks, their contacts, especially the more distant ones, have no way of knowing if a profile encountered in a social networking site has been created by the same person who created the profile in the other site.

The usual assumption is that a new profile, claiming to be related to a pre-existing contact, is a valid profile; either a new or secondary one. Unsuspecting users tend to trust the new profile and actions initiated from it. This can be exploited by attackers to lure victims into clicking links contained in messages that can lead to phishing or drive-by-download sites. Furthermore, a cloned profile could be used to send falsified messages in order to harm the original user. The victimized user has no way of knowing the existence of the fake profiles (especially if across social networks). For that matter, we believe profile cloning is a silent but serious threat in today's world of social networks, where people might face consequences in the real world for actions of their (counterfeit) electronic profiles [7, 8].

In this light, we propose a tool that automatically seeks and identifies cloned profiles in social networks. The key concept behind its logic is that it employs user-specific (or user-identifying) information, collected from the user's original social network profile to locate similar profiles across social networks. Any returned results, depending on how rare the common profile information is considered to be, are deemed suspicious and further inspection is performed. Finally, the user is presented with a list of possible profile clones and a score indicating their degree of similarity with his own profile.

6.1 System Design

In this section we outline the design of our approach for detecting forged profiles across the Web. Our system is comprised of three main components and we describe the functionality of each one.

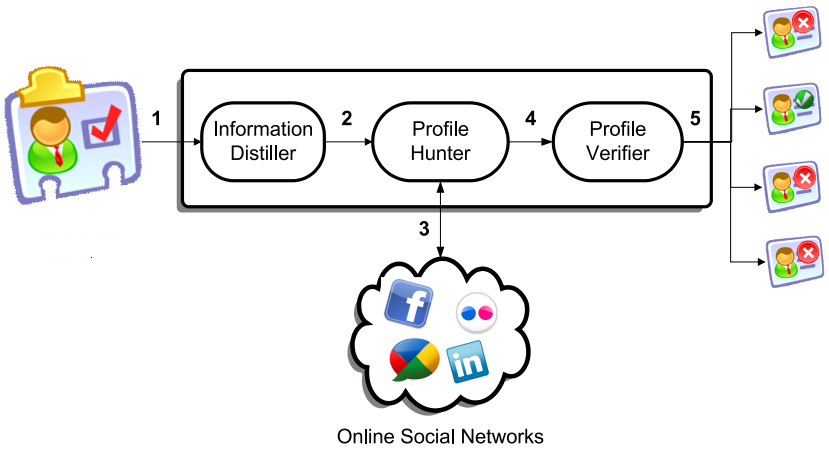


Figure 6.1: Diagram of our system architecture.

1. **Information Distiller**. This component is responsible for extracting information from the legitimate social network profile. Initially, it analyzes the user's profile and identifies which pieces of information on that profile could be regarded as rare or user-specific and may therefore be labeled as user-identifying terms. The information extracted from the profile is used to construct test queries in search engines and social network search services. The number (count) of results returned for each query is used as a heuristic and those pieces of information that stand out, having yielded significantly fewer results than the rest of the information on the user's profile, are taken into account by the distiller. Such pieces of information are labeled as user-identifying terms and used to create a user-record for our system along with the user's full name (as it appears in his profile). The record is passed on to the next system component that uses the information to detect other potential social network profiles of the user.

2. **Profile Hunter**. This component processes user-records and uses the user-identifying terms to locate social network profiles that may potentially belong to the user. Profiles are harvested from social-network-specific queries using each network's search mechanism that contain these terms and the user's real name. All the returned results are combined and a profile-record is created. Profile-records contain a link to the user's legitimate profile along with links to all the profiles returned in the results.

3. **Profile Verifier**. This component processes profile-records and extracts the information available in the harvested social profiles. Each profile is then examined in regards to its similarity to the user's original profile. A similarity score is calculated based on the common values of information fields. Furthermore, profile pictures are compared, as cloned profiles will use the victim's photo to look more legitimate. After all the harvested profiles have been compared to the legitimate one, the user is presented with a list of all the profiles along with a similarity score.

We can see a diagram of our system in Figure 6.1. In step (1) the Information Distiller extracts the user-identifying information from the legitimate social network profile. This is used to create a user-record which is passed on to the Profile Hunter in Step (2). Next, Profile Hunter searches online social networks for profiles using the information from the user-record

in step (3). All returned profiles are inserted in a profile-record and passed on to the Profile Verifier in step (4). The Profile Verifier compares all the profiles from the profile-record to the original legitimate profile and calculates a similarity score based on the common values of certain fields. In step (5) the profiles are presented to the user, along with the similarity scores, and an indication of which profiles are most likely to be cloned.

6.2 System Implementation

Here we provide details of the proof-of-concept implementation of our approach. We use the social network LinkedIn [55] as the basis for developing our proposed design. LinkedIn is a business-oriented social networking site, hosting profiles for more than 235 million registered users. As profiles are created mostly for professional reasons, users tend to make their profiles viewable by almost all other LinkedIn users, or at least all other users in the same network. Thus, an adversary can easily find a large amount of information for a specific user. For that matter, we consider it a good candidate for investigating the feasibility of an attack and developing our proposed detection tool.

6.2.1 Automated Profile Cloning Attacks

We investigate the feasibility of an automated profile cloning attack in LinkedIn. Bilge et al. [103] have demonstrated that scripted profile cloning is possible in Facebook, XING and the German sites StudiVZ and MeinVZ. In all these services but XING, CAPTCHAs were employed and CAPTCHA-breaking techniques were required. In the case of LinkedIn CAPTCHA mechanisms are not in place. The user is initially prompted for his real name, valid e-mail address and a password. This suffices for creating a provisionary account in the service, which needs to be verified by accessing a private URL, sent to the user via e-mail, and entering the account's password. Receiving such messages and completing the verification process is trivial to be scripted and therefore can be carried out without human intervention. To address the need for different valid e-mail addresses, we have employed services such as 10MinuteMail [1] that provide disposable e-mail inbox accounts for a short period of time. Once the account has been verified, the user is asked to provide optional information that will populate his profile.

We have implemented the automated profile creation tool and all subsequent experiments detailed in this chapter rely on this tool and not manual input from a human. This was done to test its operation under real-world conditions. Let it be noted that all accounts created for the purposes of testing automated profile creation and carrying out subsequent experiments have been now removed from LinkedIn, and during their time of activity we did not interact with any other users of the service. Furthermore, due to ethical reasons, in the case where existing profiles were duplicated, they belonged to members of our lab, whose consent we had first acquired.

6.2.2 Detecting Cloned Profiles

We employ the cURL [12] command-line tool to handle HTTP communication with the service and implement the logic of the various components of our tool using Unix bash shell scripts.

1. **Information Distiller**. This component requires the credentials of the LinkedIn user, who wishes to check for clones of his profile, as input. The component's output is a

user-record which contains a group of keywords, corresponding to pieces of information from the user's profile, that individually or as a combination identify that profile. After logging in with the service, this component parses the HTML tags present in the user's profile to identify the different types of information present. Consequently, it employs the Advanced Search feature of LinkedIn to perform queries that aim to identify those keywords that yield fewer results that the rest[1]. Our goal is to use the minimum number of fields. If no results are returned, we include more fields in an incremental basis, according to the number of results they yield. In our prototype implementation, we identify the number of results returned for information corresponding to a person's present title, current and past company and education. We insert the name along with the other information in a record and provide that data to the next component.

2. **Profile Hunter**. This component employs the user-record, which contains a person's name and information identified as rare, to search LinkedIn for similar user profiles. We employ the service's Advanced Search feature to initially find out the number of returned matches and subsequently use the protected and, if available, public links to those profiles to create a *profile-record* which is passed on to the next component. The upper limit of 100 results per query is not a problem since at this point queries are designed to be quite specific and yield at least an order of magnitude less results, an assumption which has been validated during our tests.

3. **Profile Verifier**. This component receives a *profile-record* which is a list of HTTP links pointing to protected or public profiles that are returned when we search for user information similar to the original user. Subsequently, it accesses those profiles, uses the HTML tags of those pages to identify the different types of information and performs one to one string matching with the profile of the original user. This approach is generic and not limited to a specific social network, as the verifier can look for specific fields according to each network. In our prototype implementation, we also employ naive image comparison. We assume that the attacker will have copied the image from the original profile. We use the convert tool, part of the ImageMagick suite, to perform our comparisons. In detail, to discover how much image 'A' looks like image 'B', we calculate the absolute error count (i.e., number of different pixels) between them and then compare image 'A' with an image of random noise (i.e., random RGB pixels). The two error counts give the distance between 'A' and something completely random and the distance between 'A' and 'B'. This way we can infer how much 'A' and 'B' look alike. To correctly estimate the threshold of error that can be tolerated, we plan on conducting a study where images will be manipulated so as to differ from the original photo but remain recognizable. The component outputs a similarity score between the original profile and each of the other profiles.

6.3 System Evaluation

In this section we evaluate the efficiency of our proposed approach for detecting forged social network profiles. First, we provide data from a study on LinkedIn regarding the amount of information exposed in public or protected[2] user profiles.

[1] Those with results in the lowest order of magnitude or, in the worst case, the one with the least results.
[2] To view the profile information, a service account is required.

Trace Name	Description	Profiles
surnames	Popular 100 English names	11281
companies	Fortune 100 companies	9527
universities	Top 100 U.S. universities	8811

Table 6.1: Summary of data collected.

6.3.1 LinkedIn Study

In order to understand how much information is exposed in public profiles of LinkedIn users, we compiled three distinct datasets of profiles and studied their nature. The idea is that an adversary seeking to perform automated profile cloning, can create such datasets and copy their information. Here we study the type and amount of information available for such an attack.

Table 6.1 presents those three distinct datasets. To do so, we created a fake LinkedIn account, that contains no information at all, and used the service's search feature to locate profiles that matched our search terms. In the free version of the service, the number of search results is bound to 100 but one can slightly modify his queries to count as different searches and at the same time return complementary sets of results. In our case, we used three lists as search terms to retrieve user profiles; one with the most common English surnames, one with the top companies according to Fortune Magazine [29] and one with the top U.S. universities.

Each of the ∼30K search results returned a summary of the user's profile, which we consider adequate information to convincingly clone a profile. As we can see in table 6.2, almost one out of every three returned search results is public and contains the user's name, along with current location and current title or affiliation. These profiles are accessible by anyone on the web, without the need for a LinkedIn account. In detail, in the *surnames* dataset 89% of the profiles has a public presence on the web. On the other hand, for profiles collected from the *companies* and *universities* datasets, public presence is merely 2.3% and 1.6% respectively. The big discrepancy is probably due to the fact that users from the industry and academia use LinkedIn for professional purposes and therefore set their profiles as viewable by other LinkedIn users only.

Table 6.3 presents the core profile information in all the profiles that are publicly available. Interestingly, besides the person's name, almost all public profiles carry information about the present location and relative industry. Additionally, about half of the profiles include a person's photo, current title or affiliation and education information.

In Table 6.4 we can see the information available in all the profiles that require a LinkedIn account for viewing. While the percentage of profiles from which we can access the user's photo is smaller compared to the public profiles, all the important information fields present a much higher availability. The fact that we cannot access the photos in many profiles is due to default privacy setting of LinkedIn where a user's photo is viewable only to other users from the same network. Nonetheless, an adversary could set his account to the specific network of the targeted victims in order to harvest the photo. Furthermore, all users reveal their location, and connections, and almost all their industry field. Most profiles from the surname dataset contain information regarding the user's current work status and education (86% and 70% respectively). The other datasets have an even larger percentage verifying the professional usage orientation of the users. Specifically, 99% of the profiles from the companies dataset contained information on current status and 92% revealed the user's education, and profiles

	surnames	*companies*	*universities*
Public Name	90.5%	2.5%	2.0%
Public Profile	89%	2.3%	1.6%

Table 6.2: Exposure of user names and profile information.

	surnames	*companies*	*universities*
Photos	47%	59%	44%
Location	98%	99%	99%
Industry	85%	97%	98%
Current Status	70%	86%	72%
Education	53%	66%	82%
Past Status	42%	54%	63%
Website	36%	50%	39%
Activities / Societies	21%	22%	55%

Table 6.3: Information available in public LinkedIn profiles for each dataset.

from the universities dataset stated that information in 94% and 99% of the cases. Therefore, any user with a LinkedIn account can gain access to user-identifying information from profiles in the vast majority of cases.

Our older study presented in Chapter 5.3 concerning the type and amount of information publicly available in Facebook profiles, demonstrated a similar availability of personal information. While those results showed a lower percentage of Facebook users sharing their information publicly, close to 25% of the users revealed their high school, college and employment affiliation, and over 40% revealed their current location.

As demonstrated from both of these studies, it is trivial for an adversary to gather information from social network accounts that will allow him to successfully clone user profiles. With the creation of a single fake account, an adversary can gain access to a plethora of details that we consider sufficient for deploying a very convincing impersonation attack. Even so, this information is also sufficient for the detection and matching of a duplicate profile from our tool.

6.3.2 Detection Efficiency

Initially, we evaluated our hypothesis that different pieces of information from a user profile yield a variable number of results when used as search terms, for instance in a social network's search engine. To do so, for each profile in our datasets, we extracted the values from different types of information and used them as search terms in the Advanced Search feature of the service. Next, we recorded the minimum and maximum number of results returned by any given term. Finally we calculated the range (maximum - minimum) of search results for information on that profile. Figure 6.2 presents the CDF of the range of search results returned for each profile in our dataset. One may observe a median range value of ∼1000 and also that only 10% of profiles had a range of search results lower that 20. Overall, we can see that the majority of profiles exhibited diversity in the number of search results returned by different pieces of information, and by leveraging this can be uniquely identified by the carefully crafted queries of our system.

Next, we conducted a controlled experiment to test the efficiency of our tool. Due to

	surnames	*companies*	*universities*
Photos	22%	52%	26%
Location	100%	100%	100%
Industry	94%	100%	100%
Connections	100%	100%	100%
Current Status	86%	99%	94%
Education	70%	92%	99%
Past Status	58%	96%	95%
Twitter Username	13%	0%	1%
Websites	41%	2%	1%

Table 6.4: Information available in protected LinkedIn profiles.

obvious ethical reasons we were not able to deploy a massive profile cloning attack in the wild. Thus, we selected a set of 10 existing LinkedIn profiles, that belong to members of our lab, and cloned them inside the same social network using the automated method described in 8.4. We then employed our tool to try and find the duplicates. Overall, we were able to detect all the profile clones without any false positives or negatives.

Finally, we used public user profiles as seeds into our system to try and detect existing duplicates inside LinkedIn. The Information Distiller produced user-records using information from current or past employment and education fields. Overall, we used 1,120 public profiles with 756 being from the surnames dataset, the 224 public profiles from the companies dataset and the 140 public profiles from the universities dataset. The Profile Hunter component returned at least one clone for 7.5% of the user profiles (in 3 cases our tool discovered 2 cloned instances of the profile). Our prototype system relied on the exact matching of fields and did not employ our image comparison technique to detect cloned profiles. Furthermore, similarity scores were based on the number of fields that contained information on both profiles (in several cases, one profile had less fields that contained information). After manual inspection, we verified that all detected profiles pointed to the actual person and that the score produced by the Profile Verifier was accurate. We cannot be certain if those clones are the result of a malicious act or can be attributed to misconfiguration. Furthermore, our prototype may have missed cloned profiles where the attacker deliberately injected mistakes so as to avoid detection. We discuss how our system can be improved in Section 6.4.

6.4 Future Work

The most important drawback of our system is that it currently uses only the LinkedIn social network. Our next step is to extend its functionality to utilize other popular social networks and create a profile parser for each network.

The next axis upon which our tool can be improved lies in the accuracy of comparing two profiles and assigning a similarity score. Our current implementation of the Profile Verifier looks for exact string matches in information fields when comparing two profiles. Instead of looking for exact matches we can use fuzzy string matching to overcome wrongly typed information, or deliberately injected mistakes. An important aspect of this is to correctly tune the fuzzy matching algorithm to match our needs. Since the presentation of the same information across OSNs may vary, we must implement information extracting functions specific for each social network, that extract the information and convert it to a custom

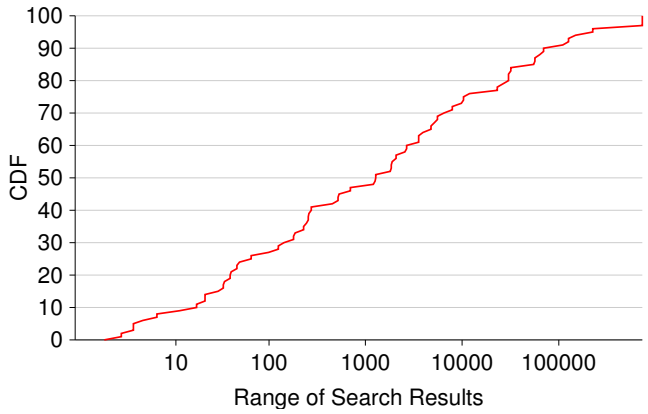


Figure 6.2: CDF of the range of search results returned for different pieces of information on a user profile.

representation format. Finally, we want to conduct a study to calculate the error threshold for the image comparison.

Chapter 7

Social Forensics

As the popularity and use of online social networks has increased, these services have become platforms for conducting nefarious activities and exhibiting offensive behavior (e.g., cyber-bullying). Furthermore, even malicious individuals (not only cyber-criminals, but perpetrators of physical-world crimes) have also adopted these technologies. While the amount of personal information disclosed by users [147] or leaked by services [181, 182] is troubling and has raised the concern of the research community, in certain cases it can have a positive "side-effect". Law enforcement agencies have been able to solve criminal cases after extracting the digital footprints of users, as they contained clues that ultimately led to the discovery of the perpetrators. Ideally, users will learn to be more privacy-aware, and limit the visibility scope of their personal information to a well-defined set of friends [86]. In such a scenario, when agencies *lawfully* acquire a suspect's device they will still be able to extract useful data from the accounts.

Social forensics tools aim to facilitate the discovery of this digital "trail of breadcrumbs", and extract data that can guide criminal investigations towards uncovering crucial information. Even though a multitude of digital forensics tools exist, they mostly focus on recovering deleted files or information from the device's volatile memory. The very few existing tools that target social networks tend to be proprietary commercial solutions which not all law enforcement agencies around the world can get access to. Our goal is to provide an extensive open source framework that will assist forensics analysts in this daunting task.

We have designed and implemented our toolset with the following usage model in mind: the authorities seize the digital devices (be it desktop, laptop or just hard disk drives) of someone suspected for a crime[1] and wish to acquire all the information regarding online activities. Social forensics analysis presents three major challenges: (i) acquiring as much data as possible from the suspect's online accounts and relevant local artifacts, (ii) correlating contacts across services, and (iii) visualizing this extensive collection of data. Our modular framework contains components for handling all three tasks.

The core functionality of any forensics analysis tool is the extraction of user data. We create a series of modules, each designed for extracting data from a specific service. When available, we take advantage of public APIs. In the remaining cases, we build custom crawlers for acquiring the data.

The correlation of users across services is a very crucial, yet challenging, aspect of our framework. Our correlation component follows a series of techniques for mapping user accounts

[1] For the remainder of this chapter, we will refer to this person as the *suspect* for reasons of simplicity.

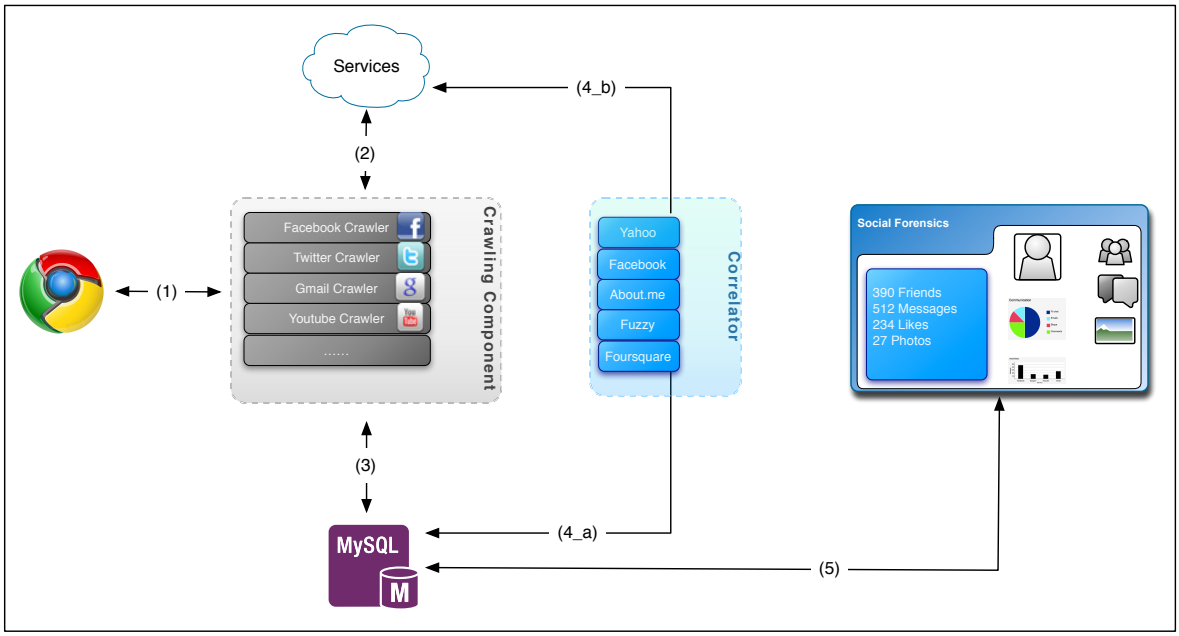


Figure 7.1: The architecture of our framework which is comprised of three major components.

from different services. Using the method we presented in Chapter 5.3, we map email addresses extracted from the suspect's accounts to Facebook profiles, which are the core sources of information. We conduct a similar process in Foursquare utilizing the search functionality of the official API. Furthermore, we employ data from `about.me`, a social directory site where users create a profile page with links to their social accounts, to further improve our correlation results. Finally, we also use fuzzy matching techniques for matching user names and email handles collected from different services.

The datasets collected during the extraction process contain a range of different types of information regarding online activities. Existing forensics-related visualization tools usually focus on the depiction of graph-related data. However, various visualization libraries exist, and can handle multiple types of data. As such, we build upon existing libraries and create a visualization framework, designed specifically for visualizing data representing user activities in online social networks and communication services. Furthermore, the massive amount of data necessitates the creation of dynamic viewpoints of varying granularity, that will assist analysts in surveying aggregated statistics, as well as focusing on specific users or interactions.

7.1 Impact of Digital Forensics

Digital forensics analysis has been a valuable asset in solving crimes in spite of its relatively young "age" compared to traditional forensics. Initially, the focus was on analyzing data stored on a computer, and recovering files that suspects had erased. However, as a result of the explosive advances of technology and its use creeping into all aspects of life, nowadays digital devices (e.g., laptops, smartphones) contain a significant amount of data that can assist the authorities in solving crimes.

A large amount of interaction takes place in online social networkings services and over digital communication media such as emails, instant messaging and VoIP networks. Users access information through these devices and save entries about their appointments in digital calendars. Furthermore, a large amount of data is saved online and not on a specific device. Thus, it is mandatory for forensics tools to extract data saved online, and not only extract data stored locally on a device. The goal of social forensics is to leverage social networking and communication services for extracting as much information as possible, regarding the online activities and communications of a suspect.

Multiple reports describe cases where the authorities have resorted to social networks for acquiring information, which has ultimately led to cases being solved (e.g., [190]). Even murder cases have been solved with the use of clues extracted from the suspect's digital communication and online activities [14, 136]. A survey held in 2012, among 600 law enforcement agencies from 48 states in the USA, reported that 92.4% of the agencies surveyed online social services [47]. For 77.1% this was done as part of criminal investigations. This survey reflects the significance of the data available in online services for assisting authorities in solving crimes. The importance of this information was also made evident by the case of PRISM[2], the electronic surveillance program operated by the United States National Security Agency (NSA), which was just recently made known publicly [44].

It is evident that the evolution of technology, and its widespread adoption in everyday life, mandates the evolution of investigating techniques as well. Thus, there is need for an extensive toolset that can extract data from all these services, correlate contacts and information across services, and provide visualization of the data in a dynamic and intuitive way.

7.2 System Implementation

The core of our framework has been implemented in `Python` as a collection of components. We have designed it in a modular way so it can easily be extended by adding new modules for other social networks and services. In this section, we provide a high-level overview of our system, describe the role of each component, and present technical details regarding the implementation of some of the components we have created. Figure 7.1 presents the architecture of our framework and the steps that comprise the whole procedure:

(1) The data collection component uses stored session cookies and user credentials to log into the online services as the suspect.

(2) Each crawling component extracts as much data possible from each service that the user has an account for.

(3) All extracted data is saved into a MySQL database.

(4) The account correlator component:

   (a) Pulls the account information of the suspect's contacts from the database.

   (b) Uses several techniques for correlating the accounts, some of which leverage online services.

(5) The data visualization component fetches data from the database asynchronously and dynamically presents the viewpoints requested.

[2]Discussing the ethics of such a program is outside the scope of this work.

7.2.1 Usage Scenario

We implemented our framework with the following usage scenario in minds. The *forensics analysis investigator* has acquired the suspect's digital device (or hard disk and connected it to a computer) and connected it to the Internet, since the data extraction and correlation components must connect to online services. Nonetheless, even though we have developed our system as a social forensics framework, it can also be useful in other situations. One of the most important subjects in news headlines recently, has been the revelation that the NSA (and, potentially, other government agencies as well) monitors popular social networking sites and other digital communication services. As such, *users* can utilize our system to create a complete and unified view of their online activity history, with statistics regarding specific activities and per-user interactions. This can raise user awareness regarding how correlating disjoint online accounts can result in privacy leakage. The Immersion project [48] similarly explored privacy issues by visualizing email data shared by users. Finally, components of our framework can be leveraged by *researchers* for collecting, correlating, or inspecting data from social networking experiments collected from social networking experiments..

An important design aspect of our system, was to make its execution as simple as possible. Ideally, the analyst would need only execute a program and everything else would be done automatically. However, due to the requirement of authenticating the crawling modules that use public APIs with the social networks through `OAuth`, a small amount of manual intervention is needed. Specifically, after the system logs into a service, the investigator is prompted to authorize the crawling component for the suspect's profile.

After the authorization phase, everything else is completed automatically. The framework installs a MYSQL database and creates a series of tables for storing all the information from the suspect's accounts. The libraries required by the crawling component, for example fbconsole [130] and Tweepy [80], are downloaded and installed automatically. The libraries for the visualization component are included within the web application.

7.2.2 Data collection components

Depending on the targeted service, the corresponding crawling component attempts to extract as much information as possible. In the case of online social services we leverage existing public APIs, if available. Otherwise we create custom crawlers for extracting the data. Here we provide technical details for certain modules.

Log-in process. Our tool uses the credentials saved in the browser's password manager or existing session cookies, to log into the targeted services as the suspect. Alternatively, the analyst can manually add the suspect's credentials in a configuration file, when no other method of logging in is available for a service.

The password managers of Chrome and Firefox utilize a `SQLite` database as their password manager back-end. Some browsers, like Firefox, retain this database encrypted using a "master password". On the contrary, the Chrome browser does not employ any encryption mechanisms, thus, storing the credentials in plaintext. We implemented a custom password extractor that locates Chrome's `SQLite` password file in the filesystem, and extracts credentials belonging to social networks and relevant services.

Browser session cookies are also stored in `SQLite` databases found locally in the filesystem. The same process is followed to extract session cookies.

Facebook. Once logged in, a custom application is installed in the suspect's profile, so the

data can be retrieved through Facebook's Graph API [131]. This application has access to all resources available in the profile. After installation, our system leverages the Facebook Query Language (`FQL`) to extract the data from the user profiles [129]. `FQL` provides an `SQL`-like interface for querying user data, and can evaluate multiple queries in a single API call through `FQL` multi-query requests. Queries are packed as a `JSON`-encoded dictionary and sent as a single request. The response includes a similar dictionary with the respective results.

Twitter. In order for us to use the Twitter API, similar steps are followed. An application that has full access to the profile data has to be installed in the suspect's profile. Twitter poses an extra overhead during the crawling phase, due to its rate-limiting policy. Requests are performed with 10-second intervals, for avoiding potential rate-limiting issues. Protected accounts (whose information is only available to followers) are collected with the highest priority. Next, we focus on accounts with small volumes of data, i.e., those with the smallest volume of tweets.

Google+. We utilize the official Google Plus API [38] for extracting the data. Through this API and the `OAuth` authentication method, we extract all the public information from the users' profiles, and the contacts from public circles. The information also includes the name of the city where the user resides. We rely on the Google Geocoding API [41] for converting the city to a pair of geographical coordinates.

Foursquare. Our crawling component is built upon a `Python` wrapper [34] for the official Foursquare API [31]. After the `OAuth` authentication is completed and an authorization token is acquired, the crawler extracts the data through API calls that return the data formated as `JSON` objects.

7.2.3 Account correlation component

This component has a very important role. As our goal is to collect data from a multitude of online services, we require a method for correlating the suspect's contacts across services. Several separate modules comprise the component, each leveraging a different service or technique. We can see an overview of the correlation process in Figure 7.2. The Yahoo module is executed once as its output is not affected by the outcome of the other modules. The remaining modules are executed in a loop, as each module might result in information that can be processed by other modules as well. Thus, the correlation process executes these modules in a round-robin fashion until none of the modules produce new information within an iteration.

Yahoo. The goal of the first module is to extract the email addresses of the suspect's Facebook contacts. Even though Facebook apps are not allowed to obtain the email addresses of the user's contacts [19], there is a method to bypass that restriction. Specifically, Yahoo mail allows one to export their Facebook contacts and add them to their email contact book. The output of this process is the Facebook profile names and the email addresses used to setup the profiles. While not all email addresses are returned, as users can change the visibility of their email address, by default the email is returned. In a small study we conducted among colleagues, we found that approximately 70% of their friends tend to keep the default setting and their email addresses were imported.

Facebook. In Chapter 5.3 we demonstrated how Facebook can be leveraged as an oracle for mapping a user's email address to his profile. We use this technique for mapping email addresses to the Facebook accounts that did not yield results in the Yahoo module. Again, while a user can change the privacy settings to be removed from such searches, it is enabled by default and not disabled by average users. This module also creates synthetic email addresses

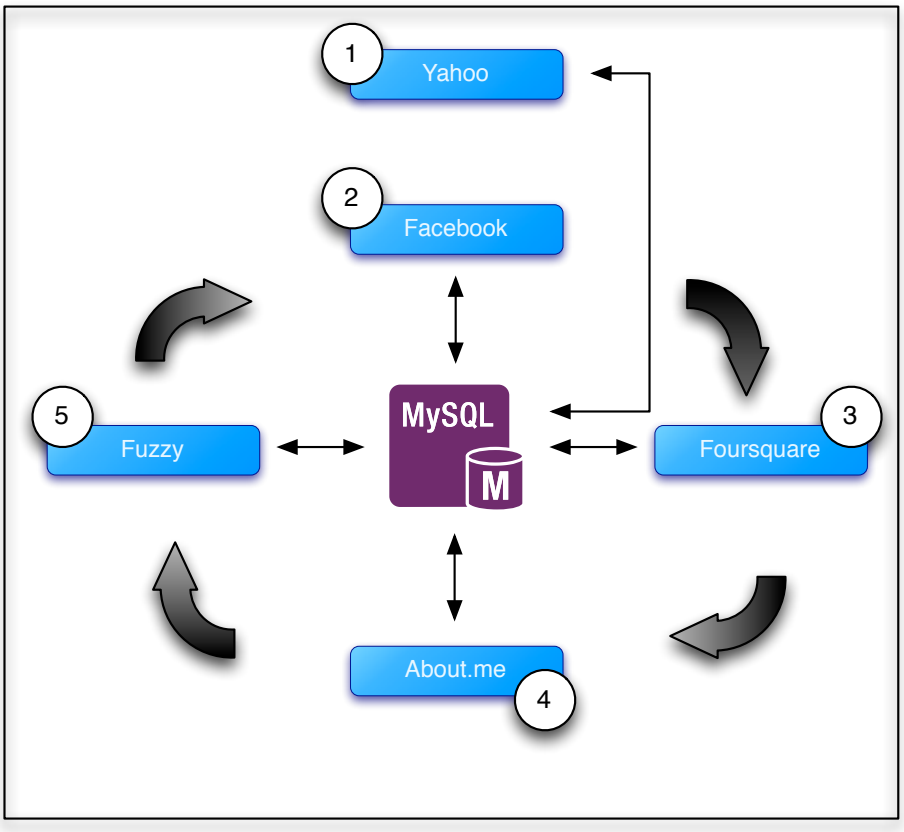


Figure 7.2: The account correlation process.

using certain variations of the given user name (e.g., "`john_doe`", "`doe_john`") along with the most common email providers (namely "`windowslive.com`", "`hotmail.com`", "`gmail.com`", "`yahoo.com`","`msn.com`") in an attempt to guess the user's email address.

Foursquare. The official API contains a call that searches for Foursquare accounts based on different types of information and can, thus, also be used as an oracle for correlating user accounts. Specifically, the API call takes as a parameter any one of the following pieces of information and returns the relevant Foursquare account (if it exists): Facebook ID, Twitter handle, email address, name, phone number. Thus, apart from locating a user's Foursquare account, we can also associate disjoint pieces of information we have collected from other services.

About.me. This site offers a platform for users, where they can create a personal page that contains links to their accounts on popular social networking services. Using the names extracted in previous steps, we search for `about.me` profiles with the same name and extract the links to their profiles on social services. We then attempt to verify that the account belongs to the same user by comparing the account IDs to any we have correlated previously.

First we leverage the website's search functionality for locating the suspect's contacts that have an `about.me` profile. As the search query results are dynamically rendered through `Ajax` requests, we scrape the results through `PhantomJS` [67], a headless webkit that also offers

a `Javascript` API. After obtaining the user profiles, we extract the available links towards social network profiles. Each link to a specific `<network>` is accessible through a unique URL[3]. Similarly, other social directory sites could be used.

Fuzzy matching. Some of the services we extract data from don't provide the email addresses of the account's contacts, which would allow us to deterministically correlate user accounts across services. Furthermore, different email addresses may have been used for different services. To overcome this, we compare information collected from different services and match them based on similarity. While this method follows a "fuzzy" approach, we are able to obtain results, as users tend to reuse user names across services, or simple variations of them. For example, a user with a Facebook profile under the name "John Doe" might have an email address handle "`john_doe`", "`johndoe80`" etc.

User input. While the above methods yield results, nonetheless, certain accounts may not be correlated with others belonging to the same user. This could be due to users creating multiple accounts under completely different user names. As this correlation can provide invaluable information during the visual inspection of the data by analysts, our visualization component enables the manual correlation of accounts. Specifically, the user can correlate an account from one service with accounts from other services. That information is saved, and the dynamic perspectives will reflect the new associations. Similarly, the user can remove any erroneous correlations made during the automatic correlation procedure by the fuzzy matching module.

7.2.4 Visualization components

Our goal is to develop a modern visualization platform that will offer a wide variety of graphic data representations, while remaining portable. This led us to create it as a web application. The front-end is designed to run on the same machine where the data is kept.

The vast amount of data mandated the use of an asynchronous, event-driven model for the front-end, where data is fetched upon request. The front-end is built upon `AJAX` requests using the `jQuery` framework [52] for data retrieval and manipulation.

The ever-growing need for complex data visualization has lead to the release of powerful frameworks. `D3.js` [13] is a JavaScript visualization library capable of rendering a variety of schematics such as Graph layouts and Calendar Views among others. This framework is used for the majority of visualizations incorporated in the front-end. Moreover, we leverage the Google Maps JavaScript API [40] to render location-based information on a map.

7.3 Data Collection

Here we present a list of the services from which we collect user data, as well as a description of the types of information acquired. For every online social network, we also collect any information that is reachable for every one of the suspect's contacts.

Facebook. This is the main source of information, as it is the most popular online social network, and users tend to reveal a large amount of personal information on it. Our crawling component extracts any of the following information that exists:

- *Personal information*: this may include current location, hometown, education and work information.

[3] `http://about.me/content/<username>/<network>`

- *Contact list*: apart from the list of the suspect's contacts, we also collect any custom lists and the contacts contained in each list.
- *Status updates* and any links contained.
- *Chat logs* along with timestamps for each message.
- *Photos:* links to the photos and information regarding photo albums, photo timestamps and tagged users.
- *Videos* uploaded by the suspect, and videos he has been tagged in.
- *Check-ins:* the places the suspect has checked into, the timestamp and the data and coordinates of the place, along with tagged users.
- *Likes:* activities and articles the suspect has liked.
- *Shares:* pages the suspect has shared.
- *Fan pages* (also checks if the user is an administrator of the page).
- *Events* and the information of the users that participated.
- *Groups* the suspect is a member of, and the information of the other members.
- *Notifications* the suspect has received.
- *User notes*.
- *Contact information:* we also collect all of the aforementioned data from the contacts that is viewable through the suspect's account (e.g., a contact's chat messages are not viewable.)

Twitter. We first collect the account's information and contact list. That includes the accounts the suspect follows as well as those following the suspect. We also collect the suspect's tweets as well as any tweets re-tweeted, and all available metadata (e.g. timestamps, location).

Foursquare. We collect the suspect's check-ins along with the corresponding metadata. Specifically, we collect the timestamp, the venue's name, VenueID, and location coordinates. We also collect the list of friends, and any links to their profiles on other networks. Unfortunately, due to limits set by the API and website, we can only retrieve the last 100 check-ins of the suspect's friends.

Skype. We first collect the list of contacts and their disclosed information (which may include location, gender, date of birth). Then we extract the history of chat logs and relevant metadata, as well as call history (and duration) and file exchanges. We also attempt to retrieve any exchanged files that are still located on the hard drive.

Gmail. We collect all emails exchanged with the suspect, and extract the email addresses and any names associated with those addresses. For each email we also collect the relevant metadata.

Google. We access the suspect's account in Google and extract the relevant information from Google calendar and Google Docs. Specifically, we collect all calendar entries (which may contain a location, a description, and other users attending), and download documents accessible (we also retrieve information about which other contacts have access to the documents).

Google+. We first collect the suspect's contacts contained in the various "circles" (i.e. contact groups), and the suspect's activities; posts, comments, shares, and "+1"s (similar to likes in Facebook). We extract publicly available data from the accounts of the contacts, as

(a) User granularity

(b) Service granularity

Figure 7.3: Two elements from the aggregated statistics perspective. We provide details regarding the most interesting activities, at a user-granularity and service-level granularity.

well as any accounts that have commented on the suspect's profile (even if they are not part of one of his circles).

Youtube. We first collect the suspect's information. Then we extract the history of watched videos, and channel subscriptions, playlists, uploaded videos and their comments and favorited videos.

Dropbox. We first locate the Dropbox folder, depending on the suspect's operating system, by retrieving the information from the application data. Then, by traversing the Dropbox directory tree, we extract all the files with their corresponding metadata. We also keep the application data that can be used for other aspects of forensic analysis [28].

7.4 Activity Visualization

In this section we describe the various methods for visualizing our collected data. The plethora of services that can be used by suspects require a grouping of this disjoint information into a unified set, where actions across services are correlated (e.g. what type of communication does the suspect have with user X from all services). Furthermore, the abundance of available information necessitates the ability to shift focus to specific activities (e.g., status updates on Facebook), and interactions (e.g., users with the largest amount of shared activities with the suspect). Thus, we provide the analyst with dynamic "perspectives" of varying granularity, with aggregated correlations as well as fine-grained views of the collected data. We have several viewpoints for creating the different perspectives.

Aggregated. Here we present aggregated statistics regarding the most interesting activ-

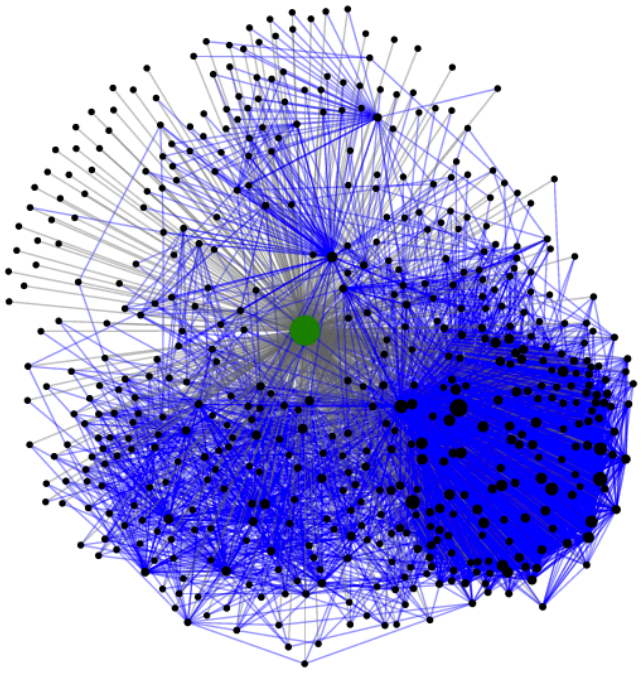


Figure 7.4: An example plot of the suspect's social graph. The suspect is depicted with the green node. The size of a node is defined by its degree of connectivity. Edges toward the suspect's node are grey, while edges between contacts are blue.

ities from all the services. With one glance, the analyst can see which services the suspect mainly uses, and what data is available. In Figures 7.3(a) and 7.3(b) we see an example screenshot regarding some of the aggregated statistics presented in this viewpoint. Specifically, we can see the most important types of data across services and a more detailed description of activities per service, respectively.

Service. Here we focus on a specific service, and present aggregate statistics regarding the users activities. A list presents the contacts that have had the most communication with the suspect. Next, as shown in Figure 7.4, we depict the structure of the social graph and the interconnections between all contacts. The node's size is based on the number of connections the contact has. Thus, the analyst can immediately recognize heavily connected users or outliers. The graph can plot contacts of a specific service as shown here, or a combined view of all services where the contact's of each service have a common color. Each graph node represents a user, and when clicked presents the contact's name and photo. Furthermore, a contact search function dynamically detects and highlights nodes in the graph, allowing investigators to quickly identify contacts of interest in the graph. In Figure 7.5 we present a screenshot of a graph that visualizes the total communication between suspect and online contacts. The amount of shared activity defines the width of the connector. This enables the users with the most communication to be easily identified and scrutinized. When the connector is clicked, a window presents all the shared activities.

User. A very important viewpoint is that which focuses on a specific user. Once the analyst has identified online contacts that might be of interest, he can use one of two perspectives. First, one can select the contact and be redirected to an aggregated statistics viewpoint,

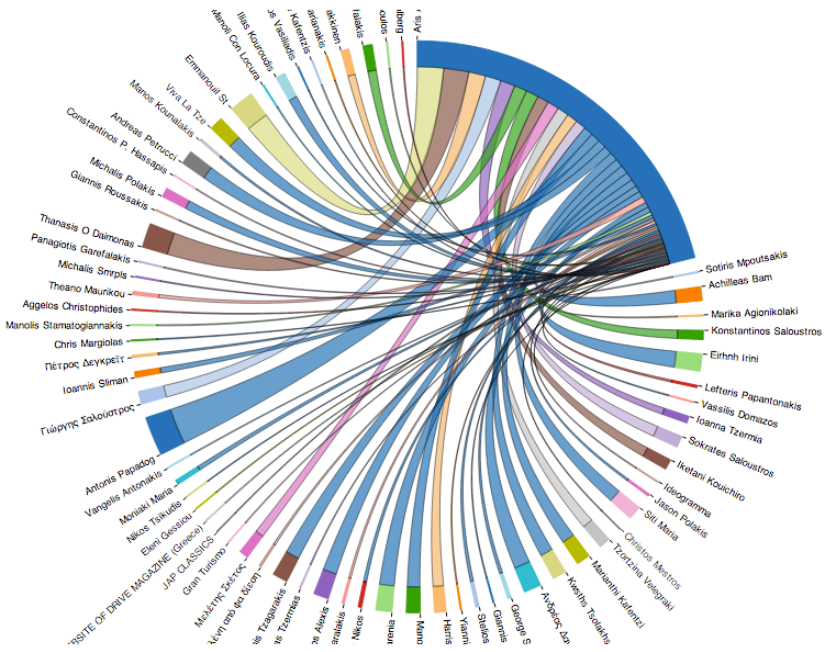


Figure 7.5: This graph plots the overall communication between the suspect and his contacts. This visualization facilitates the recognition of important contacts, as the volume of communication determines the width of the connection between suspect and contact.

containing all information available regarding the actions of that contact across all services (based on the number of accounts that have been associated during the correlation phase). Second, the analyst can choose to focus only on the shared activities the contact has with the suspect across all services. That includes, chat messages, emails, wall posts, shared photos, etc. The coarse-grained perspective presents aggregated statistics, while the fine-grained perspective allows to focus on a specific type of activity. In both viewpoints, the investigator can ultimately view all individual activity and communication resources, e.g., exchanged messages, pages "liked", or Skype calls. Furthermore, the viewpoints can be dynamically configured to visualize data from one or all services.

Timestamp. Time is an important factor for visualizing relevant data. Every perspective contains a color-coded calendar depicting the amount of activity a user has conducted on a specific date, which allows a fine grained overview of a specific period. We show a segment of an example calendar in Figure 7.7, where blank squares depict days where the user did not send or receive any emails, while the red square depicts a day with a large volume of exchanged emails. Furthermore, a histogram, as seen in Figure 7.6, presents an overview of the suspects activities. The analyst might wish to focus on the activities of the suspect during a specific time period which is of interest, or a specific service. As such, certain viewpoints can dynamically change and focus on a specific time window.

Content. A word cloud provides the analyst with a quick view of the most common words contained in the suspect's communication, which can be across services or focused on a specific service or user. Thus, recurring motives and topics can easily be spotted. In the case of Twitter, we also create a word cloud with the hashtags (i.e., topics) of the suspect's tweets

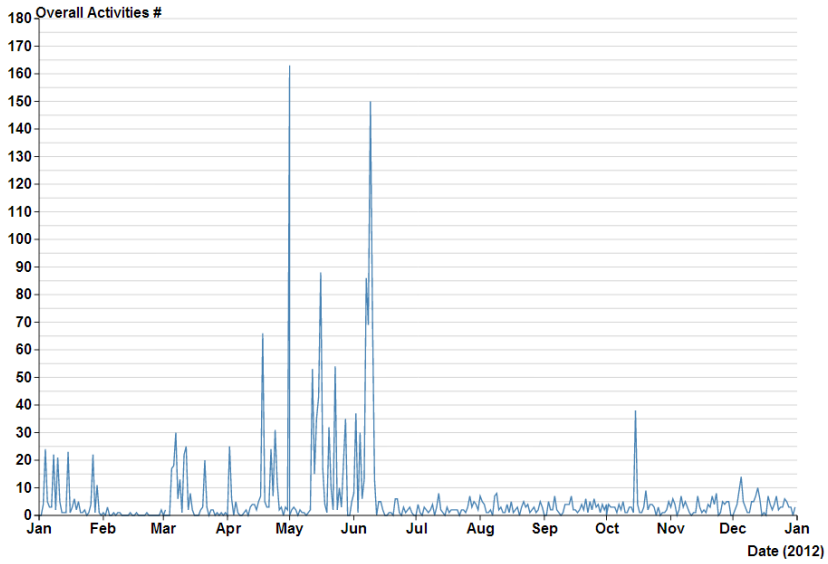


Figure 7.6: Histogram presenting the suspect's activities for the entirety of the Facebook accounts' lifespan.

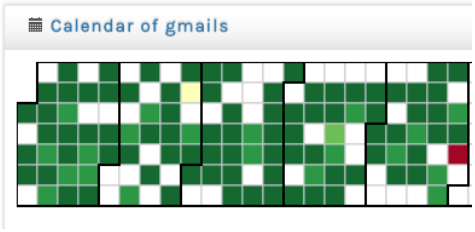
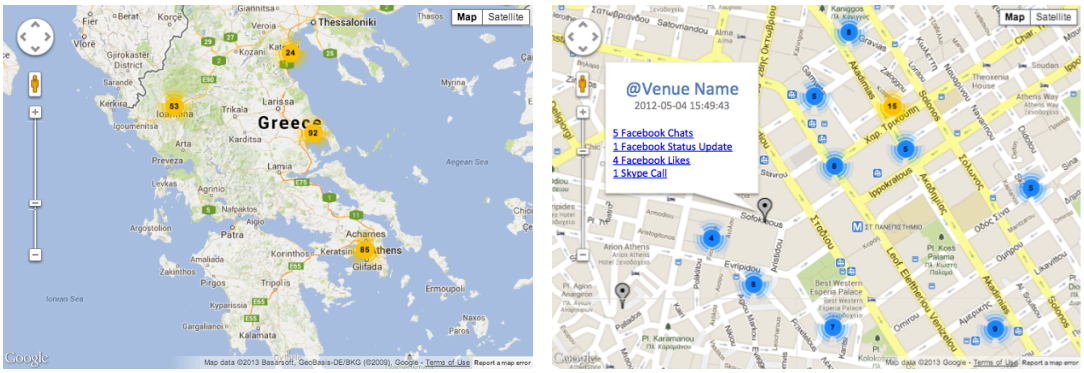


Figure 7.7: Extract of the calendar element, depicting the email exchange activity of the suspect over a period of five months.

as we show in Figure 7.8. This can also reveal subjects that the suspect tends to follow or comment on (e.g. politics, religion) and can be relevant to the analyst's investigation. Clicking on one of the terms will fetch all the messages, emails, tweets or posts containing the term. Furthermore, we also follow a more targeted approach, by employing the list of keywords that the US Department of Homeland Security searches for in social networks [56]. Specifically, we search for 377 keywords that belong to 9 categories, ranging from terrorism to drug-related incidents. The number of occurrences are broken down per-category and per-service. By clicking on the respectful information, the analyst is presented with the resources that contain the keywords.

Location. A very important piece of information is the suspect's location. Using information from the suspect's check-ins and residence we plot a map with the locations he has visited, and also visually annotate the amount of times each location has been visited. Furthermore, the analyst can also define a time window, within which all of the suspect's activities are correlated with that location. For example, with a time window of one hour, by clicking on the location marker, a window will inform of all the activities (e.g. chat, Skype

Figure 7.8: The word cloud shows the words most frequently contained in the suspect's communications. Here we see an example created from a user's Twitter hashtags (topics).



(a) Aggregated view (b) Close-up view

Figure 7.9: Two views of the map plotting the suspect's check-ins. (a) An aggregated city-level view. (b) The details of a specific check-in and the associated activities.

calls) the suspect conducted up to one hour after the check-in. Thus, the analyst can associate important activities to specific locations or even search for patterns of activities at certain locations. In Figure 7.9(a) we can see an example screenshot showing the aggregated check-ins at a city-level granularity. Figure 7.9(b) depicts a closer view of a specific region, with the information window for a specific check-in. The window presents the name of the venue, the check-in timestamp and a series of activities that have been completed within a one-hour time window. All elements are click-able for presenting the resources of interest. Furthermore, as a specific period might be of interest, we can plot the check-ins conducted during a specific time-period. Also, the investigator can select a contact, a distance X and a time duration T, and the map presents check-ins that the suspect and the contact conducted with a time difference up to T at venues that have a max distance of X.

Photographs. A valuable resource of information in criminal investigations are photos found in social networks, as demonstrated in the case of the Vancouver riots [190], where

Network	Connect	Profiles	**FB_Corr.**	**Correl.(total)**	**Complete(%)**
Facebook	517	517	-	390 (415)	75.4
Twitter	77	114	67	98 (111)	85.6
Google+	24	121	16	24 (26)	19.8
Foursquare	1	115	70	70 (108)	60.9
Skype	64	113	46	55 (55)	48.7
Module	FB_results	Results	Duplicates	FP	
Facebook	51	51	0	48	
Yahoo	-	352	0	-	
Foursquare	63	98	4	-	
Fuzzy	78	121	5	3	
About.me	1	9	1	-	

Table 7.1: Summary of the number of contacts extracted from each social network. The correlation contacts refer to the number of users that were mapped to a profile through each of the correlation techniques.

vandals were identified through photos posted in social networks. The investigator can select to view all the photos collected from the suspect's profiles. Any available user tag information is also presented, and statistics show the contacts with the most common photos with the suspect (based on tag information).

7.5 Data Correlation - Case study

We conducted a small case study to evaluate the effectiveness of our correlation process. In our case study, one of the authors assumed the role of the "suspect" and we ran our toolset on his computer and extracted the data from his social services.

Table 7.1 presents the results from this short case study. We consider the Facebook account as the core dataset of the experiment, with the suspect having 517 friends. Subsequently, we manually searched the other online services of interest and calculated how many of those Facebook users also have a profile in each one of the other social networks (*Profiles*) and how many of those are actually connected with the suspect's corresponding profile (*Connections*). While some accounts may have been overlooked due to nicknames and alternate email addresses being used for setting up the accounts, we consider this dataset as the ground truth for evaluating the effectiveness of the correlation modules. The *FB_Corr.* column refers to the number of accounts from each service that have been correlated to a Facebook account. The *Correl.* column refers to the overall number of accounts that have been correlated to accounts of any service. In certain cases, the modules discover accounts that belong to the suspect's contacts but are not connected to the suspect's Facebook which we consider the core source of information (the overall number of correlations that also contains these is presented in the *(total)* column); for example, the suspect and a user are connected in Twitter but are not friends in Facebook. *Complete* evaluates the completeness of the correlation process and denotes the percentage of accounts of a service that has been correlated to an account of a different service, for users that are Facebook friends with the suspect. We plot a view of all the correlations created by our modules in Figure 7.10. For each user, the points depict the

accounts from various services that have been associated through the correlation procedure.

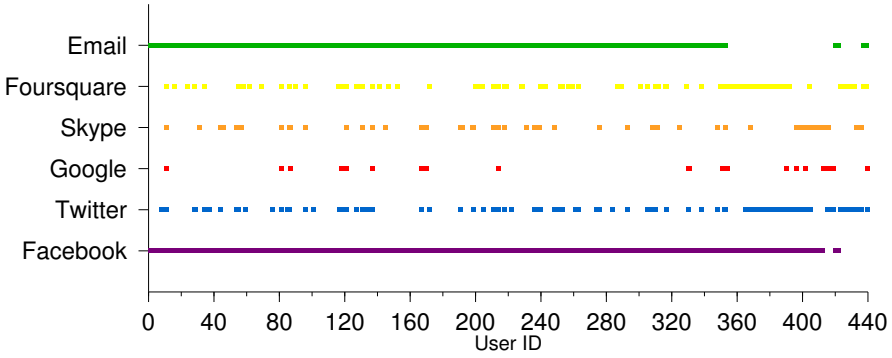


Figure 7.10: The associated accounts for the suspect's contacts. A user's account is plotted only if it has been correlated with at least one account from another service.

The lower part of the table presents in detail the results of each correlation module. The process of each module has been described in Section 7.2.3. The *FB_Results* column contains the Facebook accounts returned by the specific module, while the *Results* column contains the overall number of results returned by each correlation component regardless of service. The *Duplicates* column refers to overlapping results, i.e., the number of discovered correlations between two accounts that were also discovered by other modules.

FP refers to the number of false positives, i.e., accounts returned by a module that do not correspond to an actual friend of the suspect. These were verified manually and were due to users having similar names. In the case of the Facebook module, where potential crafted email addresses are used to discover Facebook profiles, there was a large number of false positives, as the emails we crafted belonged to unknown users that had similar names to the suspect's friends. As such, while this technique is very effective for targeted phishing attacks, it is not very accurate for automatically correlating accounts in social forensics due to the false positives. Nonetheless, it was able to discover three profiles that had not yielded results in the Yahoo module. Thus, the results of this module should be manually verified by the forensics analyst.

Even though the case study was performed on a single user, it demonstrates the effectiveness of the correlation techniques, and should work as a warning to users regarding their privacy awareness, as seemingly disjoint sets of information can be correlated.

Chapter 8

User Actions: Exploiting Location-based Services

Several location-based services that are also social networks have emerged during the last couple of years, Foursquare and Facebook Places being the most famous examples. The core operation of these social utilities is based on a large number of users that are willing to share their true geographic location. Users of these systems announce their location to the rest of the community or their on-line contacts, and can win awards depending on how often they share their location. For example, American Express offers discounts as an incentive for their customers to connect their account with the Foursquare application [4].

Foursquare is currently the most successful LBS. A very important aspect of its business model is the rewarding system for users that frequently check into specific venues. The user with the most check-ins for a venue in the last sixty days is crowned the venue's mayor. Venues attract customers by providing special offers for their mayors. This entails an incentive for users and, therefore, it is crucial to prevent fake check-ins that will have a negative impact on the system and deter honest users from participating [186]. As articles describing simple methods to post fake check-ins were published (e.g., [60]), Foursquare implemented a cheating-detection mechanism for prohibiting cheating users from becoming mayors. The deployment of such a feature [32, 66] was mandatory for reassuring users that cheating was deterred, and preventing a major decrease of the user base. Another important aspect of Foursquare is its recommender system [200], built upon the suggestions and tips left by users after checking into a venue. According to their CEO [33], "*check-ins drive the data, which drive the recommendation engine*". Since these services base their operation on the honest disclosure of location, it is vital for clients to transmit their position accurately so as to prevent the loss of the user base and the degradation of the recommender system.

Various methods have been used to post fake locations to mobile social networks. The most trivial is to hijack the GPS driver and provide applications with arbitrary coordinates. There are research efforts for the development of Trusted Sensors [211], including geolocation sensors. If smartphones are equipped with a trusted computing base, tampering with the data returned from the GPS antenna, or modifying the system to receive the coordinates from an application, can be harder although, arguably, still possible [229]. We do not tamper with GPS readings, but conduct a systematic study of how the *application layer* of LBS can be leveraged to transmit fake information. Thus, our methodology is not affected by the presence of trusted sensors.

We create a testing platform that leverages public APIs available to application developers, and follow a black-box approach where we perform arbitrary check-ins in various places of the world, without changing our actual physical location. We then systematically analyze all server-side heuristics that aim to detect misbehaving clients. To ensure the completeness of our study, we also conduct experiments where we masquerade our actions to appear as if originating from the official applications, to reveal potential differences of the detection heuristics for public API calls. This is achieved using authentication tokens extracted from the official mobile applications through low-level reverse engineering.

We reveal a series of thresholds, which, if taken into consideration, allow a user to check into Foursquare, while traveling around the globe with a speed of over 900 mph. We also discover that users can check into a venue from as far as 200 meters away, and the maximum number of check-ins a user is allowed to commit is enforced using a 24 hour sliding window. Our technique revealed a bug in Facebook Places which allows anyone to perform check-ins all over the world with *unlimited* speed. No fix has yet been released.

Based on our findings we create an attack algorithm that takes into account heuristic thresholds and can maintain continuous mayorship in a set of venues across the globe. By employing 10,000 accounts, sold in the underground market for $150-$450, our adaptive algorithm can acquire the mayorship of all venues, and severely impact Foursquare's business model. Our experiments demonstrate that anomaly detection heuristics cannot secure a LBS against fake-location attacks. Detecting malicious clients and distinguishing fake check-ins from legitimate ones is not trivial. Even if heuristics become stricter, the attacker can simply follow a stealthier approach, as multiple accounts are used to carry out the attack. Stricter heuristics will also result in the system becoming too restrictive for legitimate users as well, which can have a negative impact on user participation. We argue that new directions need to be followed for securing LBS.

Subsequently, we present our proof-of-concept implementation of *Verified Check-in*, an NFC server solution, along with a security analysis of how it holds up against a series of attacks, as well as an evaluation of its performance. With a total cost of about $75 at retail price, we consider our system to be ideal for deployment by venues that offer awards to LBS customers.

8.1 Location-based Services

With smartphones, users can use networking services on the go. This has introduced the aspect of user location, which has radically shifted user behavior and led to the blooming of location-based social services, that allow users to inform their contacts of their current location.

Foursquare has over 30 million users and 1 million registered businesses, with users conducting millions of check-ins per day (January 2013). A very important aspect, which has ultimately led to its success, is the concept of achievements for users based on their check-in behavior was integral to its success. Achievements belong to three different categories: points, badges, and mayorships. Users earn points for every activity such as adding a new venue, or a check-in into a venue, while badges require combinations of activities. The mayorship is awarded to the user with the most check-ins for that venue in the last 60 days, and only one check-in per venue is allowed each day. As a result Foursquare is perceived as a game and, thus, cheating users discourage honest users from further participation.

Facebook Places follows a similar approach, where users can check into places and share that information with their friends. A major goal of the service is the integration of users' location with all the other types of information they post in Facebook profiles, such as photographs and status messages. It does not present an award system to create a "gaming" experience. Nonetheless, it also provides venue owners with the ability to create offers for users that check into their place. Recently, Facebook merged this service into its system and discontinued it as a separate service. For the remainder of the chapter, we will refer to this component as Places. Furthermore, venues are referred to as pages, however we will retain Foursquare's naming convention.

Check-in economy. The opportunity for venues to use LBS for advertising and attracting customers has led to the creation of a new business model that relies on users' activities combined with their geographical location. When users check into venues and post that information on their profiles, they are actually advertising the venues. As a result, an increasing number of venues are attracting Foursquare users by offering awards, ranging from discount prices to free products. This is similar to the *Like economy* [110], associated with users *liking* particular resources in Facebook, evolving with check-ins stemming from Foursquare's activity. Ensuring the produced economy is stable, requires that check-ins reflect clients announcing their true geographical location. However, this stability seems very fragile. A logical consequence of venues using Foursquare's achievement system for offering awards is the appearance of users cheating the system for fun or profit. This is done through *fake-location attacks*, where users check into venues without being there.

Apart from mobile social networks, other web sites offer services that rely on user location and are susceptible to fake location attacks. For example, Gym-pact [45] offers real cash for users that successfully check-into their gym a certain number of times each week. As we show in this paper, attackers can deploy, by using commodity resources, fake-location attacks that will result in direct profit.

Fake-location attacks have a major impact on the credibility of LBS. They pose a great threat as competitive users will leave the system if fairness is not ensured and, thus, break all economics associated with these services. To make matters worse, as smartphones become widely used and the popularity of such services greatly increases, these attacks are bound to transit from sporadic incidents to organized fraud. We argue that LBS share certain properties that render them vulnerable. Thus, it is important to explore such attacks in detail and design effective countermeasures. We identify the following fundamental properties:

1. The user's location is sent from the client (user's device).
2. The LBS has no definitive way of verifying the location.
3. Heuristics are used to detect behavior that exceeds acceptable limits, using the following information: (a) venue location, (b) user location, (c) timestamp of check-in, (d) history of previous user check-ins.
4. With such limited information, heuristics can only be applied on the following: (a) user's distance from venue, (b) user's speed between successive check-ins, (c) distance traveled in certain time windows, (d) number of check-ins in certain time windows.

As long as (2) stands true, LBS will remain vulnerable to fake-location attacks. The limited nature of information available (3), dictates the types of heuristics that can be deployed (4). We have designed our system to be fully configurable in regards to such heuristics. Thus, it

can be used to identify the heuristic thresholds of any LBS that follows (1, 2) and demonstrate the extent of potential attacks. Our testing infrastructure can also assist LBS providers in detecting implementation bugs (as we demonstrate with Places).

User location. We expect user location to play a pivotal role in future services with functionality that will deviate from a simple check-in approach. Foursquare is also expanding by utilizing user data to build a reliable recommender system.By implementing an effective mechanism for validating the location reported by users, we can create a stable foundation for other novel services to be built upon.

8.2 Methodology

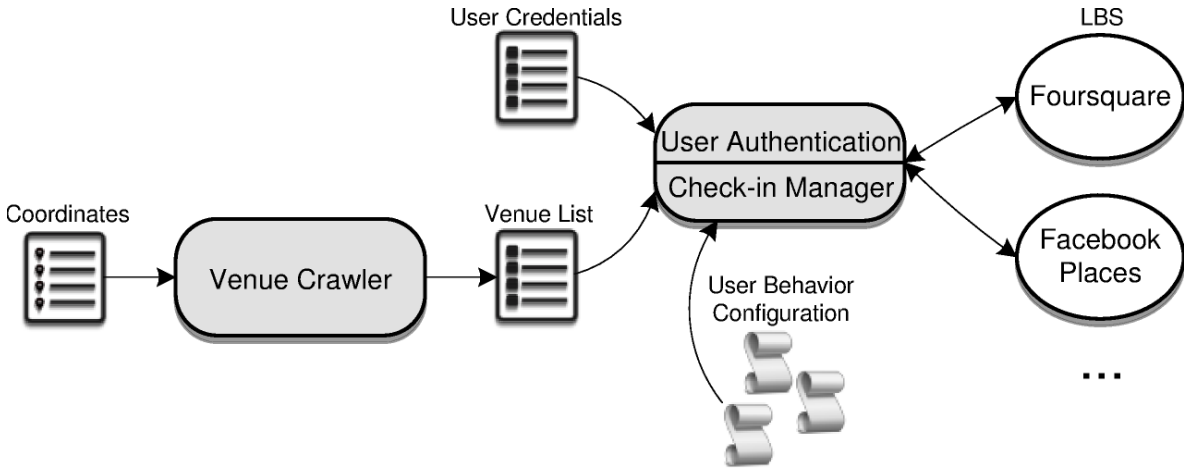


Figure 8.1: Overview of our system architecture and its various components.

Our initial goal is to create an infrastructure that is able to perform arbitrary check-ins in LBS. Figure 8.1 presents an overview of our architecture. Even though our current implementation supports Foursquare and Places, our testing approach is applicable to any LBS. User behavior in our system is configurable and, thus, we can explore the detection mechanisms deployed by any service simply by modifying the log-in and check-in functions. To uncover the heuristics used by Foursquare to detect cheating we follow a black-box testing approach. To do so, we create a series of accounts the we configure to exhibit a certain behavior depending on the heuristic we want to test. Each user is modeled by a Python Script that uses the Foursquare API calls to perform all actions. In the text we refer to it as our custom application. In certain experiments, we masquerade our actions to look as if our users are using the official Android Foursquare application.

Use of Public APIs. Both Foursquare and Places provide public APIs that allow the development of custom applications. They include a set of HTTP requests, which cover the complete functionality of the service. The Places API is provided as part of the Facebook Graph API. For developing applications one only needs to register and obtain API credentials. Application code is not reviewed and, thus, anybody can create applications that post fake check-ins. We now describe how someone can construct a custom application for interacting with Foursquare. Upon the initial registration of an application, Foursquare provides the

developer with a unique client id and client secret. One method for the user to be authorized through an application, requires the developer to first submit a fixed callback `URL` for the application to Foursquare. When a user runs the application for the first time, he is redirected to the Foursquare servers along with the application's client id and the assigned callback `URL` as parameters. Subsequently, the user is redirected back to the callback `URL`, providing a user access code. The application, then, performs a request to the Foursquare servers with the application's client id, client secret and the returned user access code. The response includes an access (or OAuth) token for that user that is tied to the specific application and remains valid for a period of time. Each `(user, application)` pair has a unique access token. Subsequently, all API calls that the application posts on behalf of the user must contain the access token. This enables the application to perform any action on behalf of the user, just by providing the access token. As an application running on arbitrary servers can post actions on behalf of Foursquare users, the user's IP address is hidden from the service.

Mimic Official Applications. For one experiment we want to masquerade all calls made by our custom application to seem as if originating from the official one. Our goal is to explore whether the official app includes further information (i.e., custom headers) that makes it receive different "treatment" from the service. To do this, we must format all API calls like the official ones, and use the corresponding authorization token for each user[1]. Revealing the original protocol is challenging, since communication is sent over an encrypted channel using HTTPS. Therefore, the only approach is to modify the application to provide us with the actual communication (described in detail in Section 8.3). By decompiling the `.dex` class files, we manage to extract the Dalvik bytecode in which we inject code for logging all HTTPS requests and responses. Thus, we are able to extract the unique application client ID and client secret. We extract the required information by modifying the application to provide us with the actual communication. If the requests are sent in cleartext, decompiling the application is not necessary. However, sending requests over SSL/TLS is considered safe practice and we expect that most LBS will do so.

Black-box testing. To reveal the detection mechanisms deployed by LBS, we follow a black-box testing approach and use test profiles that post arbitrary check-ins. We design our system to allow the configuration of several parameters of user behavior. By modifying the behavior, we are able to trigger the heuristics and identify their thresholds.

User accounts. For our experiments, we create accounts for posting the fake check-ins which enable us to determine the heuristics and thresholds that detect cheating.

Ethical considerations. To minimize the impact of our experiments, and analyze the detection mechanisms without affecting other users, we took two precautionary measures. First, when exploring the heuristic thresholds, we modified our accounts so as not to acquire mayorships in venues which already had mayors. Specifically, mayorships are not awarded to accounts without a profile photo. Second, when experimenting with our adaptive attack algorithm, we targeted small venues with no mayors and used multiple accounts to serve as other customers. We were able to explore the heuristics in depth and demonstrate our automated attack, without having a negative impact on legitimate users.

[1]Each `(user, application)` pair has a unique access token.

```
1  W/System.err(2283): HTTP connection attempt:
2  W/System.err(2283): https://api.foursquare.com/v2/checkins/add
3  W/System.err(2283):   method:    POST
4  W/System.err(2283):   user-agent:
5         com.foursquare.android:2011111801:20111118:4.0.3:google_sdk
6  W/System.err(2283):   headers:
7         User-Agent=com.foursquare.android:2011111801:20111118:4.0.3:google_sdk,
8         Accept-Language=en-US,
9  W/System.err(2283):   params:
10 W/System.err(2283):     ll -> 40.68995833333334,-74.04563499999999
11 W/System.err(2283):     venueId -> 4e2204841838712abe78bdcf
12 W/System.err(2283):     shout -> Hello !
13 W/System.err(2283):     broadcast -> public
14 W/System.err(2283):     wifis ->
15 W/System.err(2283):     oauth_token -> OHO2WFBEVPTKJFYI.........JFUCZR23ACC2VAOOAC2U4ZH
16 W/System.err(2283):     v -> 20111118
17 W/System.err(2283):   response status code: 200
```

Figure 8.2: A successful check-in attempt performed by the official Foursquare Android application as logged unencrypted.

8.3 Reverse Engineering

We assume the official application is considered trusted by the service and we aim at extracting all protocol and authentication tokens [65], as they are communicated between clients and server. In this way, we are able to construct our own custom clients that can masquerade as instances of the official application. Revealing the original protocol is challenging, since all communication takes place over an encrypted channel using HTTPS. First, we provide a short description of the internals of Android applications in general, and, then, continue with details about specific functions of the Foursquare application and modifications we performed for extracting information. Specifically, we extracted the application's bytecode and modified it to log all HTTPS communication with the service's servers, unencrypted, as well as the user's authentication token application's client ID and secret. Our technique can be in principle applied to similar Android applications that communicate over HTTPS. An in depth analysis of Android internals can be found in [128].

Android Applications. Over recent years mobile applications have evolved rapidly, incorporating functionality comparable to feature-rich applications of Desktops PCs. Factors that made this evolution possible are the improvement of mobile device hardware in terms of CPU, memory and storage, as well as the development of modern mobile OSes. Nowadays, commonly used mobile OSes include Google's Android which is based on Linux, Apple's IOS which is based on the XNU kernel (Mach/BSD), Nokia's Symbian and Microsoft's Windows Phone. For our research, we chose the Android OS as our testbed platform for several reasons. It is open source and Linux based, it supports a wide range of devices, and it provides a well documented Software Development Kit (SDK). Also, both Foursquare and Places distribute native Android applications. Applications for Android are usually developed in Java. The code is compiled into `.dex` file classes which are packaged, along with any accompanying elements, into an archive with an `.apk` extension. Such elements can be manifest files, certificates, resources, assets or native Java/C libraries. The `META-INF` directory is used for storing package

integrity data. It includes the `MANIFEST.MF` file, which is used to index package related data and the application's certificate and file signatures. Application resources, such as images, are stored in the `res` directory, or compiled into `.arsc` resource files. Android applications also support asset files which are regular files that lie in the `assets` directory and are accessible by the application at runtime. An example is HTML files that contain license information. An `XML` file, named `AndroidManifest.xml`, is used to provide information about the package name, version, permissions and others. Android applications run through the Dalvik process virtual machine [118]. During the building process, code is compiled into Java bytecode format and stored in `.class` files. After that, it is converted to an alternative instruction: Dalvik Executable files have a `.dex` extension, such as those included in Android package files.

Our next step is to understand the way the official application communicates with the Foursquare servers. By observing the application's requests and responses, we can compare the output of the official application with that of our custom application that uses the Foursquare API. This is done for identifying differences in the communication or any hidden mechanisms that make the official application deviate from the public API. As the HTTPS protocol is used to create a secure channel between the application and its servers, monitoring the plaintext of the exchanged communication from outside the application is prohibited. Therefore, the only approach is to modify the application to provide us with the actual communication. By reverse engineering the `.dex` class files, we manage to extract the Dalvik bytecode in which we inject code for logging all HTTPS requests and responses. Thus, we are able to extract the unique application client ID and client secret which are observed upon the login process.

For authentication purposes, the Foursquare application includes its native Foursquare API and the Facebook Android API. The Facebook API is used to allow users to log into Foursquare using their Facebook credentials. The application uses the `org.apache.http` core interfaces and classes to achieve HTTP connectivity with their respective servers. It also includes an `HttpImpl` class which handles the communication with its servers. The HTTP GET or POST requests are initiated by Foursquare API using `httpPost` or `httpGet` functions. By injecting code in the appropriate functions, we are able to log all HTTPS requests and responses to Android's system log file, *unencrypted*. The modified code is re-compiled into a new `.dex` class file, and we create a new application `.apk` archive with its old resources and required Manifest files. For an application to be installed into an Android image, it must be digitally signed with a certificate, contained in the package. After signing the new `.apk` file, the modified application is installed in the emulator. Those functions invoke the `executeWithRetries` function, which processes and executes the communication request by calling the `createGetOrPost` and `buildClient` functions. The Private function `createGetOrPost` builds HTTP GET or POST requests, while the `buildClient` function initiates the HTTPS connections with Foursquare servers. We modify the `buildClient`, `createGetOrPost` and `executeWithRetries` functions, and the `mDebug` boolean variable is also set to True.

We run the application for testing our modified package inside the emulator image, by fixing the GPS location and simulating user behavior. The user logs into Foursquare and checks-into different venues. This is done for revealing the whole communication flow, which can be monitored in Android's system log file[2]. We notice the GET/POST requests and their headers, as well as the responses in `JSON` format. The application appends a specific User Agent, the secret user's access token and the application version to all GET or POST

[2]Our black-box testing is conducted separately using a custom application, and not within the emulator.

requests, along with the location parameters and venue ID. The unique application client ID and client secretare also observed upon the login handshake of the application with the Foursquare servers. An example check-in as performed by the official Foursquare Android application is depicted in Figure 8.2. By monitoring the communication characteristics and variables, we are able to use the specific hidden data and emulate the application behavior through our custom code.

8.4 System Implementation

Our system has been implemented in `Python` as a collection of components, and can run on any computer.

Venue Crawler. Foursquare and Places have API functions that search for venues based on certain parameters. Given a set of coordinates, and a category description (e.g., bar), both services return a list of relevant venues nearby. The *Venue Crawler* takes as input a set of coordinates and searches for different categories of venues through API calls. We use this to collect venues across different countries. For every venue we collect the name, venue ID, and location coordinates. When submitting a user check-in to the LBS, the request must contain the venue's ID. We submit the venue's coordinates as our user's coordinates (unless we want our user to appear as being at a distance from the venue).

User Authentication. This part of our central component is responsible for authenticating the user to the LBS. It takes as input the user's access token used for authenticating with the service. We can select to authenticate with the access token that was created for use by our custom application, or the one extracted from the official application. Based on which one we select, we can appear to be sending the check-ins from the custom application or the official one.

Check-in Manager. This implements the core functionality of our system as it simulates a user checking into venues. It takes as input a list of venues that will be used for the arbitrary check-ins, and a set of values that configure the user's behavior. Several aspects of user behavior can be configured to explore the heuristics deployed by a LBS.

8.5 Measurements - Foursquare

Foursquare has implemented a system, which they refer to as "cheater code", for detecting users that post fake check-ins. While the mechanism has not been disclosed, according to Foursquare [46] detection is based on information from:

- Technology: Information collected from user phone and location.
- Software: Information collected from the official Foursquare application and system.
- An advanced detection algorithm.

A check-in is accepted even if it triggers one of the heuristics, however, it is not taken into consideration for mayorships. By discovering the heuristics used to detect fake check-ins, we are able to model an attack having the maximum impact, without being detected. We follow a black-box testing approach using a set of test profiles that post a series of arbitrary check-ins. By modifying the behavior, we are able to trigger the heuristics used and identify their thresholds. Here we describe our approach for each case, as well as the maximum

Heuristic	Description	Threshold	Range	IT | IN
Maximum check-ins	Number of check-ins in specific time window.	5 check-ins 8 check-ins 49 check-ins 90 check-ins	$t \leq 1$ min $t \leq 15$ mins $t \leq 24$ hours $t \leq 72$ hours	-
User speed	Elapsed time and distance traveled between check-ins.	4 km/min 25 km/min	dst $<$ 100 km dst $\geq$ 100 km	0.2% | 49% 3.0% | 37%
GPS distance	Distance between coordinates of user and venue.	200 meters	-	1.1% | 5.5%

Table 8.1: Detection heuristics and the respective thresholds after which check-ins are flagged as cheating.

thresholds allowed for the heuristics we detect. In Table 8.1 we provide a short description of the heuristics we discovered, along with the threshold values after which check-ins are flagged.

8.5.1 Service Responses

All check-ins posted by our system through the API, receive a response message. If a check-in is considered legitimate, Foursquare returns a message verifying the check-in, while ones that are considered cheating receive an error message. By configuring our users to perform specific actions with precise timing, we can model various types of behavior. Based on the response messages, we know when a specific heuristic was triggered, and based on the user actions we discover the conditions under which it happened.

GPS distance: the user's location exceeds the maximum acceptable distance from the venue. The distance is calculated based on the user's coordinates sent by the application, and the venue's coordinates in the Foursquare database.

High speed: the user's speed exceeds the maximum threshold. Speed is calculated based on the time elapsed between the current check-in and the previous *legitimate* check-in, and the distance of the respective venues.

Rapid fire: the user exceeds the maximum number of acceptable check-ins for a certain time window. This is triggered by two different behaviors. First, when a user sends a burst of check-ins in a short amount of time. Second, when the user exceeds the maximum acceptable number for one day. This mechanism follows a 24-hour sliding window, and is not reset at the beginning of each calendar day. Whenever a user checks in, Foursquare examines the amount of check-ins the user has committed in the previous 24 hours.

8.5.2 Device-based heuristics

The official application queries the device for the GPS coordinates, which are correlated with those of the venue. Even though heuristics aim to prevent fake logins, users are allowed to check-in before they arrive at a venue or after they leave. However, in large venues, a user may be present but not at the exact coordinates the system has registered. Additionally, cell phone GPS readings cannot always identify the location with high accuracy, and include an "accuracy" parameter as an indication of a margin of error. Applications can query the device if such information is needed. The API check-in function has an optional field for

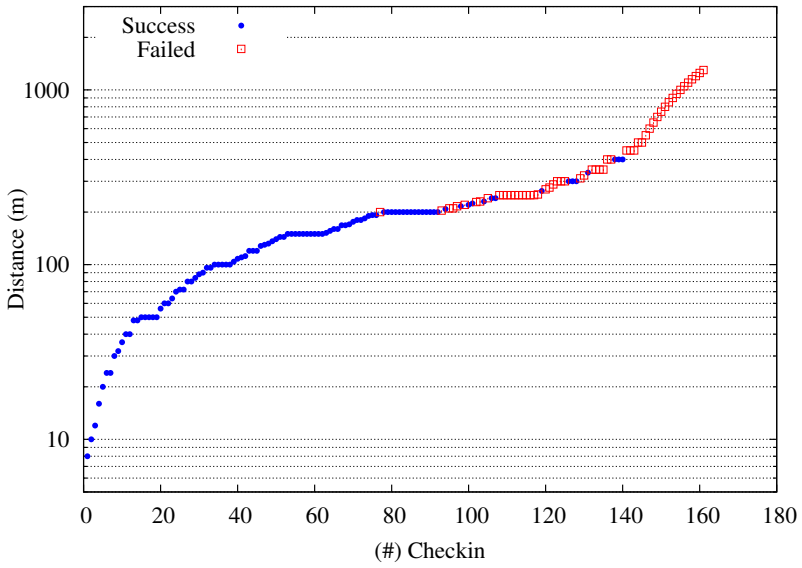


Figure 8.3: The distance between the GPS coordinates of the user and the venue.

this information. To compensate for low accuracy readings, heuristics allow check-ins from a certain distance. While a reasonable threshold that allows reasonable distances will facilitate legitimate users, high-distance tolerance enables users to cheat without spoofing the GPS data. To detect the threshold we conduct *Experiment A*:

- The system takes as input a predefined list of venues and their coordinates.
- After each check-in, it waits for a specific amount of time, before the next check-in. The amount of time is large enough, to avoid triggering heuristics that enforce constraints on user speed. It calculates a set of coordinates for the user that are X meters away from the venue, using Vincenty's formula [234]. This formula is a well known method used in geodesy for calculating the distance between two points on the surface of a spheroid.
- Our system starts with an X value of 0, and increases X by Y meters after every check-in. We run multiple rounds of experiments with different values of X,Y.

Results were consistent across all experiments conducted over a period of 6 months. The results from a representative experiment can be seen in Figure 8.3. Check-ins are accepted from up to 200m away. Once the distance between a user's reported position and that of the venue exceed that threshold, the check-in is flagged as cheating and receives the "GPS distance" error. A high threshold makes it trivial for users to check into venues without being near them. We discuss the accepted check-ins for over 200m in Section 8.5.4.

8.5.3 User-behavior heuristics

A series of variables, based solely on the user's behavior, are evaluated before a check-in is deemed legitimate.

Maximum check-ins. Foursquare sets a limit on the number of check-ins in a given time window. To estimate the threshold of this heuristic we setup *Experiment B*:

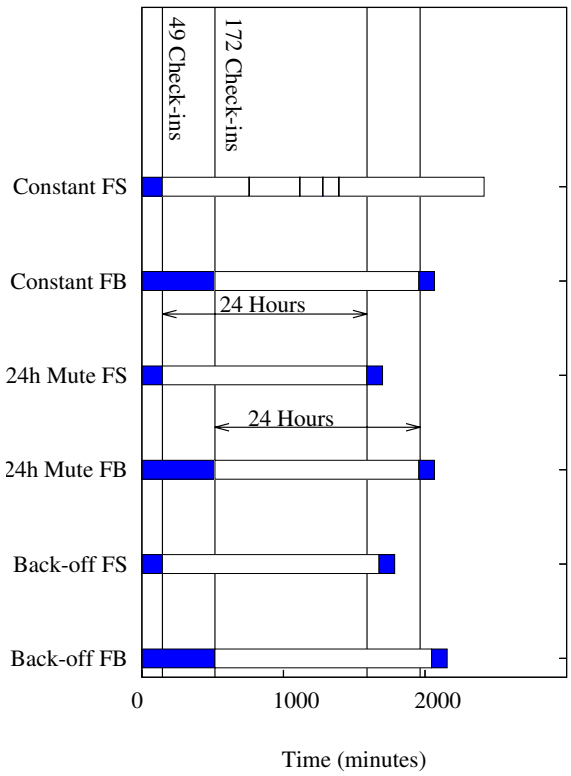


Figure 8.4: The maximum check-ins allowed per user. We depict three strategies. They all simulate a user that checks in with a constant rate, until it is prohibited by the service, upon which the strategy changes. (a) user performs check-in attempts with the same rate, (b) user pauses for 24 h, and (c) user retries to check in following an exponential back-off.

- Our system takes as input a predefined list of venues and their coordinates. It places the user at the venue's exact coordinates, and after each check-in, waits for X seconds, before the next check-in.
- If a check-in receives the "rapid fire" error message we follow one of three different strategies.
  1. *Constant*: follow the same pattern, and sleep for X seconds after each check-in.
  2. *24h Mute*: sleep for 24 hours before attempting another check-in.
  3. *Exponential Back-off*: whenever an error message is received, X is doubled. In all 3 cases, whenever the user successfully checks-in again, X is reset to its original value.
- We repeat this procedure with different values of X.

We summarize our results in Table 8.1. Different thresholds apply for the number of check-ins users are permitted to make in certain time windows. Specifically, constraints are set for prohibiting bursts of check-ins by allowing a small number of check-ins to be posted

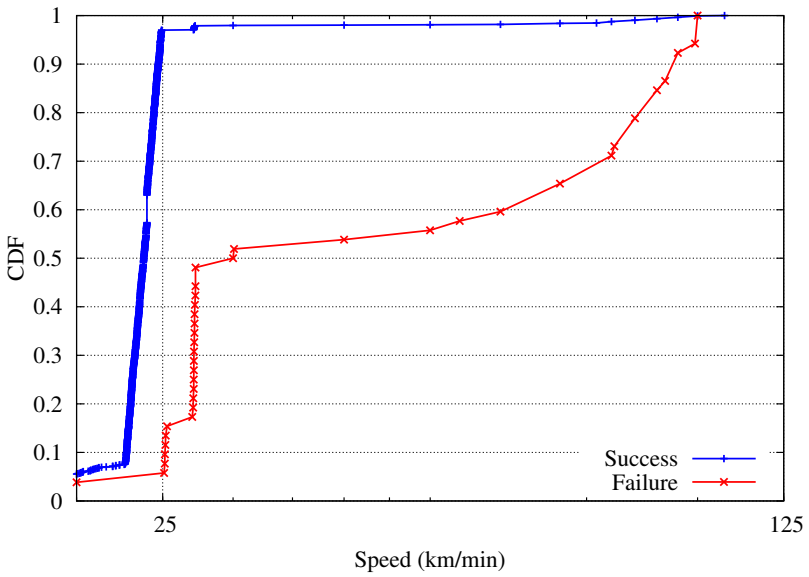


Figure 8.5: Speed heuristic for distances over 100km.

within a time window of 15 minutes. Foursquare also sets a limit on the number of check-ins a user can commit in a 24 hour period. Figure 8.4 shows the results for the three approaches (marked with FS), with the bold sections indicating successful check-ins. We can see that all users receive errors after 49 check-ins. Foursquare does not reset the number of check-ins for a day at a specific moment in time, but checks them within a 24-hour sliding window. The user that never stops bombarding the service with check-ins can escape the ban period only for very short time windows (the short lines in the Constant FS bar), because even check-ins that receive error messages count as part of the 49 allowed. The user that pauses for 24h after the first error, escapes the ban period faster than the other two. Foursquare also examines the check-ins committed in a 72-hour window, and allows only 90 check-ins. Even if a user commits 49 check-ins in the first 24 hours, he cannot exceed the threshold of 90 in a given 72 hour window. The 3 timing strategies used are not the most efficient in regards to attacking an LBS, but aim at revealing the detection mechanism deployed by the system. In Section 8.7 we describe our adaptive attack that uses a more efficient timing strategy.

User speed. This heuristic measures the geographic distance traveled and the time elapsed between two consecutive check-ins. The "high speed" error is returned when the acceptable thresholds are exceeded. This is the main heuristic for detecting fake-location check-ins, as it is impossible to post legitimate check-ins that exceed the thresholds set by Foursquare. We conduct *experiment C*:

- Our system takes as input a predefined list of venues and a user velocity V.
- After each check-in, it calculates the exact distance X to the next venue and how many seconds T it must wait before checking into the next venue, so the user will appear to be traveling at a speed V.
- We run multiple rounds of this procedure with different values of X and V.

We model our users to appear as traveling at any speed we want. Depending on the value of X, Foursquare allows different speeds. For distances up to 100 kilometers users are allowed

to travel at 4 kilometers per minute (approximately 150 miles per hour). For any distance longer than 100 km, users can travel at any speed below 25 kpm (approximately 932 mph) which is much faster than commercial airplanes (that average 600 mph). Figure 8.5 shows the results from a representative set of experiments for distances longer than 100 km. When conducting experiments with user speeds over the threshold, several interspersed check-ins are accepted. This is because Foursquare uses the last accepted (i.e., not flagged as cheating) check-in as the user's last location when calculating the elapsed time and traveled distance. Thus, it perceives that it took a longer time than it actually did to travel the distance. This results in calculating a speed that is below the threshold, and accepting the check-in. When the speed is slightly over the thresholds, the number of accepted and flagged check-ins are almost equal.

For example, consider a case where we had three venues A, B, C. The distance of B from A was equal to the distance of C from B. The user checked-into venue A and waited the appropriate amount of time before checking into B to imitate a travelling speed of 1,000 mph. The user waited for the same amount of time before checking-into C. While Foursquare evaluated the check-in for venue B as cheating, it accepted the check-in for venue C.

Traveling distance constraints. We explore if any constraints apply for the distance users can travel. We conduct *Experiment D*, which is the same as *Experiment C* except that we use a list of venues, located in different countries.

- Our system takes as input a predefined list of 15 venues, each one located in a different country.
- After each check-in, our system calculates the exact distance X to the next venue. Then, depending on the distance, our system calculates how many seconds T it must wait before checking-into the next venue, so the user maintains a steady speed V.
- We run multiple rounds of experiments with different values of V.

We model our user to travel right below the speed threshold, at 24 kilometers per minute. At that speed, all check-ins posted by our system were accepted, and our user covered a distance of 36,120 kilometers (which is 90% of the circumference of Earth) in 25 hours. As our user traveled steadily for over a day, we conclude that there are no heuristics for imposing constraints on the distance a user can cover.

History heuristics. According to Foursquare, their system relies on a user's (check-in) history to decide upon evaluating a new check-in, only in the cases where the request does not contain the user's coordinates [32]. This can happen in cases where the post comes from a device with no GPS capabilities, or an application that does not send coordinates. Nevertheless, we explore if a user's check-in history affects the detection mechanism when the application contains the user's coordinates. Specifically, if thresholds are adjusted and there is a higher possibility of the heuristics being triggered, if previous user actions have been flagged as cheating. To discover the exact way with which check-in history affects the thresholds would require a very large number of fake accounts that we could configure to exhibit varying behavior. After an adequate "training" period that would create a sufficient history of check-ins, we could compare heuristic thresholds for different behaviour models and extract a more fine-grained model of how user history affects the cheating detection algorithm. However, creating such a large number of fake accounts is impractical. Therefore, we rely on a limited number of fake accounts used in the previous experiments to attain a coarse-grained

view of this heuristic. We compare the thresholds for three accounts with varying behavior: an account with no check-in history, one with many legitimate check-ins and a few that exceeded the maximum number allowed, and one that greatly exceeded all thresholds. We repeat our previous experiments with all accounts running simultaneously with the exact same variables. Results showed that thresholds are the same regardless of the user's cheating history.

Cheating penalties. We also found that Foursquare does not impose any penalties on users that have triggered the heuristics, no "ban" periods are enforced, and heuristic thresholds remain the same.

8.5.4 Heuristic inconsistencies

During our experiments, we detected two types of inconsistent behavior of the heuristics and here we present certain examples. In the first case heuristics are triggered while we remain beneath the thresholds (inconsistent triggering), and in the second case their mechanisms are not triggered by behavior that exceeds the thresholds (inconsistent non-triggering). While they may not be errors of the detection mechanism in all cases, they do present an inconsistent behavior in regards to the thresholds calculated based on the extensive number of experiments we have conducted. Nonetheless, we refrain from the standard terms of false positives and false negatives used for evaluating detection mechanisms. The last column of Table 8.1 shows the percentage of these cases. In the user speed experiments, the ratio of inconsistently accepted check-ins is high due to the way Foursquare calculates user speed, as explained in *experiment C*. Here, we omit these and present some other incidents as examples.

Inconsistent triggering (IT). In experiments with a speed beneath the threshold (e.g., 0.26 kpm), some check-ins received the "high speed" error. In several cases we received the "GPS distance" error even though the user had the exact coordinates Foursquare returns for the venue. If we immediately repeated the check-in, it was deemed legitimate.

Inconsistent non-triggering (IN). In several cases our system was able to check in our users from as far as 900m away. While a velocity of (right below) 25 kpm was the maximum speed our user could exhibit without any check-ins being flagged, in the experiments with higher speeds some check-ins were considered legitimate. We had a check-in accepted with a speed of 107 kpm, without an intermediate flagged check-in to alter the speed calculation by Foursquare.

8.6 Measurements - Facebook Places

We follow the same methodology as with Foursquare. Through the official Facebook application for Android, we gather the required POST URLs and fields to simulate the logins and check-ins We noticed that, after the login process, no auth token is needed for check-ins. Our system uses `cUrl`[3] for parsing the required URLs and posting the data, without using Facebook APIs. After a successful login, a cookie is stored into a local file and every checkin attempt uses this cookie, to avoid multiple logins.

Service Responses. Places also returns error messages when a detection heuristic is triggered, which we use to explore the detection mechanisms deployed by Facebook.

"The checkin is a significant distance from the users previous checkin in too short a timeframe." when the user is traveling too fast. The distance is calculated based on the user's

[3]http://curl.haxx.se/

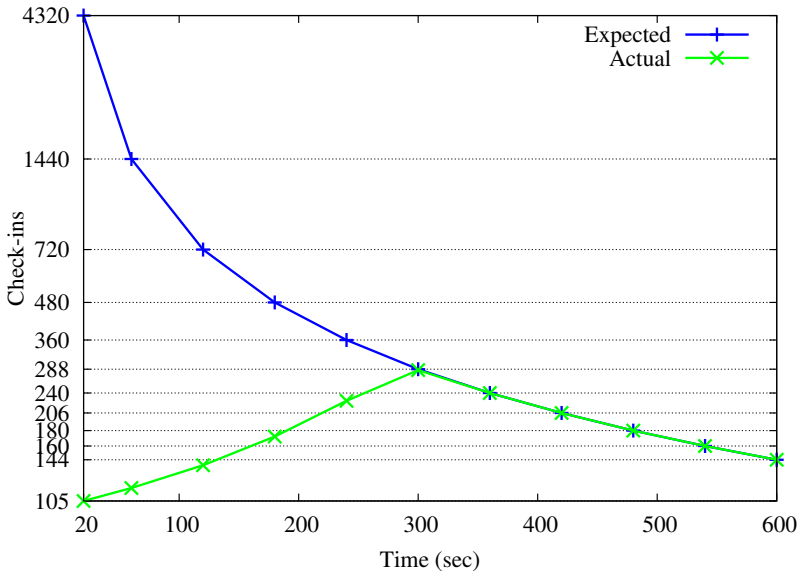


Figure 8.6: Expected vs. achieved check-ins for different time intervals T between check-ins for 1 day.

reported GPS coordinates sent by the application and those of the previous check-in.

"You checked in to too many places in a short amount of time. You will be allowed to check in to more places after some time has passed." when the user exceeds a number of allowed check-ins. This mechanism is implemented with a 24 hour sliding window, and does not take into account the actual date. Thus, whenever a user posts a check-in, Foursquare checks the amount of check-ins posted by the user in the previous 24 hours.

Device-based heuristics. We replicate *Experiment A*, as outlined in Section 8.5, to reveal the maximum distance from which users can check into venues. For a series of venues, we gradually increase the user's distance from the venue's location. For each check-in attempt, we send the venue's ID and the user's location coordinates. We did not receive an error message for any of the distances we tried. As distances increased even more, we were able to check our users into places with coordinates located on different continents. Our experiments show that the user's coordinates are never compared to those of the venue when a user checks in, and any coordinates are accepted. We setup *Experiment E*, to demonstrate how this can be exploited by an attacker:

- Our system takes as input a list of venues from around the world, and a set of user coordinates. For every venue, the user checks-in with the same location coordinates, regardless of the venue's location. The user waits for 1 minute between check-ins.

With this experiment, we are able to check our user into venues around the globe in just a few minutes. As the user always sends the same coordinates for his location, when the system compares his new position to that of his previous check-in, it detects no change and the speed heuristic is never triggered. Thus, *an attacker can completely bypass the traveling speed constraints and check into venues around the globe with unlimited speed.* After further investigation, we found a bug report [18] submitted to Facebook two weeks prior, for the Graph API. The bug report was acknowledged and received an "assigned" status, but was

Algorithm 8.7.1: ATTACK(N)

```
L ← ListOfVenues(N)
c ← 0
while
do { p ← L.Dequeue()
     n ← MayorCheckinsAtVenue(p)
     m ← OurCheckinsAtVenue(p)
     if m ≤ n
       then { checkin ← Checkin(p)
              CList.Enqueue(checkin)        (1)
              c ← c + 1
              t ← AdjustSleep(CList, c)     (2)
              if c = MAX
                then { c ← c − 1
              Sleep(t)
     L.Enqueue(p)
```

Figure 8.7: Pseudo-code of the actual attack.

closed one year after its submission, without any fix being released. This can be attributed to the fact that the report describes the problem in the check-in mechanism without pointing out the security implications of this bug, and how it can be exploited to bypass the other detection mechanisms.

User-behavior heuristics. As we can completely bypass the speed heuristic, we repeat *Experiment B* to identify the limit of acceptable check-ins. The results reveal that if this threshold is exceeded, an exact 24 hour ban is applied. Figure 8.4 compares the check-ins allowed by Foursquare and Places for a time window between check-ins of 180 seconds. We also identify that the threshold is not constant for different intervals between check-ins. We repeat the experiment with different time intervals T for 24 hours each. Subsequently, we calculate the expected check-ins, which are the ideal number of check-ins that should be successful if no heuristic is applied, versus the actual successful ones. As shown in Figure 8.6, prior to approximately $T < 300$ seconds, the actual check-ins are less than the expected ones, since the heuristic is triggered and a 24 hour ban is applied.

8.7 Attacking LBS

An adversary with the knowledge of the detection mechanisms can create an adaptive attack that maximizes its impact while remaining undetected. In the case of Foursquare a potential attacker would try to acquire the mayorship in top venues around the world, discouraging other users from competing. We focus on Foursquare, because there is clear notion of game incentives. However, it is quite generic and, with minor modifications, can be adapted to other LBS. The following aspects led to the design of our strategy.

Timing between check-ins. Foursquare restricts the number of check-ins allowed, based on a 24 hour sliding window. If a check-in receives a "rapid-fire" error, it is not eligible for

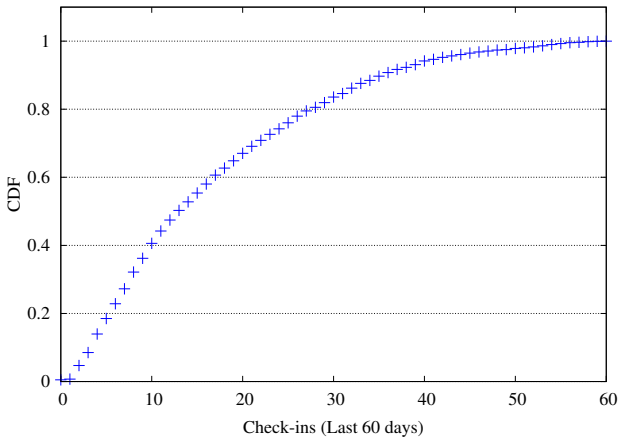


Figure 8.8: Number of check-ins for mayorship in popular venues.

points but still counts as one of the check-ins allowed. The attacker must keep account of the timestamps of his check-ins to calculate their number in the last 24 hours. As long as the count of check-ins of the last 24 hours is 30 (since 90 are allowed in 72 hours), no check-ins must be attempted. Once the number reaches 29 he can check in once again. This way, one can commit 1,800 check-ins in a 60 day period.

Venue selection. Selecting the target venues depends on several variables, and can lead to different strategies. An attacker can select venues depending on the mayor rewards, or can target venues where mayors have a small number of check-ins. The adversary can acquire this information through the API.

Number of check-ins needed for mayorship. When a user checks into a venue where he is not the mayor, the response message by Foursquare also indicates how many more check-ins are needed to acquire the mayorship of the specific venue. Based on this information, the adversary can decide if the cost is too high and target other venues.

Minimizing necessary check-ins. Checking into venues after having been crowned the mayor, results in unnecessary check-ins that should be used for other venues. The adversary can stop checking into a venue once he has acquired the mayorship, and only resume if he temporarily loses it. The remaining check-ins can be used for other venues.

Only check-ins that do not trigger one of the heuristics are considered valid and can result in mayorships. We design a strategy to use the limited number of check-ins effectively. Its pseudo-code is presented in Figure 8.7. The system takes as input a list of N arbitrary venues. A counter c holds the number of check-ins made in the last 24 hours. After selecting the next venue from the list, we retrieve the number of check-ins n the mayor of that venue has. If we are mayors of the venue, we do not check-in and add the venue to the end of our list. If we are not, we check-in and save the relevant information (line 1). We increase the number of check-ins and then run the function to *adjust* the waiting time (line 2): If we haven't reached the MAX allowed check-ins for the last 24 hours, we set t to our normal small sleep interval. If we have reached MAX check-ins, we retrieve the info of the check-in which is located $MAX - 1$ positions from the end. We calculate how much time t our system has to sleep so it "wakes up" 24 hours from that check-in. We decrease our counter, sleep for t, and add the venue to the end of the list.

An adversary with the goal of disrupting the system and deterring legitimate users from participating, will target the most popular venues as this will impact the largest number of users. While obeying the thresholds, in the worst case scenario where each mayorship requires 60 check-ins, the attacker can acquire the mayorship of 30 venues with a single account. In February 2012, we collected the number of check-ins of the mayors of 2,420 of the most popular venues in New York through the API function that returns the most popular venues for a given location. As shown in Figure 8.8, 90% of the venues had a mayor with 36 or less check-ins, and only 2.2% had over 50 check-ins. The average number of check-ins required for mayorship was 17. While it might be higher than the average across all venues, since it reflects activity for popular venues in a metropolitan area, it provides a rough estimation of the average number needed to acquire a mayorship. Thus, an attacker following our attack algorithm can use the 1,800 available check-ins to sustain mayorship in 105 venues, on average, with one account. Based on that, an attacker can maintain constant mayorship in all venues with less than 10,000 accounts.

The VeriSign iDefence Intelligence Operations Team released a report about a cybercriminal selling 1.5 million Facebook[4] accounts [2]. According to the report, the cost of 1,000 accounts without any contacts was $15. For compromised accounts with friends the price ranged from $25 - $45. Assuming such prices are representative, an attacker can acquire the needed number of accounts to sustain mayorship across all Foursquare venues with as little as $150 - $450. Furthermore, Trend Micro released a report [73] about the Russian underground where 2,000 bots can be bought for $200. That number of bots is more than enough for deploying the 10,000 accounts. *Overall, an attacker with the knowledge of the detection heuristics and their respective thresholds, can acquire mayorships across all venues and have a significant impact on Foursquare with less than $1,000.* Similarly, any LBS can be severely damaged with minimal resources.

As the attack is carried out by multiple accounts with legitimate behavior, each targeting a small subset of venues, Foursquare will not be able to distinguish them from other accounts. Even if the heuristics are made more restrictive, the attack variables can easily be modified to remain beneath the new thresholds. Making the heuristics too strict, will have a negative impact as legitimate users will be greatly inconvenienced. Thus, it is evident that detection heuristics are not effective against large-scale fake-location attacks and other types of countermeasures must be implemented.

8.8 Countermeasures

Fake-location attacks are possible because clients can communicate an arbitrary geographical position to the service. This is mainly due to the fact that the actual geolocation cannot be (currently) verified by the service. First, we propose three countermeasures that can hinder attacks by validating the user's location. Next, we discuss the inefficiency of detection mechanisms.Finally, we present our proof-of-concept implementation of Verified Check-in, a hybrid solution of hardware and software.

[4]Foursquare allows to sign-up with a Facebook account.

8.8.1 Validating user location

Ensuring user presence. One approach is to enforce verification based on information provided only at a geographical position. By requiring users to also submit information that is only available at a location, the service can validate the user's presence. One can take advantage of the NFC capabilities of smartphones, which enable communication between devices within a very short range (i.e., a few centimeters). By deploying a NFC device at venues, the LBS can validate user check-ins. We provide a detailed overview of our proof-of-concept implementation in section 8.9. Interestingly, Foursquare recently introduced unpowered NFC tags to identify the venue and prompt the user to check-in [30]. This minimizes interaction as users need only swap their device over the tag. By building upon this idea, we can hinder fake-location attacks.

Temporary codes. The service can generate a temporary code for venues that are valid for certain time (e.g., one day), and are only obtainable at the venues. This can be a string, a QR code [135], or even a NFC tag. The QR code and NFC tag have the advantage of users being able to scan it with their device, while the string can be used by users with feature phones. There are various ways to implement this mechanism. The venue can have a monitor with the codes or print-outs at several spots. A cheaper way to implement this, is using a wireless access point that periodically advertises a particular `SSID`, which is used for authenticating the coordinates to the service. While this method has the advantage of not requiring dedicated hardware,it is susceptible to **wormhole attacks** [155] where users share the code with other individuals that will be able to check-in without actually visiting the location. Another drawback is that it can only be used in commercial venues and not public places. Alternatively, a location proof scheme that uses existing access points can be implemented [187, 210]. However, such solutions can be bypassed from users that are within the range of the access point, but not at the actual venue.

Third party verification. In the third countermeasure, the user's location is verified by a third party. Seeking for generic protocols that can authenticate user coordinates is challenging. Currently, a client's location can be verified by telecommunication providers (they know the cell the device is connected to), and large IT vendors, such as Google, that have constructed extensive maps of wireless access points around the globe. However, the location information is not accurate enough to verify the user's presence within a venue, but only within a larger area.

8.8.2 Adapt existing detection mechanisms

As we demonstrated, existing heuristic implementations are either too relaxed (Foursquare) or can be bypassed (Places). However, even with more restrictive thresholds, such mechanisms cannot prohibit cheating. Furthermore, they cannot defend against system-wide threats carried out by multiple accounts that are indistinguishable from legitimate ones. On the contrary, our NFC countermeasure can effectively hinder such attacks and provides an affordable solution for sustaining the viability of the emerging business model of LBS.

Penalties for cheating. This is an efficient mechanism for discouraging legitimate users from cheating, but is ineffective against system-wide threats. Due to false positives, penalties should be imposed when users repeatedly trigger the heuristics, in which case they should face timeout periods where no check-ins are accepted by the system. After that time window, users can once again use the system normally. If a user continues to exhibit cheating behavior,

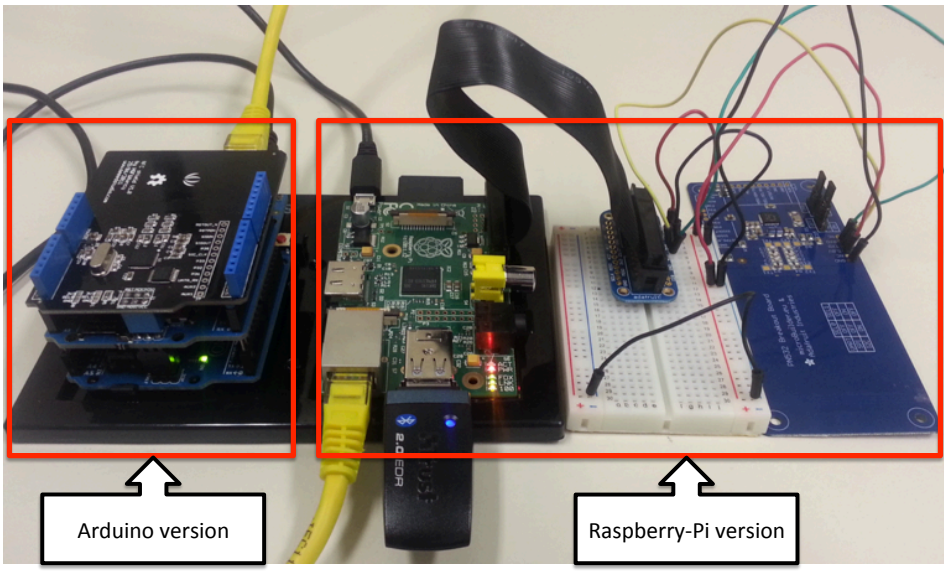


Figure 8.9: Our two Verified Check-in prototype implementations. On the left, the Arduino version, and on the right the Raspberry-Pi version.

each new time window will be greatly increased. If the cheating persists, the user should be permanently banned from the service.

Revocation. As a measure of preventing users from acquiring mayorships through fake-location attacks, Foursquare has introduced a feature that allows venue owners to revoke the mayorship of users which may have cheated. This mechanism does not assist in identifying or preventing fake-location attacks, but in discouraging potential cheaters. This feature might be effective in certain cases, yet there are many conditions where it is not applicable, like in venues with many simultaneous customers (e.g., clubs, shopping malls).

8.9 Implementation of Verified Check-in

We present the details of **Verified Check-in**, our proof-of-concept implementation of the NFC server countermeasure. Affordable electronics frameworks are a rapidly growing market, and we selected two of the most popular devices. First, the Arduino board, an open source electronics prototyping platform, which can be extended through various modules that provide specific functionality. We used Arduino Uno and the Seeedstudio NFC Shield with a total cost of about $50 at retail price. Second, the Model B Raspberry-Pi, an ARM GNU/Linux box, using the Adafruit NFC breakout board with a total cost of $75. Our prototype implementation for both versions can be seen in Figure 8.9.

To evaluate our testbeds, we developed an Android application which implements the user functionality. The application communicates with the NFC Server, using classes from the

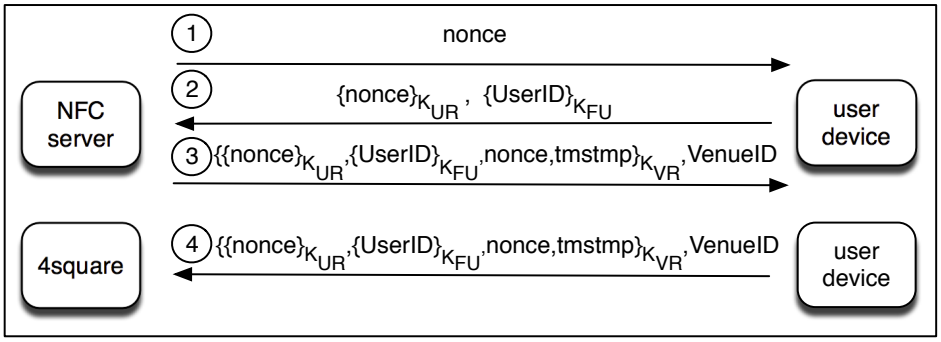


Figure 8.10: Data exchanged during the check-in.

`android.nfc` package. Arduino was programmed with the LLCP-SNEP protocol implementation for P2P communication. For the Raspberry-Pi we used the `libnfc` and `openssl` libraries. Our user device was a Samsung Galaxy S3 with a stock Android version 4.1.1.

Our solution relies on cryptographic primitives for securing communication between the NFC server and user device and prevents different types of attacks. Upon activation of the mobile application, venues and users calculate a set of asymmetric keys. Foursquare must receive a copy of the public keys, and all venues and users save a copy of Foursquare's public key. After setting up the venue account, copying the keys on the NFC server and synchronizing the internal clock through NTP, no Internet connectivity is required.The UserID and VenueID are already used by Foursquare, as parameters in the API calls.

8.9.1 Verified Check-in protocol

In Figure 8.10 we present the information exchanged in each step of our Verified Check-in protocol during the check-in process.

1. The NFC server sends a random nonce to the user.
2. The device encrypts the nonce using the user's private key K_{UR}, and the UserID using Foursquare's public key K_{FU}, and sends them both to the NFC server. If they are not received by the server in an acceptable time window, it terminates the process.
3. Using the venue's private key K_{VR}, the NFC server re-encrypts the encrypted UserID and nonce, along with the nonce in cleartext and a timestamp. These, as well as the VenueID in cleartext, are sent to the user device.
4. The device sends the data to Foursquare, that uses the VenueID to retrieve the venue's public key K_{VP} and decrypt the ciphertext. If the timestamp is valid, Foursquare uses its private key to decrypt the UserID and verifies it is that of the user that sent the check-in. Then it uses the user's public key to decrypt the nonce value. If it matches the one sent by the venue, the user is checked-in.

8.9.2 Security Analysis of the Verified Check-in protocol

We designed our protocol to secure the check-in process against a series of attacks. While our main goal is to hinder fake-location attacks, Verified Check-in secures the check-in process

against a series of attacks. We discuss how our protocol is effective against each type.

Fake-location attacks. The attacker creates a bogus check-in request for a venue, while being at a different location. However, during the normal check-in process the information sent to Foursquare contains a timestamp encrypted using the venue's private key. Thus, the user cannot create a valid check-in (even if he has stored the received information from a previous check-in).

Wormhole attacks. An attacker located at a venue exchanges information with an accomplice, so he can perform a check-in as well. The NFC server completes the protocol only if the user sends the response within the acceptable time window. However, the nonce cannot be predicted or created in advance, and the challenge can't be relayed in time. The attacker can also use a device that mimics an NFC server and try to check the user into a venue X other than the one he is at. Again, the NFC server at venue X will not receive the challenge in time.

Impersonation attacks. An attacker with stolen credentials (name, password) tries to check into a venue as the victim. In this case, the attacker will not be able to successfully commit a check-in as he does not have the user's private key. When Foursquare decrypts the cyphertext using the victim's public key, the resulting cleartext will be garbage and not the correct UserID.

Check-in inflation attacks. A venue wants to post a check-in for a user without the user being there, so as to inflate its number of check-ins and appear more popular. This would allow a venue to appear in the lists of popular venues for a given location provided by Foursquare, which can result in an increase of actual customers. To conduct this attack, the venue will attempt to create a fake check-in request and use that to check a user in, by sending it to Foursquare. However, as the user encrypts a timestamp along with the UserID using his private key K_{UR} during a legitimate check-in, the venue will not be able to create a valid check-in without knowing the user's private key. Furthermore, Foursquare considers only one check-in per day valid for a specific venue. Thus, even if the venue posts multiple check-ins during the time window that the timestamp is valid, only one check-in will count.

Eavesdropping attacks. A physically present eavesdropper passively monitors the communication between the users and the NFC server to discover the users' identity. However, he is only able to acquire the VenueID which is publicly known, since the UserID is encrypted with Foursquare's public key.

System-wide Sybil Attacks. The attacker aims to disrupt the whole system and drive legitimate users away by acquiring the mayorship of all (or most) venues using multiple accounts [126]. As shown in Section 8.7, with the existing defense mechanisms, this can be done with minimal resources. With our countermeasure this attack can be deterred, as our NFC server imposes physical constraints on the check-in process. The attackers will have to physically visit each venue (practically impossible) to perform the check-ins, thereby negating the concept of a fake check-in.

Targeted Sybil Attacks. The attacker targets a few venues that might offer deals to all customers after a certain number of check-ins. The attacker can utilize several accounts to collect multiple offers. Even with our countermeasure deployed, one can still perform this attack. This can be done with a smartphone that contains the passwords and keys of all the attacker's accounts (or those of accomplices). Nonetheless, our countermeasure will be able to greatly mitigate such an attack, as it imposes physical and time constraints on the check-in process. Due to the NFC technology, the attacker will have to stay next to the server for a unnatural amount of time to check-in a large number of accounts.

Keysize	512	1,024	2,048	4,096
Arduino	25,056	224,279	1,587,550	NA
Rasp.Pi	2.745	3.228	5.130	12.150
Galaxy	2.265	2.834	5.042	12.501

Table 8.2: Average encryption time (ms) for different RSA key sizes (bits).

8.9.3 Performance analysis of Verified Check-in implementation

Here we discuss the performance of Verified Check-in in regards to various aspects of our implementation and discuss the potential overhead for a LBS deploying our system.

Encryption. An important factor that affects the applicability of our solution is the time needed for the data encryption. Table 8.2 presents the average times (over 100 runs) for applying RSA encryption to a buffer using keys of various sizes. The Arduino presents the worst performance due to its limited computational capabilities and RAM size, with similar times to those reported in [92]. Even for small keys, the time required for the user device and NFC server to stay in range is not acceptable in realistic scenarios. For a 512-bit key (which is very weak), the encryption process takes 25 seconds. Thus, the Arduino board is not a suitable solution. On the other hand, the Raspberry-Pi server is very efficient and even with 2,048-bit keys encryption takes merely 5 ms.

Check-in process. To initiate the check-in process, the user has to tap the device's screen to enable the NFC data exchange. Once tapped, our application sends a simple message that informs the NFC server to start the protocol. Currently, Android requires the user to tap the screen for authorization before data is sent over NFC, and multiple messages can only be sent in batches. As the NFC server has to enforce a strict time window for the challenge-response step, no user interaction should be needed between steps (1) and (2), as that would result in a window large enough for a wormhole attack. However, the data sent in step (2) of our protocol is based on the data received in step (1) and can't be sent in a batch. Thus, the user is required to tap the screen a second time. This is very restrictive for building NFC apps, as it does not allow multiple steps of communication between devices. This has been reported by developers [5], was acknowledged by Google, and is awaiting a fix. To overcome this limitation, we also implemented a version that uses NFC to pair the devices, and sends the protocol data over Bluetooth. This version only requires one tap to initiate the pairing and everything else is done automatically. Nonetheless, we expect this to be fixed soon, enabling our NFC-only approach.

In the Bluetooth version, using 2,048-bit keys and a 32-byte nonce, the entire check-in process lasts 105 ms (average over 100 runs). Based on the encryption times and the time needed to send the data (28.7 ms per message), we set the time window for the challenge-response step to 45 ms. After sending the nonce, the NFC server terminates the process if the response is not received within 45 ms. We plan on conducting a study using a variety of smartphones to calculate the time needed for the encryption, to select a value suitable for real-world deployment. Overall, the performance of Verified Check-in renders it an ideal solution for LBS.

LBS workload. Public-key cryptography is considered computationally expensive. Verified check-in requires commodity hardware (the NFC server or the user device) to occasionally perform RSA encryptions. Assuming a 2,048-bit key, the computation time for the NFC server and user device is realistic. However, Foursquare must validate thousands of check-ins

per minute, as users conduct a few million per day. Fortunately, hardware acceleration for cryptographic operations has evolved. Consider that 7 years ago [72] Sun's UltraSPARC T1, equipped with a Modular Arithmetic Unit for RSA, performed 20,425 signature verifications per second with a 2,048-bit key utilizing all 32 cores. Decryption can be further sped up by using GPUs [169] or modern `x86` CPUs, with operations needed by cryptographic algorithms implemented in the hardware, and encryption can be handled at line speeds [177]. Others [105] also argue that cryptographic operations at line speeds are no longer an issue.

8.10 Limitations

Here we discuss certain limitations of our approach.

Check-in Serialization. Our current implementation allows only one check-in at a time. While this will not be restricting in normal cases, it may prove to be troublesome in situations where a very large number of users are present at a venue at the same time and wish to check-in. In such cases, multiple Verified Check-in servers might be required to handle the check-in requests.

Cost of hardware components. Components to build our system cost $75 at retail prices. While this might seem high, keep in mind that a LBS can purchase bulk quantities of the components at much lower prices. Thus, they can provide Verified Check-in to collaborating venues for a very low price, with the ultimate benefit of imposing fairness which will ensure a robust check-in economy.

Multiple user devices. When a user has multiple devices, the keys have to be manually copied to the other devices, which poses an overhead on the user's side. However, assuming the user will check-in only from a few mobile devices (smartphone, tablet) replicating keys will be rare, usually for device upgrades. Eventually migration tools will be available for transferring old data to the new devices.

Chapter 9

Related Work

9.1 User accounts: Social Authentication

Previous work showed that information available in users' profiles in social networks can be used to break authentication mechanisms, or deduce information that threatens their privacy. A study performed by Rabkin [206] attempted to assess the security properties of personal knowledge questions that are used for fallback authentication. In §3.1.5 we discuss a similar study, although focused on Facebook SA. Rabkin argues that since such mechanisms owe their strength to the hardness of an information-retrieval problem, in the era of online social networks and the vast availability of personal information, their security is diminishing. In this study 12% of the sampled security questions from online banking sites is automatically attackable (i.e., the answers are on a user's profile).

The work most related to Chapter 3 is a recent study by Kim et al. [173], already discussed in §3.1.4. They formally quantify the advantage an attacker has against SA tests when he is already inside the victim's social circle. The researchers thus demonstrate that SA is ineffective against one's close friends and family or highly connected social sub-networks such as universities. However, in our work we extend the threat model to incorporate any attacker located outside the victim's social circle. Furthermore, we implement a proof-of-concept infrastructure, and use publicly available information to quantify the effectiveness of such attacks. Thus, we are able to show the true extent to which SA is susceptible to automated attacks. Previous work [103, 107, 196, 231] has proved the feasibility of positioning one's self among a target's social circle using a mix of active and passive [162] techniques ranging from social engineering (e.g., attractive fake profiles) to forgetful users accepting friendship requests from fake profiles of people they are already linked. As such, the proposed countermeasures by Kim et al. [173] for a more secure social authentication mechanism remain equally vulnerable to our attack. Finally, we also present a theoretical estimation of the attack surface based on empirical data from our experiments as well as those reported by previous studies.

Boshmaf et al. [107] explore the feasibility of socialbots infiltrating social networks, and operate a Socialbot Network in Facebook for 8 weeks. A core aspect of their operation is the creation of new accounts that follow a stealthy behavior and try to imitate human users. These actions are complimentary to our attack, as a determined attacker can use them to infiltrate the social circles of his victims and expand the attack surface by gaining access to private photos. This will result in much higher percentages of solved SA tests. Gao et al. [140] found that 97% of malicious accounts were compromised accounts of legitimate users. This

reflects the importance of social authentication as a mechanism for preventing attackers from taking over user accounts using stolen credentials. Accordingly, we explore the feasibility of an automated attack that breaks SA tests though the use of face recognition techniques. Our results validate the effectiveness of our attack even when the attacker uses only publicly available information.

[108] conducted an extensive study to measure the success rate of users when presented with CAPTCHA tests from various services. An important difference of their study was that they employed users through an underground solving service and workers from AMT. As these users are employed for such tasks, their results are higher than those that would have been achieved by average users. Nonetheless, in our user study, participants were able to solve over 99% of the medium challenges. An important observation the authors make is that the difficulty of CAPTCHAs is often very high resulting in their solution being a troublesome process for users. On the other hand, our approach is more user-friendly as users are required to identify their friends in photos.

Our user study demonstrated the ability of users to rely on secondary or contextual information to identify the depicted friends even when no faces are contained in the photo. By definition, attacks utilizing face identification software will fail in such cases. Similarly, previous image-based CAPTCHA schemes have leveraged semantic content information to defend against attacks employing image analysis software [233].

Previous work [144, 209] has explored the use of human faces for creating CAPTCHAs. [144] built upon the failure factors of face detection algorithms to propose a CAPTCHA system that uses randomly distorted human and non-human faces as a test. Users are prompted to identify all human faces without making any mistakes. The pool of faces is compiled from publicly available databases and a series of distortions are applied, such as adding stripes, covering key face characteristics, rotating the image and overlaying the actual face over a background of random noise and blending the two. Although, an attacker could get access to the public photo database and be aware of the distortion algorithms, such tests significantly raise the bar due to the random fashion that the transformations are being applied.

Overall, the ultimate goal of a CAPTCHA is to make it very hard for a machine to discern noise from signal in the input. We build upon such practices and extend them, as for us this is only the first step since we want to make it difficult for a machine to (i) match two distinct inputs (photos of a specific human) based on similarity of the detected facial characteristics and (ii) match the two different distortions of the same input based on similarity. Systems like FaceDCAPTCHA [144] help mitigate the first case as they remove parts of the signal and substitute with noise, but are not effective against the second case.

9.2 User information

9.2.1 Data mining and correlation

To understand the ways that spammers obtain target email addresses, Shue et al. in [215] post a number of email addresses on popular websites and monitor the inflow spam that these addresses receive. They also conduct a study of the current web crawlers that spammers are equipped with. The results of this work led the authors to three major conclusions; first, email addresses are discovered quickly on the Internet by spammers. Second, spamming crawlers can be tracked and, finally, most spammers use multiple harvesting techniques on a plethora

of sources including blogs, social networking sites, mailing lists etc. According to the authors, even a single exposure of an email address can result in instant and high-volume spam.

Prince et al. [203] present the results of Project Honeypot [70], which aims to reveal the primary way by which spammers collect new email addresses. Project Honey Pot is a highly distributed network consisting of millions of spam traps (email honeypots) placed in websites. As the authors state, harvesting is the basic methodology used by spammers to obtain new email addresses. They divide harvesters into two classes according to their turnaround times from the moment an email address is harvested until the first message is sent. *Hucksters*, are characterized by a slow turnaround time, while *fraudsters* send the first message almost instantly after the address is harvested.

Kreibich et al. [178] measure the orchestration of spam campaigns by hooking into botnet command-and-control (C&C) protocols. Among others, the authors make an analysis of hundreds of millions of harvest reports (lists of target email addresses that spam bots harvest) that were collected through their proxies. They observe that the most frequently harvested domains correspond to major email services such as *hotmail.com, yahoo.com, aol.com.* However, almost 10% of all harvested email addresses do not correspond to valid top level domains.

Krishnamurthy and Wills [181] describe how third-party servers can exploit the personal identifiable information (PII) leakage of social networks so as to link it with user actions inside these networks or even elsewhere on the Internet. The authors demonstrate that most users can have their PII linked with tracking cookies. They state that this is a corollary of users' ignorance about the importance of strong privacy settings in a social network.

Wondracek et al. [241] introduce a novel attack that aims to de-anonymize users of social networking sites. More precisely, they argue that an adversary who runs a website can reveal the identity of social network users who visited his website, by learning their group memberships through web browser history stealing techniques. Moreover, the authors show the effectiveness of their attack by applying it to Xing, a real world social network and they state through empirical analysis, that other larger networks such as Facebook are also vulnerable to their attack.

Balduzzi et al. [98] also demonstrated how to use social networking sites as an oracle through the search utilities provided by such sites; an attacker can search for an email address, and if a user profile was registered with that email, the profile is returned. Thus, the attacker can map an email address to a social network profile. Using a dataset of over 10 million email addresses taken from a compromised machine the police took down, they were able to identify around 876,000 of those addresses on at least one the eight social networks they investigated. An interesting finding was that almost 200,000 users had registered on at least two OSN using the same email address. Furthermore, the information extracted from the multiple profiles shows that users enter different information across profiles. For example, 2,213 users (12% of the ones registered in more than one network) pretended to be male on a network and female on a different one. A serious consequence of this attack, is the ability to uncover hidden profiles and online identities that users wish to keep secret

In [103], the authors demonstrate the feasibility of automated identity theft attacks in social networks, both inside the same network and different ones. They are able to create forged user profiles and invite the victims' contacts to form social links or open direct messages. By establishing a social link with the forged profile, the attacker has full access to the other party's protected profile information. Furthermore, direct messages, originating from the stolen and implicitly trusted identity, may contain malicious HTTP links to phishing or malware web sites. This attack is possible mostly due to users revealing a large amount of information

on their profiles that can be accessed by practically anyone. A study conducted by Gross et al [146] revealed that only 0.06% of the users hide the visibility of information such as interests and relationships, while in [179] the authors report that 99% of the Twitter users that they checked retained the default privacy settings. Therefore, a first defense measure against such attacks could be employed by social networking sites if they promoted more strict default privacy policies. Baden et al [97] argue that by using exclusive shared knowledge for identification, two friends can verify the true identity of each other in social networks. This can enable the detection of impersonation attacks in such networks, as attackers that impersonate users will not be able to answer questions. Once a user's identity has been verified, public encryption keys can be exchanged. Furthermore, by using a web of trust one can discover many keys of friends-of-friends and verify the legitimacy of user profiles that they don't know in the real world and don't share any secret knowledge.

9.2.2 Social Forensics

In [113] the authors demonstrate the acquisition of data from the RAM of a desktop PC with a goal of reconstructing the previous Facebook session, by locating some distinct strings. Garfinkel introduced the Forensic Feature Extraction and Cross-Drive Analysis techniques [141] for extracting and correlating information from large sets of images of hard drives. In the experiments conducted on 750 drives acquired on the secondary market, the author was able to recover sensitive information ranging from credit card numbers to social security numbers and email addresses.

Mutawa et al. [87] explore what data can be recovered from mobile devices regarding user activities in social networking apps. They reported that both Iphone and Android devices contain a significant amount of valuable data that could be recovered, while Blackberry devices did not contain any such traces. For example, they were able to recover user IDs, contents of exchanged messages, URLs of uploaded pictures, and timestamps of activities from a directory of the Android Facebook app saved on an external SD card.

In [91] Andriotis et al. investigated the presence of data regarding the use of Wi-Fi or Bluetooth interfaces in system log and database files of Android smartphones. Their results showed that the elapsed time between a criminal activity and the acquisition of the device was critical, as a lot of information was lost from the logs after just a few hours, due to their fixed size. However, database files were found to retain the useful information.

An interesting technique that could be applied in social forensics investigations was presented by Mao et al. [189]. They characterized the leakage of information in Twitter and, specifically, if users divulged vacation plans, tweeted under the influence of alcohol or revealed medical conditions. Building activity-classifiers (not only for parsing text) for crime-related topics can assist investigators by highlighting important activities of the suspect.

Data Collection. The work most relevant to ours, which focused on extracting data from social networks in the context of forensics analysis, was by Huber et al. [158]. They extracted data from Facebook through the use of an automated browser and a third-party application, and focused on measuring the completeness of the data their system collected. They also referred to the correlation of users across services, and how analysis of the collected data could be done through graph and timeline visualization. In [195] they also presented connected graphs depicting users that had exchanged messages or had been tagged in the same photo.

Overall, our work presents several differences. First, we provide an extensive framework

that extracts data from a large number of social networks and communication services. Second, we have implemented a user correlation component that provides analysts with a unified representation of users, as each contact is represented by their activities that span across multiple services. Third, we provide a dynamic visualization framework with viewpoints of varying granularity that allows forensic specialists to analyze the collected data and focus on different contacts, services or types of activity.

Account Correlation. Papers that detect cloned user profiles across social networks [171, 174] have techniques that we could incorporate into our account correlation component. Specifically, when the correlation occurs through the fuzzy string matching technique, which might be wrong (as opposed to the Facebook oracle and `about.me` techniques which guarantee that the result is correct), profile content and social graph similarities can be used to further ascertain that the correlation is correct.

Data Visualization. Numerous existing libraries can visualize graphs depicting the structure of social networks. Also, the visualization of online social networks is an active research area, and multiple publications [115, 153, 213] have focused on implementing visualization techniques. We are, to the best of our knowledge, the first to develop a dynamic framework that visualizes an extensive range of user activities and communications in online social networks. However, there are additional visualizations that can be added to our framework. For example, an interesting extension we would like to add to the visualization component, is to measure the social influence between the suspect and online contacts [226].

9.3 User actions: Location-based services

Joining LBS. Researchers have tried to understand the motives of users that join systems like Foursquare. Their findings suggest that there is a significant portion of users that participate for the discounts and special offers [186]. This is also supported by [201], where nearly 20% of the test-subjects reported that offers were an important reason for participating. Cramer et al. conduct a user survey [116, 117], and find that both the "gaming" aspect and venue offers are significant incentives for user participation.

Attacking LBS. One of the first to raise awareness about the implications of fake-location attacks against LBS was [152]. Even though part of this paper focuses on the same problem, our work presents several characteristics that differentiates it. While He et al. refer to certain heuristics used by Foursquare, they have not explored them in depth so as to identify their thresholds. We follow a systematic black-box testing approach that accurately identifies the thresholds for each heuristic. Furthermore, they develop a semi-automatic tool that demonstrates that fake check-ins are feasible, but due to the lack of knowledge of thresholds, cannot demonstrate the true extent of potential attacks. Our automated tool systematically explores the detection mechanisms, reveals the extent to which attacks are feasible and highlights their true impact. Finally, our adaptive attack uses information at runtime to avoid redundant check-ins and better distribute their limited number. Overall, our systematic exploration of the detection mechanisms deployed by two very popular LBSs, reveals the extent to which fake-location attacks are feasible and highlights their true impact. We consider this to be an important step towards better understanding such services.

Validating User Location. Carbunar et al. [112] present a mechanism that allows users of LBS to communicate with the service in a private manner, i.e., the service cannot link users to a specific location at a specific moment in time. Furthermore, users are still able to acquire

badges and mayorships through the use of cryptographic tools and, thus, the system retains its functionality and gaming aspect. However, their solution for validating the user's location is based on an LCD screen that displays a QR code that contains information that the user sends to the LBS to verify his position. To defend against wormhole attacks, a new QR code is created whenever the device detects a person (or object) passing by. This is done using the ambient light sensor embedded in an Android device (which is typically used for controlling the Android's screen brightness [6]). However, their proposed solution remains susceptible to wormhole attacks under certain conditions. The user can communicate the QR code to a third party before a new QR code is computed, by remaining in front of the ambient light sensor until the third party also uses the QR. As the ambient sensor will not detect any change to the light level during that time, it will not invalidate the current QR code and compute a new one. On the other hand, our implementation of an NFC server is immune to wormhole attacks. In their follow-up paper [111] they propose two methods for extending their initial implementation for detecting wormhole attacks. In the first solution called NES, the user is presented with a random challenge which the user has to compute a hash of and forward to the system along with a nonce the user receives from the LBS. The system will use these to create the check-in information which is sent to the user, who forwards it to the service that will use it to verify the user's check-in. While the first approach is somewhat similar to our NFC countermeasure, it presents disadvantages. First, the LBS is required to keep state regarding each check-in attempt by a user, as it has to store the nonce values sent to each user (apart from the keys of all venues and users as we do). It also imposes higher computational overhead on the service per check-in and requires the venue to be able to communicate with the service, while our approach works even without Internet connectivity. The second countermeasure, WES, relies on a WiFi router present at the location that periodically changes its SSID to a value that the user will forward to the LBS. However, there is no description of the process by which the LBS will predict the SSID values of each venue's router at a given time, or the extra state the service will have to store. Finally, neither solution has actually been implemented.

The Echo protocol [212] is for securely verifying location claims, using a time-of-flight approach. This protocol does not require a setup or registration step, which excludes cryptographic operations, and can be deployed for various applications. However, this solution requires devices that can emit both radio and ultrasound frequencies, while we take advantage of the NFC capabilities present in many modern smartphones. Furthermore, for large or non-circular areas, multiple nodes are required. On the other hand, we target a specific deployment scenario, for LBS to ensure user presence at a specific venue during the check-in process. Their solution relies on calculating the time needed for a successful communication by a legitimate user, and demonstrate that a wormhole attack results in much higher times for the attacker. However, they conduct their experiments with users connecting over WiFi and not mobile networks. As mobile networks present much higher delays, distinguishing between legitimate users and attackers through timing analysis may not be applicable.

Narayanan et al. [198] present secure protocols for private proximity testing, where two friends can be notified when within a specific distance of each other, while their location remains secret. Adams et al. [114] present a secure location sharing system that works over untrusted infrastructure, to solve the problem of location information being wrongfully shared with unintended audiences. Extensive work has been published regarding distance bounding protocols (e.g., [207]), for verifying a user's position. In [148], the RFID distance bounding protocol assumes that the prover does not collude with a third party closer to the verifier and is, thus, vulnerable to wormhole attacks.

GPS Spoofing. Techniques to test if a user is present within a small area, or absent from a large area by simply listening on the broadcast GSM channels have been presented in [121]. Xu et al. [243, 244] propose a feeling-based model for location privacy protection in LBSs, which allows users to express the level of privacy they require based on their location. Although we focus on spoofing attacks targeting the application layer, there is research for spoofing the location at a low-level. In [229] the authors explore requirements for successful GPS spoofing attacks on individuals and groups of victims equipped with civilian receivers, and perform a set of experiments and find the minimal precision of the attacker's spoofing signals required for covert satellite-lock takeover.

User location. Enck et al. [128] conducted a study to understand the underlying security issues of android applications. Among other findings, they present issues regarding applications accessing a user's location. They found that 45.9% of the applications attempt to access the user's location, while only 27.6% have the permission to do so. In several instances, the location was disclosed to advertisement servers. In [124] the authors propose two security mechanisms for Android to prohibit applications from using other applications with higher privileges to perform actions they lack the credentials for, such as acquiring the user's location. Mixzone-based privacy preserved mechanisms are discussed in [104], while exposing the network location of devices for launching DoS attacks is presented in [205]. The danger of inferring sensitive user information based on the knowledge of his location mandates the incorporation of an adversary's ability to predict location in a formal definition of location privacy [191, 214].

Chapter 10

Conclusion

10.1 Summary

In this dissertation we explored the security properties of online social networks. Due to their proprietary nature, we assumed the role of the adversary, and explored the defense mechanisms deployed by OSNs to protect their assets.

User accounts. We researched the problem of adversaries attempting to gain access to user profiles, using stolen credentials or by hijacking sessions. As compromised accounts in social networks have become valuable commodities sold in the underground markets, attackers have employed various attacks for tricking users into divulging their credentials. In an effort to prevent unauthorized account access, Facebook deployed its social authentication mechanism. We documented the security weaknesses of using social authentication as part of a two-factor authentication scheme and empirically calculated the probability of an attacker obtaining the information necessary to solve social authentication tests when relying on publicly accessible data as well as following a more active approach to gather restricted information.. We found that if an attacker manages to acquire the first factor (password), he can access, on average, 42% of the data used to generate the second factor, thus, gaining the ability to identify randomly selected photos of the victim's friends. Given that information, we managed to solve 22% of the real Facebook SA tests presented to us during our experiments and gain a significant advantage to an additional 56% of the tests with answers for more than half of pages of each test. We have designed an automated social authentication breaking system, to demonstrate the feasibility of carrying out large-scale attacks against social authentication with minimal effort on behalf of an attacker. Our experimental evaluation has shown that widely available face recognition software and services can be effectively utilized to break social authentication tests with high accuracy.

We then explored the possibility of designing a more secure social authentication mechanism, and proposed a novel approach that retains its usability while being robust against attacks that utilize image analysis techniques. The key concept is to filter out faces that can be identified by face recognition software, and craft the challenge photos in a manner that obfuscates any areas with sensitive information from the initial photos. We conducted a measurement to explore the ability of users to identify their friends in photos taken under realistic conditions. The faces in the photos were processed by face recognition software and assigned to 3 categories, depending on the visibility of the faces. Our results demonstrated that users are very effective at identifying their friends even when their face is not clearly visible or present at all. Based

on our study results and a series of observations we set a series of guidelines for implementing a secure yet usable SA mechanism. We also implemented a working prototype and discussed the benefits of an OSN deploying it as a user-gnostic CAPTCHA service. We discussed its security properties against CAPTCHA-breaking attacks, and the major advantage of being resistant to outsourcing attacks as each challenge is crafted for a specific user.

User information. We explored techniques that can be used by adversaries for collecting user information from social network profiles, and other digital services. The abundance of publicly available personal information is exploited by adversaries for deploying personalized attacks. We demonstrated how information, that is publicly available in social networking sites, can be used for harvesting email addresses and deploying personalized phishing campaigns. We argue that an inherent challenge of a social network is the visibility of its members. The mere participation of users renders them as targets for personalized attacks. We presented two different approaches to harvesting email addresses. *Blind harvesting* uses names collected from social networking sites and aims to collect as many email addresses as possible. Using this technique we were able to harvest millions of email addresses in an efficient fashion. *Targeted harvesting* aims to harvest email addresses that can be mapped to a name and publicly available information and, thus, greatly enhance the efficiency of a spam campaign. We presented three such techniques. The first technique blindly harvested email addresses and uses Facebook to map them to a user name, with a success rate of 11.5%. By using information available in the Twitter network we were able to narrow the search space and accurately map 43.4% of the user profiles. Next, we used names collected from Facebook fan pages to harvest Google Buzz accounts, 40.5% of whom provided a direct mapping to a Gmail account.

We also designed and implemented a prototype tool that can be employed by users to investigate whether they have fallen victims to such an identity theft attack. The core idea behind our tool is to pinpoint any information contained in a user's profile that is uniquely identifying. We evaluated our assumption regarding the effectiveness of such a tool and found that user profiles usually reveal information that is rare and, when combined with a name, can uniquely identify a profile and thereby any existing cloned profiles. We presented the findings from a study regarding the type and amount of information exposed by social network users and concluded that the same user-identifying information which allows an attacker to clone a profile also assists us in identifying the clone. This is demonstrated by a test deployment of our tool, in which we searched LinkedIn for duplicate profiles, and found that for 7% of the user profiles checked, we discovered a duplicate profile in the same social network..

Finally, we explored how we could associate disjoint sources of information for creating more complete user profiles. In the context of our social forensics toolset, we developed a correlation module that attempts to map user profiles from separate social networks and digital services to the same user. Several techniques are incorporated for correlating the accounts, ranging from employing service-specific search functionality, to fuzzy matching approaches.

User actions. Once the adversary has gained access to a service through a compromised account, or crafted fake profiles, they can continue with malicious activities by targeting users, or even the actual service. Their actions within the service can have a negative impact on the service's functionality or business model. Due to the growing popularity of location-based functionality in social networks, and the critical impact of fake-location attacks on such services we decided to explore the security aspects of location-based services. Using a black-box testing approach we revealed the server-side heuristics employed by Foursquare and Facebook Places for detecting fake-location attacks and discovered their respective thresholds. Our technique,

without prior knowledge, revealed a bug that allows one to bypass speed constraints set by Facebook Places. We also presented an algorithm that leverages all discovered thresholds of Foursquare for maintaining mayorship in venues. Finally, we implemented an NFC server that can eliminate system-wide threats, analyzed its security properties and evaluated its performance. Overall, we demonstrated the extent to which LBS are vulnerable and the true impact of fake-location attacks, so as to draw the attention of the research community. Second, to exposed the inefficiency of anomaly detection mechanisms and utilized our findings for designing and implementing an effective countermeasure.

10.2 Future Work

We have discussed limitations of our current implementations and outlined possible solutions that can be explored as part of future work in previous chapters. In this section we summarize the most important future steps for eliminating the limitations, and also outline directions that can be taken as complementary steps for extending our current work.

Social Authentication. To empirically explore the robustness of our photo selection process against automated face identification, we will have to conduct an extensive study using a wide range of face recognition software. While individual photos may be robust against identification, combining the 3 test photos may provide enough information to identify the depicted user. Furthermore, we wish to conduct an extensive user study to explore the usability of our photo presentation technique. While our technique renders photos robust against photo recognition attacks, the crafted photos must remain recognizable by humans.

Social Forensics. The use of mobile devices to access the Internet has seen a dramatic rise, with 21% of website access in 2013 originating from mobile devices [62]. While social networking activities synchronize and can be extracted from any device using the user's credentials, mobile devices contain a plethora of information that is unique to each device, such as location data, and communication activities. As such, a complete forensics toolset requires modules for extracting such information from mobile devices.

Location-based Services. Despite their obvious advantages in terms of innovative functionality, location-aware services also threaten the locational privacy of users, i.e., "the ability of an individual to move in public space with the expectation that under normal circumstances their location will not be systematically and secretly recorded for later use" [15]. Services are able to track users and gain detailed knowledge of their whereabouts, which has raised many privacy concerns within the research community [69, 228, 237]. To illustrate the severity of the issue, we present examples of sensitive information that can be inferred by monitoring the locations visited by a user: health issues (user visits a doctor specializing in AIDS or cancer), religious beliefs (user visits a church), political inclinations (user attends a political party's rallies), etc. Users also expose themselves to physical threats such as stalking. As users grow accustomed to revealing this sensitive information to an ever-growing set of services, it becomes more likely that they may also reveal it to untrusted third parties.

In the future we plan to explore the following direction regarding the functionality of location-based services and social networks;designing mechanisms that will maintain the usability of the services while ensuring users' locational privacy. Specifically, we plan to design a protocol that will allow users to communicate their location to a service in an anonymized fashion.

Bibliography

[1] 10 Minute Mail. http://10minutemail.com/.

[2] 1.5 million facebook accounts offered for sale. http://www.zdnet.com/blog/security/1-5-million-facebook-accounts-offered-for-sale-faq/6304.

[3] 51 Amazing Facebook Facts. http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/.

[4] American Express discounts in FourSquare. https://sync.americanexpress.com/foursquare/.

[5] Android issues: Enable real nfc p2p communication. http://code.google.com/p/android/issues/detail?id=28014.

[6] Android sensors overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html.

[7] BBC News - Facebook prison officer is sacked. http://news.bbc.co.uk/2/hi/uk_news/england/leicestershire/7959063.stm.

[8] BBC News - "Ill" worker fired over Facebook. http://news.bbc.co.uk/2/hi/8018329.stm.

[9] Bussiness Insider - Facebook Users Are Uploading 350 Million New Photos Each Day. http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9.

[10] A case study of eurograbber. https://www.checkpoint.com/products/downloads/whitepapers/Eurograbber_White_Paper.pdf.

[11] Clean Tweets. http://blvdstatus.com/clean-tweets.html.

[12] cURL. http://curl.haxx.se/.

[13] D3.js - Data-Driven Documents. http://d3js.org.

[14] Department of Law, State of New Jersey. Melanie McGuire found guilty of murder in 2004. http://www.nj.gov/oag/newsreleases07/pr20070423a.html.

[15] Electronic frontier foundation on locational privacy, and how to avoid losing it forever. https://www.eff.org/wp/locational-privacy.

[16] F-Secure: Success rate of Facebook Spam. http://www.f-secure.com/weblog/archives/00002015.html.

[17] Facebook Acquires Face.com. http://mashable.com/2012/06/18/facebook-acquires-face-com/.

[18] Facebook developers - bugs. https://developers.facebook.com/bugs/244713388933143?browse=search_4f12b26febf840e00208758.

[19] Facebook developers: Email permissions. https://developers.facebook.com/docs/reference/login/email-permissions/.

[20] Facebook entry gets office worker fired. http://news.cnet.com/8301-17852_3-10172931-71.html.

[21] Facebook Hacker A Dangerous Tool. http://www.malwarecity.com/blog/facebook-hacker-a-dangerous-tool-889.html.

[22] Facebook hit by phishing attacks for a second day. http://edition.cnn.com/2009/TECH/04/30/facebook.phishing.attacks/index.html.

[23] Facebook message points to a phishing site. http://www.labnol.org/internet/facebook-phishing/9444/.

[24] Facebook Password Reset Scam Threatens Computers Worldwide. http://siblog.mcafee.com/consumer/consumer-threat-alerts/facebook-password-reset-scam-threatens-computers-worldwide/.

[25] Facebook social plugins. https://developers.facebook.com/docs/plugins/.

[26] Facebook suspends app that permitted peephole. http://news.cnet.com/8301-10784_3-9977762-7.html.

[27] Firesheep, A Firefox extension that demonstrates HTTP session hijacking attacks. http://codebutler.github.com/firesheep/.

[28] Forensic focus: Dropbox forensics. www.forensicfocus.com/Content/pid=429/page=2/#database.

[29] Fortune magazine. http://money.cnn.com/magazines/fortune/.

[30] Foursquare adds nfc support to its android app. http://techcrunch.com/2012/02/10/foursquare-adds-nfc-support-to-its-android-app/.

[31] Foursquare API Endpoints. https://developer.foursquare.com/docs/.

[32] Foursquare blog - the follow-up to our "mayorships from your couch" post. http://blog.foursquare.com/2010/04/08/505862083/.

[33] Foursquare ceo: 'not just check-ins and badges'. http://money.cnn.com/2012/02/29/technology/foursquare_ceo/index.htm.

[34] Foursquare wrapper. https://github.com/mLewisLogic/foursquare.

[35] Friendster. http://www.friendster.com/.

[36] Genealogy data: Frequently occurring surnames. http://www.census.gov/genealogy/www/data/2000surnames/index.html.

[37] Geocities. http://geocities.yahoo.com.

[38] Google+ API. https://developers.google.com/+/api/.

[39] Google buzz. http://buzz.google.com/.

[40] Google Developers - Google Maps JavaScript API v3. https://developers.google.com/maps/documentation/javascript/.

[41] Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/.

[42] Google safebrowsing API. http://code.google.com/apis/safebrowsing/.

[43] The guardian - china suspected of facebook attack on natos supreme allied commander. http://www.guardian.co.uk/world/2012/mar/11/china-spies-facebook-attack-nato.

[44] The guardian: NSA prism program taps in to user data of apple, google and others. http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data.

[45] Gympact. http://www.gym-pact.com/.

[46] How does foursquare handle cheating? http://support.foursquare.com/entries/188307-how-does-foursquare-handle-cheating-fraud-detection-make-sure-check-ins-are-real.

[47] IACP center for social media, 2012 survey results. http://www.iacpsocialmedia.org/Resources/Publications/2012SurveyResults.aspx.

[48] Immersion. https://immersion.media.mit.edu/.

[49] Introducing google+ sign-in. http://googleplusplatform.blogspot.in/2013/02/google-plus-sign-in.html.

[50] Jakobsson, M. Modeling and preventing phishing attacks. In Phishing Panel at Financial Cryptography. Feb. 2005.

[51] joewein.de LLC Fighting spam and scams on the Internet. http://www.joewein.net/.

[52] jQuery. http://jquery.com.

[53] Latest Facebook Scam: Phishers Hit Up "Friends" for Cash. http://techcrunch.com/2009/01/20/latest-facebook-scam-phishers-hit-up-friends-for-cash/.

[54] A lexical database for English. http://wordnet.princeton.edu/.

[55] LinkedIn. http://www.linkedin.com/.

[56] List of searched keywords. http://animalnewyork.com/2012/the-department-of-homeland-security-is-searching-your-facebook-and-twitter-for-these-words/.

[57] Mailing list archives. http://marc.info/.

[58] Malicious facebook app propagates via users. http://securitylabs.websense.com/content/Blogs/3563.aspx.

[59] Malicious facebook apps can steal your info. http://www.gadgetell.com/tech/comment/malicious-facebook-apps-can-steal-your-info/.

[60] Mayor of the north pole. http://krazydad.com/blog/2010/02/15/mayor-of-the-north-pole/.

[61] Messenger: One place for your social updates. http://explore.live.com/windows-live-messenger-updates-social-one-place.

[62] Mobile marketing statistics 2013. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[63] More employers use social networks to check out applicants. http://bits.blogs.nytimes.com/2009/08/20/more-employers-use-social-networks- to-check-out-applicants/.

[64] MySpace XSS QuickTime Worm. http://securitylabs.websense.com/content/Alerts/1319.aspx.

[65] Oauth 2.0. http://tools.ietf.org/html/draft-ietf-oauth-v2-23.

[66] On foursquare, cheating, and claiming mayorships from your couch. http://blog.foursquare.com/2010/04/07/503822143/.

[67] PhantomJS: Headless WebKit with JavaScript API. http://phantomjs.org/.

[68] Please Rob Me. http://pleaserobme.com/.

[69] Privacy rights clearinghouse/ucan - a chronology of data breaches. http://www.privacyrights.org/ar/ChronDataBreaches.htm.

[70] Project honey pot: Help stop spammers before they ever get your address! http://www.projecthoneypot.org/.

[71] Rogue Toolbars Serve Up Facebook Phishing Pages. http://sunbeltblog.blogspot.com/2010/03/rogue-toolbars-serve-up-facebook.html.

[72] Rsa performance of sun fire t2000. http://blogs.sun.com/chichang1/entry/rsa_performance_of_sun_fire.January2012.

[73] Russian underground. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf.

[74] Social, digital video drive further growth in time spent online. http://www.emarketer.com/Article/Social-Digital-Video-Drive-Further-Growth-Time-Spent-Online/1009872.

[75] Spam detector. http://www.cs.ucsb.edu/~gianluca/spamdetector.html.

[76] Symantec - Industrial Espionage: Targeted Attacks and Advanced Persistent Threats (APTs). http://www.symantec.com/threatreport/topic.jsp?aid=industrial_espionage&id=malicious_code_trends.

[77] Techcrunch - latest facebook scam: Phishers hit up friends for cash. http://techcrunch.com/2009/01/20/latest-facebook-scam-phishers-hit-up-friends-for-cash/.

[78] Telegraph - twitter claims 15m active users in the uk. http://www.telegraph.co.uk/technology/twitter/10291360/Twitter-claims-15m-active-users-in-the-UK.html.

[79] TinyURL.com - Making long URLs usable! http://tinyurl.com/.

[80] tweepy. https://github.com/tweepy/tweepy.

[81] Twitter hit with phishing attacks. http://www.computerworld.com/s/article/9133405/Twitter_hit_with_phishing_attacks.

[82] URIBL - realtime URI blacklist. http://uribl.com/.

[83] W3C public mailing list archives. http://lists.w3.org/.

[84] A. Acquisti and R. Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In *Proceedings of 6th Workshop on Privacy Enhancing Technologies*, 2006.

[85] A. Acquisti, R. Gross, and F. Stutzman. Faces of Facebook: How the largest real ID database in the world came to be. BlackHat USA, 2011, `http://www.heinz.cmu.edu/~acquisti/face-recognition-study-FAQ/acquisti-faces-BLACKHAT-draft.pdf`.

[86] F. Adu-Oppong, C. K. Gardiner, A. Kapadia, and P. P. Tsang. Social circles: Tackling privacy in social networks. In *Symposium on Usable Privacy and Security (SOUPS)*, 2008.

[87] N. Al Mutawa, I. Baggili, and A. Marrington. Forensic analysis of social networking applications on mobile devices. *Digital Investigation*, 9:S24–S33, 2012.

[88] R. Albert, H. Jeong, and A. L. Barabasi. The diameter of the world wide web. *Nature*, 401:130–131, 1999.

[89] Amazon. Amazon Mechanical Turk. `https://www.mturk.com/mturk/`.

[90] J. Anderson, C. Diaz, J. Bonneau, and F. Stajano. Privacy-enabling social networking over untrusted networks. In *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, pages 1–6. ACM, August 2009.

[91] P. Andriotis, G. Oikonomou, and T. Tryfonas. Forensic analysis of wireless networking evidence of android smartphones. In *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*, pages 109–114. IEEE, 2012.

[92] J. Arkko, A. Keranen, and M. Sethi. Practical considerations and implementation experiences in securing smart object networks. `http://tools.ietf.org/html/draft-aks-crypto-sensors-00`, 2012.

[93] E. Athanasopoulos and S. Antonatos. Enhanced CAPTCHAs: Using animation to tell humans and computers apart. In *Proceedings of the 10th IFIP Open Conference on Communications and Multimedia Security*. Springer, 2006.

[94] E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. G. Anagnostakis, and E. P. Markatos. Antisocial networks: Turning a social network into a botnet. In *Proceedings of the 11th international conference on Information Security*, ISC '08, pages 146–160. Springer-Verlag, 2008.

[95] Backstrom, Lars, Dwork, Cynthia, and Kleinberg, Jon. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007.

[96] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 135–146. ACM, 2009.

[97] R. Baden, N. Spring, and B. Bhattacharjee. Identifying close friends on the internet. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[98] M. Balduzzi, C. Platzer, T. Holz, E. Kirda, D. Balzarotti, and C. Kruegel. Abusing social networks for automated user profiling. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*. Springer, 2010.

[99] J. Becker and H. Chen. Measuring privacy risk in online social networks.

[100] J. Becker and H. Chen. Measuring privacy risk in online social networks. In *Proceedings of W2SP 2009: Web 2.0 Security and Privacy*, May 2009.

[101] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. Detecting spammers on twitter. In *Proceedings of the 7th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.

[102] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334. IEEE Computer Society, 2007.

[103] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 551–560. ACM, 2009.

[104] L. Bindschaedler, M. Jadliwala, I. Bilogrevic, I. Aad, P. Ginzboorg, V. Niemi, and J.-P. Hubaux. Track Me If You Can: On the Effectiveness of Context-based Identifier Changes in Deployed Mobile Networks. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS 2012)*.

[105] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. The case for ubiquitous transport-level encryption. In *Proceedings of the 19th Conference on USENIX Security Symposium*, 2010.

[106] J. Bonneau, J. Anderson, and G. Danezis. Prying data out of a social network. *Social Network Analysis and Mining, International Conference on Advances in*, 0:249–254, 2009.

[107] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: when bots socialize for fame and money. In *Proceedings of the Annual Computer Security Applications Conference*. ACM, 2011.

[108] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving CAPTCHAs? A large scale evaluation. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. IEEE, 2010.

[109] E. Bursztein, M. Martin, and J. C. Mitchell. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM, 2011.

[110] C. Gerlitz and A. Helmon. Hit, link, like and share. Organizing the social and the fabric of the web in a like economy. Paper presented at the DMI mini-conference, volume 24, page 25, 2011.

[111] B. Carbunar and R. Potharaju. You unlocked the mt. everest badge on foursquare! countering location fraud in geosocial networks. In *IEEE 8th International Conference on Mobile Adhoc and Sensor Systems*, MASS. IEEE, 2012.

[112] B. Carbunar, R. Sion, R. Potharaju, and M. Ehsan. The shy mayor: Private badges in geosocial networks. In *ACNS*, volume 7341 of *Lecture Notes in Computer Science*. Springer, June 2012.

[113] H.-C. Chu, D.-J. Deng, and J. H. Park. Live data mining concerning social networking forensics based on a facebook session through aggregation of social data. *Selected Areas in Communications, IEEE Journal on*, 29(7):1368–1376, 2011.

[114] M. Conti, J. Vaidya, and A. Schaad, editors. *18th ACM Symposium on Access Control Models and Technologies, SACMAT '13, Amsterdam, The Netherlands, June 12-14, 2013'*. ACM, 2013.

[115] C. Correa and K.-L. Ma. *Visualizing Social Networks*, chapter Chapter 11: pp. 307-326. Springer, 2011.

[116] H. Cramer. Gamification and location-sharing : emerging social conflicts. *Proceedings of ACM CHI Workshop on Gamification*, 2011.

[117] H. Cramer, M. Rost, and L. E. Holmquist. Performing a check-in: emerging practices, norms and 'conflicts' in location-sharing using foursquare. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11. ACM, 2011.

[118] Dan Bornstein. Dalvik VM Internals. Presented at the Google I/O Developer Conference, 2008.

[119] G. Danezis and B. Wittneben. The economics of mass surveillance - and the questionable value of anonymous communications. In *In Proceedings of the 5th Workshop on The Economics of Information Security (WEIS 2006)*, 2006.

[120] M. Dantone, L. Bossard, T. Quack, and L. V. Gool. Augmented faces. In *Proceedings of the 13th IEEE International Workshop on Mobile Vision*. IEEE, 2011.

[121] Denis Foo Kune, John Koelndorfer, Nicholas Hopper, and Yongdae Kim. Location leaks on the GSM air interface. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium NDSS*, 2012.

[122] R. Dey, Z. Jelveh, and K. Ross. Facebook users have become much more private: A large-scale study. In *Proceedings of the 4th IEEE International Workshop on Security and Social Networking*. IEEE, 2012.

[123] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006.

[124] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight provenance for smart phone operating systems. In *20th USENIX Security Symposium*, Aug. 2011.

[125] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[126] J. R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01', 2002.

[127] R. Dunbar. *Grooming, Gossip, and the Evolution of Language*. Harvard University Press, 1998.

[128] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[129] Facebook. Facebook Query Language (FQL) Reference. `https://developers.facebook.com/docs/reference/fql/`.

[130] Facebook. fbconsole. `https://github.com/facebook/fbconsole`.

[131] Facebook. Graph API. `https://developers.facebook.com/docs/reference/api/`.

[132] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 251–262. ACM, 1999.

[133] A. Felt. Defacing facebook: A security case study. `http://www.cs.virginia.edu/felt/fbook/facebook-xss.pdf` accessed Aug 09, 2007.

[134] A. Felt and D. Evans. Workshop on Web 2.0 Security and Privacy. Oakland, CA. 22 May 2008. Privacy Protection for Social Networking Platforms.

[135] I. O. For Standardization. Information technology – automatic identification and data capture techniques – qr code 2005 bar code symbology specification. (18004:2006), 2006.

[136] Forbes. Solving a teen murder by following a trail of digital evidence. `http://www.forbes.com/sites/kashmirhill/2011/11/03/solving-a-teen-murder-by-following-a-trail-of-digital-evidence/`.

[137] J. Freudiger, R. Neu, and J.-P. Hubaux. Private Sharing of User Location over Online Social Networks. In *HotPETs 2010*.

[138] K. B. Frikken and P. Golle. Private social network analysis: how to assemble pieces of a graph privately. In *WPES '06: Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 89–98. ACM, October 2006.

[139] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th annual conference on Internet measurement (IMC)*, IMC '10, pages 35–47, New York, NY, USA, November 2010. ACM.

[140] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th Annual Conference on Internet Measurement*. ACM, 2010.

[141] S. L. Garfinkel. Forensic feature extraction and cross-drive analysis. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, 3:71–81, Sept. 2006.

[142] S. Gauglitz, T. Höllerer, and M. Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *Int. J. Computer Vision*, 94(3):335–360, 2011.

[143] P. Golle. Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08'. ACM, 2008.

[144] G. Goswami, B. M. Powell, M. Vatsa, R. Singh, and A. Noore. FaceDCAPTCHA: Face detection based color image CAPTCHA. In *Future Generation Computer Systems (September 2012)*.

[145] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *CCS '10: Proceedings of the 17th ACM conference on Computer and Communications Security*, pages 27–37. ACM, October 2010.

[146] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05, pages 71–80. ACM, 2005.

[147] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, WPES '05', pages 71–80, New York, NY, USA, 2005. ACM.

[148] G. P. Hancke and M. G. Kuhn. An rfid distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. IEEE SecureComm*, 2005.

[149] S. Hanna, L. Huang, E. Wu, S. Li, and C. Chen. Juxtapp: A Scalable System for Detecting Code Reuse Among Android Applications. In *Proceedings of the 9th ...*, 2012.

[150] E. Hayashi, N. Christin, R. Dhamija, and A. Perrig. Use your illusion: Secure authentication usable anywhere. In *Proceedings of the Fourth Symposium on Usable Privacy and Security (SOUPS'08)*, pages 35–43, Pittsburgh, PA, July 2008.

[151] J. He, W. Chu, and Z. Liu. Inferring Privacy Information from Social Networks. In *Intelligence and Security Informatics*, volume 3975 of *Lecture Notes in Computer Science*, chapter 14, pages 154–165. Springer-Verlag, 2006.

[152] W. He, X. Liu, and M. Ren. Location cheating: A security challenge to location-based social network services. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, 2011.

[153] J. Heer and D. Boyd. Vizster: Visualizing online social networks. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, INFOVIS '05', pages 5–, Washington, DC, USA, 2005. IEEE Computer Society.

[154] C. Herley. The plight of the targeted attacker in a world of scale. In *Proceedings of the Ninth Workshop on the Economics of Information Security*, 2010.

[155] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. In *INFOCOM*, 2003.

[156] S.-Y. Huang, Y.-K. Lee, G. B. Bell, and Z.-H. Ou. A projection-based segmentation algorithm for breaking msn and yahoo captchas. In *Proceedings of the 2008 International Conference of Signal and Image Engineering (ICSIE'08), London, UK*, Jul 2008.

[157] M. Huber, S. Kowalski, M. Nohlberg, and S. Tjoa. Towards automating social engineering using social networking sites. *Computational Science and Engineering, IEEE International Conference on*, 3:117–124, 2009.

[158] M. Huber, M. Mulazzani, M. Leithner, S. Schrittwieser, G. Wondracek, and E. Weippl. Social snapshots: digital forensics for online social networks. In *ACSAC*, pages 113–122, 2011.

[159] M. Huber, M. Mulazzani, and E. Weippl. Who on earth is mr. cypher? automated friend injection attacks on social networking sites. In *Proceedings of the IFIP International Information Security Conference 2010: Security & Privacy — Silver Linings in the Cloud*, 2010.

[160] M. Huber, M. Mulazzani, E. Weippl, G. Kitzler, and S. Goluch. Exploiting social networking sites for spam. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 693–695. ACM, 2010.

[161] M. Huber, M. Mulazzani, E. Weippl, G. Kitzler, and S. Goluch. Friend-in-the-middle attacks: Exploiting social networking sites for spam. *IEEE Internet Computing: Special Issue on Security and Privacy in Social Networks*, 5 2011.

[162] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu. Reverse social engineering attacks in online social networks. In *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011.

[163] D. Irani, S. Webb, C. Pu, and K. Li. Study of trend-stuffing on twitter through text classification. In *Proceedings of the 7th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.

[164] C. Jackson and D. Boneh. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06. ACM, 2006.

[165] D. Jacoby. Facebook Security Phishing Attack In The Wild. Retrieved on January 2012 from `http://www.securelist.com/en/blog/208193325/Facebook_Security_Phishing_Attack_In_The_Wild`.

[166] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM*, 50:94–100, October 2007.

[167] M. Jakobsson and J. Ratkiewicz. Designing ethical phishing experiments: A study of (ROT13) rOnl query features. In *Proceedings of the 15th International Conference on World Wide Web*. ACM, 2006.

[168] M. Jakobsson and S. Stamm. Invasive browser sniffing and countermeasures. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06. ACM, 2006.

[169] K. Jang, S. Han, S. Han, S. Moon, and K. Park. Sslshader: cheap ssl acceleration with commodity processors. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11.

[170] E. Jeremy, J. R. Douceur, J. Howell, and J. Sault. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*. ACM, 2007.

[171] L. Jin, H. Takabi, and J. B. Joshi. Towards active detection of identity clone attacks on online social networks. In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11', pages 27–38, New York, NY, USA, 2011. ACM.

[172] R. Kemmerer. Keynote address: "how to steal a botnet and what can happen when you do". In *Detection of Intrusions and Malware, and Vulnerability Assessment, 6th International Conference, DIMVA 2009*.

[173] H. Kim, J. Tang, and R. Anderson. Social authentication: harder than it looks. In *Proceedings of the 2012 Cryptography and Data Security conference*. Springer, 2012.

[174] G. Kontaxis, I. Polakis, S. Ioannidis, and E. P. Markatos. Detecting social network profile cloning. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, pages 295–300. IEEE, 2011.

[175] G. Kontaxis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos. Privacy-preserving social plugins. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12. USENIX Association, 2012.

[176] A. Korolova, R. Motwani, S. U. Nabar, and Y. Xu. Link privacy in social networks. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 289–298. ACM, 2008.

[177] M. E. Kounavis, X. Kang, K. Grewal, M. Eszenyi, S. Gueron, and D. Durham. Encrypting the internet. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10'. ACM.

[178] C. Kreibich, C. Kanich, K. L. Br, O. Enright, and S. Savage. On the spam campaign trail. In *In First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET08)*, 2008.

[179] B. Krishnamurthy, P. Gill, and M. Arlitt. A few chirps about twitter. In *WOSN '08: Proceedings of the first workshop on Online social networks*, pages 19–24, New York, NY, USA, 2008. ACM.

[180] B. Krishnamurthy and C. E. Wills. Characterizing privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, WOSN '08, pages 37–42. ACM, 2008.

[181] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. In *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, pages 7–12, New York, NY, USA, 2009. ACM.

[182] B. Krishnamurthy and C. E. Wills. Leakage in Mobile Online Social Networks. In *Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010)*, June 2010.

[183] B. Krishnamurthy and C. E. Wills. Privacy Leakage in Mobile Online Social Networks. In *Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010)*, June 2010.

[184] L. Li, D. Alderson, J. Doyle, and W. Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2:431–523, 2005.

[185] J. Lindamood, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham. Inferring private information using social network data. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 1145–1146. ACM, 2009.

[186] J. Lindqvist, J. Cranshaw, J. Wiese, J. Hong, and J. Zimmerman. I'm the mayor of my house: examining why people use foursquare - a social-driven location sharing application. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11. ACM.

[187] W. Luo and U. Hengartner. Proving your location without giving up your privacy. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10'. ACM, 2010.

[188] M. Madejski, M. Johnson, and S. M. Bellovin. A study of privacy settings errors in an online social network. In *Proceedings of the 4th IEEE International Workshop on Security and Social Networking*. IEEE, 2012.

[189] H. Mao, X. Shuai, and A. Kapadia. Loose tweets: an analysis of privacy leaks on twitter. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11', pages 1–12, New York, NY, USA, 2011. ACM.

[190] Mashable. Vancouver fans riot as canucks lose stanley cup. `http://mashable.com/2011/06/15/vancouver-hockey-riot/`.

[191] K. Minami and N. Borisov. Protecting location privacy against inference attacks. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, WPES '10, 2010.

[192] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[193] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks.

[194] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas: understanding captcha-solving services in an economic context. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10. USENIX Association, 2010.

[195] M. Mulazzani, M. Huber, and E. Weippl. Social network forensics: Tapping the data pool of social networks. In *Eighth Annual IFIP WG 11.9 International Conference on Digital Forensics*, 1 2012.

[196] F. Nagle and L. Singh. Can friends be trusted? Exploring privacy in online social networks. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*. IEEE, 2009.

[197] A. Narayanan and V. Shmatikov. De-anonymizing social networks. *Security and Privacy, IEEE Symposium on*, 0:173–187, 2009.

[198] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[199] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[200] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil. An empirical study of geographic user activity patterns in foursquare. In *ICWSM*, 2011.

[201] S. Patil, G. Norcie, A. Kapadia, and A. J. Lee. Reasons, rewards, regrets: privacy considerations in location sharing as an interactive practice. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12. ACM, 2012.

[202] I. Polakis, M. Lancini, G. Kontaxis, F. Maggi, S. Ioannidis, A. Keromytis, and S. Zanero. All your face are belong to us: Breaking facebook's social authentication. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12'. ACM, 2012.

[203] M. B. Prince, B. M. Dahl, L. Holloway, A. M. Keller, and E. Langheinrich. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In *CEAS*, 2005.

[204] K. P. Puttaswamy, A. Sala, and B. Y. Zhao. Starclique: guaranteeing user privacy in social networks against intersection attacks. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 157–168. ACM, 2009.

[205] Z. Qian, Z. Wang, Q. Xu, Z. M. Mao, M. Zhang, and Y.-M. Wang. You Can Run, but You Can't Hide: Exposing Network Location for Targeted DoS Attacks in Cellular Networks. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS 2012)*, 2012.

[206] A. Rabkin. Personal knowledge questions for fallback authentication: Security questions in the era of Facebook. In *Proceedings of the 4th Symposium on Usable Privacy and Security*. ACM, 2008.

[207] K. B. Rasmussen and S. Čapkun. Realization of RF distance bounding. In *Proceedings of the 19th USENIX conference on Security*, 2010.

[208] RSA. Apt summit findings. `http://www.rsa.com/innovation/docs/APT_findings.pdf`.

[209] Y. Rui and Z. Liu. Artifacial: Automated reverse turing test using facial features. In *In Multimedia*, pages 295–298. ACM Press, 2003.

[210] S. Saroiu and A. Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, HotMobile '09'. ACM, 2009.

[211] S. Saroiu and A. Wolman. I am a sensor, and i approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10'. ACM, 2010.

[212] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *Workshop on Wireless Security*, 2003.

[213] Z. Shen, K.-L. Ma, and T. Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, 12(6):1427–1439, 2006.

[214] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux. Quantifying location privacy. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11'.

[215] C. A. Shue, M. Gupta, J. J. Lubia, C. H. Kong, , and A. Yuksel. Spamology: A study of spam origins. In *The 6th Conference on Email and Anti-Spam (CEAS)*, 2009.

[216] A. Shulman. The underground credentials market. *Computer Fraud & Security*, 2010(3):5–8, March 2010.

[217] K. Singh, S. Bhola, and W. Lee. xbook: redesigning privacy control in social networking platforms. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 249–266. USENIX Association, 2009.

[218] A. Smith. Nigerian scam e-mails and the charms of capital. *Cultural Studies *, 23(1):27–47.

[219] J. Staddon and A. Swerdlow. Public vs. publicized: content use trends and privacy expectations. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security*. USENIX, 2011.

[220] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, November 2009.

[221] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Proceedings of 2010 Annual Computer Security Applications Conference (ACSAC'10)*, December 2010.

[222] L. Sweeney. Uniqueness of Simple Demographics in the U.S. Population.

[223] L. Sweeney. k-anonymity: a model for protecting privacy. *Intl. Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 10:557–570, October 2002.

[224] Symantec. Spam report: Hacked personal email accounts used to scam contacts. `http://www.symantec.com/articles/article.jsp?aid=20080729_spam_report`.

[225] Y. Taigman and L. Wolf. Leveraging billions of faces to overcome performance barriers in unconstrained face recognition. *CoRR*, abs/1108.1122, 2011.

[226] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09', pages 807–816, New York, NY, USA, 2009. ACM.

[227] K. Thomas, C. Grier, and D. M. Nicol. unfriendly: Multi-party privacy risks in social networks. In *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, 2010.

[228] B. M. Thuraisingham. Directions for security and privacy for semantic e-business applications. *Commun. ACM*, 48(12):71–73, 2005.

[229] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 75–86. ACM, 2011.

[230] Twitter. Twitter Blog - Election Night 2012. `http://blog.twitter.com/2012/11/election-night-2012.html`.

[231] B. E. Ur and V. Ganapathy. Evaluating attack amplification in online social networks. In *Proceedings of the 2009 Web 2.0 Security and Privacy Workshop*.

[232] S. Utz. Show me your friends and i will tell you what type of person you are: How one's profile, number of friends, and type of friends influence impression formation on social network sites. *J. Computer-Mediated Communication*, 15(2):314–335, 2010.

[233] S. Vikram, Y. Fan, and G. Gu. SEMAGE: A New Image-based Two-Factor CAPTCHA. In *Proceedings of 2011 Annual Computer Security Applications Conference (ACSAC'11)*, December 2011.

[234] T. Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 22(176), 1975.

[235] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[236] L. von Ahn, B. Maurer, C. Mcmillen, D. Abraham, and M. Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895), August 2008.

[237] J. L. Wang and M. C. Loui. Privacy and ethical issues in location-based tracking systems. *Technology and Society, International Symposium on*, 0:1–4, 2009.

[238] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge Univ Pr, 1994.

[239] S. Webb, J. Caverlee, and C. Pu. Social honeypots: Making friends with a spammer near you. In *Proceedings of the 5th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2008.

[240] M. W. Wilson. Location-based services, conspicuous mobility, and the location-aware future. *Geoforum*, 43:1266–1275, 2012.

[241] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. *Security and Privacy, IEEE Symposium on*, 0:223–238, 2010.

[242] Y. Xie, F. Yu, K. Achan, R. Panigrahy, and G. Hulten. Spamming botnets: signatures and characteristics. In *In SIGCOMM*, 2008.

[243] T. Xu and Y. Cai. Exploring Historical Location Data for Anonymity Preservation in Location-Based Services. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[244] T. Xu and Y. Cai. Feeling-based location privacy protection for location-based services. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, 2009.

[245] W. Xu, F. Zhang, and S. Zhu. Toward worm detection in online social networks. In *Proceedings of 2010 Annual Computer Security Applications Conference (ACSAC'10)*, December 2010.

[246] W. Xu, X. Zhou, and L. Li. Inferring privacy information via social relations. In *ICDE Workshops*, pages 525–530, 2008.

[247] W. Xu, X. Zhou, and L. Li. Inferring privacy information via social relations. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 525–530, 2008.

[248] S. Yardi, N. Feamster, and A. Bruckman. Photo-based authentication using social networks. In *Proceedings of the first workshop on Online social networks*, WOSN '08'. ACM, 2008.

[249] E. Zheleva and L. Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, pages 531–540, New York, NY, USA, 2009. ACM.

[250] Zhou, Bin and Pei, Jian. Preserving Privacy in Social Networks Against Neighborhood Attacks. In *2008 IEEE 24th International Conference on Data Engineering*, pages 506–515. IEEE, April 2008.

[251] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10'. ACM, 2010.