Computer Science Department
University of Crete

Anontool: Per Application Field Anonymization to Promote Network Data Sharing

Master's Thesis

Michael Foukarakis

October 2008
Heraklion, Greece

University of Crete

Computer Science Department

Anontool: Per Application Field Anonymization to Promote Network Data Sharing

Thesis submitted by

Michael Foukarakis

in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

THESIS APPROVAL

Author: _____

Michael Foukarakis

Committee approvals: _____

Evangelos P. Markatos

Professor, Thesis Supervisor

_____

Sotiris Ioannidis

Visiting Professor

_____

Maria Papadopouli

Assistant Professor

Departmental approval: _____

Panos Trahanias

Professor, Chairman of Graduate Studies

Heraklion, October 2008

Abstract

As computer networks grow in size and complexity, the need for distributed network management and monitoring becomes increasingly important. Network data is the single most valuable resource available to network analysts and security professionals, yet organizations and researchers are still reluctant to share data with third parties. As a result, there is a lack of realistic network traces for research studies and prototype testing and poor cooperation in network defense.

To alleviate this problem and limit sensitive information leakage, anonymization is often applied to network data prior to being publicized. Anonymization aims to obfuscate data to protect the privacy of monitored subjects, while preserving useful information about the data. Today's approaches in this field are software utilities or ad-hoc solutions which offer limited flexibility or performance.

Our proposal is *Anontool* , an application which aims to provide a flexible and efficient solution to deal with anonymization on every layer of a network packet. *Anontool* uses the Anonymization API (AAPI) and extends it to support the popular NetFlow and IPFIX protocols. We also developed two new anonymization primitives to address attacks against the anonymized traces' privacy. Furthermore, our implementation finds and anonymizes sensitive information within binary packet payloads, in particular malicious executable payloads. Our evaluation shows that *Anontool* is one of the most flexible and powerful tools currently available. Our experimental results show *Anontool* outperforming tools with similar functionality and having similar performance with specialized tools.

Supervisor: Professor Evangelos P. Markatos

Anontool: Διατήρηση Ανωνυμίας Ανά Πεδίο που Προάγει τη Διανομή Δεδομένων Δικτύου

Φουκαράκης Μιχαήλ

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Καθώς τα δίκτυα υπολογιστών μεγαλώνουν σε μέγεθος και πολυπλοκότητα, η ανάγκη για κατανεμημένη διαχείριση και παρακολούθηση δικτύων γίνεται ολοένα και πιο σημαντική. Τα δεδομένα δικτύου αποτελούν τον πιο σημαντικό πόρο στη διάθεση των αναλυτών δικτύου και των επαγγελματιών ασφαλείας δικτύων, όμως οι οργανισμοί και οι ερευνητές ακόμα διστάζουν να μοιραστούν δεδομένα με τρίτους. Σαν αποτέλεσμα, υπάρχει μια έλλειψη ρεαλιστικών συνόλων δεδομένων δικτύου για ερευνητικούς σκοπούς και έλεγχο πρωτοτύπων, καθώς και φτωχή συνεργασία σε θέματα δικτυακής ασφάλειας.

Για την ανακούφιση από αυτό το πρόβλημα και τον περιορισμό διαρροής ευαίσθητων πληροφοριών, η διαδικασία του anonymization (διατήρηση ανωνυμίας) συχνά εφαρμόζεται στα δεδομένα πριν από τη δημοσίευσή τους. Η διατήρηση ανωνυμίας σκοπεύει να τροποποιήσει τα δεδομένα ώστε να προστατεύσει τη μυστικότητα των υπό παρακολούθηση οντοτήτων, ενώ ταυτόχρονα να διατηρήσει χρήσιμες πληροφορίες μέσα σάυτά. Οι σημερινές προσεγγίσεις στο πεδίο αποτελούνται από εργαλεία λογισμικού και ad-hoc λύσεις που προσφέρουν περιορισμένη ευελιξία ή/και απόδοση.

Η πρότασή μας είναι το Anontool, μια εφαρμογή με σκοπό να προσφέρει μια ευέλικτη και αποδοτική λύση για διατήρηση ανωνυμίας σε οποιοδήποτε στρώ-

μα ενός πακέτου δικτύου. Το Anontool χρησιμοποιεί το Anonymization API (AAPI) και το επεκτείνει ώστε να υποστηρίζει τα δημοφιλή NetFlow και IPFIX πρωτόκολλα. Επίσης αναπτύξαμε δύο νέες μεθόδους για διατήρηση ανωνυμίας για να αντιμετωπίσουμε επιθέσεις στη μυστικότητα δεδομένων στα οποία εφαρμόζεται διατήρηση ανωνυμίας με τωρινές μεθόδους. Επιπλέον, υλοποιήσαμε ένα τρόπο να βρεθούν και να διατηρηθεί η ανωνυμία ευαίσθητων δεδομένων μέσα σε πακέτα που περιέχουν κακόβουλο εκτελέσιμο κώδικα. Η ανάλυσή μας δείχνει πως το Anontool είναι ένα από τα πιο ευέλικτα εργαλεία που είναι διαθέσιμα αυτή τη στιγμή, και τα πειραματικά μας αποτελέσματα δείχνουν πως είναι αποδοτικότερο από εργαλεία με παρόμοια λειτουργικότητα και συναγωνίζεται πολύ εξειδικευμένες προσεγγίσεις.

Επόπτης Μεταπτυχιακής Εργασίας: Ευάγγελος Π. Μαρκάτος

Parts of this work have been published at the Third International Workshop on the Value of Security through Collaboration (SECOVAL), September 2007, Nice, France, and at the FloCon Workshop, January 2008, Savannah, Georgia, USA.

Acknowledgments

I'd like to thank my advisor and supervisor, Professor E.P. Markatos, for his assistance and guidance during my academic steps in the field.

Many thanks to the DCS Laboratory members for all their help, support, collective wisdom, nights out and valuable comments on this work.

Also, to my family and friends. None of this would have happened without them.

x

To my family.

Contents

List of Figures

List of Tables

1

Introduction

As computer networks constantly evolve and grow in size [10], the need for distributed network management and monitoring becomes more important than ever. To carry out management and research, both theoretical as well as experimental, vast and evergrowing amounts of network data and traces must be manipulated. Monitored network traffic allows researchers to study the characteristics of networks and observe the patterns in network behaviour, permits security analysts to evaluate network defense mechanisms and open up several other possibilities to network developers and educators. Collection and management of network data is also a vital issue to network administrators.

Although network data traces and activity logs are probably the single

most valuable resource to network analysts and security professionals, organizations and researchers are reluctant to share their data with third parties, in fear that they may contain information they deem private or sensitive. This fundamental lack of trust is a huge obstacle to the cooperation between different organizations. Some reasons behind this reluctance are concerns over user privacy and fear of exposing details of an organization's internal infrastructure. More than often the organizations involved in potential data exchanges have conflicting interests, as is usually the case with large ISPs.

To alleviate the problem of information leakage and control the amount of information that is being exchanged or revealed in the process of publishing network data, traces containing network data are usually anonymized prior to being publicized. Typical anonymization approaches operate on the TCP/IP layer headers and obfuscate or completely remove relevant information. Today's organizations provide network traces that are anonymized by software utilities or ad-hoc solutions that offer limited flexibility. As a result they can only provide unrealistic traces which are often inappropriate for use. There are also plenty of cases where data of a specific type is simply not shared because there is no appropriate agent able to anonymize them. Specifically, we lack coordinated network data from distributed and diverse sources to perform research, erect effective network defenses and perform computer forensics effectively in the case of malicious activities involving the network.

Our work aims in filling notable gaps in the field of network data anonymization. We have developed an anonymization tool, based on an existing framework [49], that aims to cover all anonymization needs, including those which are not addressed by existing tools. Besides the functionality offered by AAPI, we extended it to address open problems in network data anonymization. More specifically, we are targeting three significant opportunities to promote information sharing; the NetFlow and IPFIX network log activity

formats and executable payloads, as well as providing new anonymization primitives to help create policies that are better at preserving users' privacy.

The Cisco NetFlow [36] format is a popular format used by network activity monitoring tools and/or agents. It is based on the concept of a flow, which Cisco defines as a set of packets that have the following five properties in common: source and destination IP address, source and destination port numbers, and the IP protocol value. The most recent evolution of the NetFlow format is NetFlow version 9, which is also the basis for the IETF standard for information export (IPFIX) [11]. Given this fact, NetFlow and IPFIX are likely to gain even more in popularity, yet very few anonymization tools completely support them, leaving plenty of work to be done.

Another important source of information, especially for security professionals, is traces of network activity containing worm code which propagates through networks, or other sorts of exploits in the wild. Knowledge about them and its sharing, can be critical if a quick response to fast-spreading malware is required. The PREDICT project (Protected Repository for the Defense of Infrastructure Against Cyber Threats) [23], developed by the U.S. Department of Homeland Security, is an excellent example of an initiative which aims at that purpose. Data sets of the type of timeliness proposed by PREDICT are virtually unavailable today. This lack of critical data restricts the development of computing infrastructure and the related research. To further enhance and promote contributions to such repositories, anonymization of network data traces containing cyber-threats is a valuable asset in ensuring that both business intelligence and individual privacy will not be compromised.

Furthermore, the data sharing community relies on few anonymization policies as a panacea for all applications. Past work [63] has shown that popular solutions, such as prefix-preserving anonymization on IP addresses can still be efficiently attacked with moderate effort. We demonstrate two

similar attacks on the anonymization of NetFlow traces that are not specific to a single primitive like the prefix-preserving algorithms but are rather conceptual in nature. Nonetheless, we propose, discuss and evaluate two primitives that can defeat such attacks.

To deal with Cisco NetFlow logs and binary payloads, we will describe in detail the design and implementation of anonymization support for Cisco NetFlow datagrams, versions 5 and 9 as well as IPFIX datagrams within *Anontool* . We will also demonstrate two possible attacks against anonymization of NetFlow logs and provide two anonymization primitives which can help defend against them. Furthermore, we provide and examine a proof-of-concept *Anontool* extension which is currently able to detect potentially sensitive information within binary payloads encrypted under several popular XOR decoders.

1.1 Contributions

The novel contributions of this thesis are the following:

- An open-source, efficient and extensible anonymization tool that works on all layers of a network packet and supports popular application-level formats, such as NetFlow and IPFIX.
- Two new primitives (bi-directional mapping and random value shifting) which can help defend against attacks on anonymization of NetFlow logs.
- A proof-of-concept implementation which can anonymize sensitive information inside binary packet payloads.

1.2 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 will explain the necessary background information on the field of network data anonymization and place our work in context with past research. Chapter 3 presents

and examines the state-of-the-art anonymization solutions that have been proposed or are implemented. Chapter 4 includes problem statement and describes in detail our approach for solving them. Chapter 5 describes our evaluation methodology for each of our contributions and presents our analysis of all of the aforementioned elements, as well as our experimental results. Finally, Chapter 6 concludes this thesis and outlines potential topics for future work.

2

Motivation

This chapter will present the necessary background information on the topic of network data anonymization, such as the problem of network data sharing and the related trust issues, then define anonymization and its objectives. We will thoroughly describe the context of this thesis and place our work inside it, to explain the motivation behind our ideas.

2.1 Network Data Sharing

The Internet is growing every day and with it the world becomes more and more interconnected, allowing us to perform tasks in new ways. This constant growth enables the development of a multitude of new services and protocols, which aims to provide efficient and accessible network applications. According to [10], we are seeing a proportional growth of com-

puter networks, which is consequently reflected on the amounts of data exchanged over those networks, as well as on the amount of computer and network attacks observed, which the Internet enhanced by providing a sense of anonymity.

This evolution of computer networks is certainly reflected on the data passing through them, and data by itself provide evidence of those changes in the infrastructure, respectively. Logs and traces of network activity are therefore the single most fundamental resource available to security professionals and network analysts. They provide the means to understand network operations and threats, observe patterns and predict future needs, enhance operational and security policies of organizations and they also help design, deploy and evaluate new network algorithms and applications.

To further motivate the reader, we argue that the development of tools for computer forensics, network intrusion detection and network log analysis and accounting require true raw network data, as opposed to synthetic models or simulated traffic. While malicious worms that are deployed without further human intervention can be adequately simulated [75], this cannot be the case with human motives and complicated multi-phase attacks. Furthermore, to train, evaluate and refine those tools we need diverse data that naturally cannot be gathered completely from one vantage point.

In addition to tool development, we argue that educators also need real network data to effectively teach computer security forensic classes. Real data should be desired in order to have meaningful student projects; while simulated data is usually enough for networking classes—as usually happens with the popular NS2 simulator—they cannot suffice for security courses. At a higher level, security training courses from institutions like SANS [30] *require* students to bring sample logs to class, and while this may be easy for an established security administrator, it is near impossible for one who is receiving training in the hope of moving towards such a position. Thus, the

latter actually depend on network data sharing in order to become network administrators.

They are also important resources to network administrators and educators, both in traditional academia as well as the forensics department. The SANS Institute is a good example of such an organization which requires access to network logs. There is also a general desire to create centralized repositories [5] for network security research and network measurement studies, such as DatCat [27]. To study such vast quantities of raw data, it is reasonable to rely on distributed measurement, monitoring and analysis by several different parties, which makes log sharing essential. The academic world, the industry [12], as well as governments [28] all recognize the importance of sharing logs of network activity. This leads to an increased need and significant popularity for network activity log sharing, which based on observations of today's computer networks, is not expected to diminish anytime soon.

2.2 Lack of Trust

Today's organizations' security policies typically involve pushing offending elements away from their private networks. Furthermore, they are not concerned with alerting other organizations about this kind of activity. For example, when an organization detects an Internet attack, such as a SYN scan of their subnets, its administrators will block the originating IP addresses of the attack at the network border as a general rule of thumb and refuse to further deal with it. Of course, some security engineers and administrators have established trusted channels for sharing security events, but at this time, this is the exception rather than the rule. In this way, administrators may often fail to realise their organization is just a part of a larger target.

The Cooperative Association for Internet Data Analysis (CAIDA) [6]

also recognizes that one of the major obstacles to progress is the lack of data sharing [31]. Not only is traffic data off limits, but sharing data on the structure of the network is forbidden too—commercial ISPs are typically not even allowed to disclose the existence of peering agreements, much less their terms. So when developing tools for accurate Internet mapping, researchers cannot validate the connectivity inferences they make, since the information is typically intended to be secret. They note that the real obstacles mostly pertain to ownership and trust constraints rather than financial or technical, however the document focuses more on the legal aspect of the problem.

On the other hand, data sharing is quite common among attackers [44]. They share or trade information on vulnerable systems and/or networks by publicly posting it, they trade zombie machines and use this information to perform very effective coordinated attacks. Past events [41] have demonstrated the effectiveness of such coordinated action, and pointed out that organizations do not have efficient mechanisms in place to share, correlate and exploit data. The open problem is to tap into the multitude of the available data sources and extract and share the critical information inside the raw data, while satisfying the data owner's concerns.

The reluctance of data owners to share logs is understandable, due to the highly sensitive data that is captured within logs. The amount of sensitive data within a log does not only depend on the granularity of data contained within, but also on several external factors. For example, a log may be mishandled by a friendly peer; made publicly available by accident or malicious intent and fall into the hands of attackers directly or indirectly. Even in the case of sharing between friendly peers, it is easy to be held liable for the security compromise of a third party.

The reasons behind this reluctance to share data vary. To start with, several parties are interested in network data; researchers, developers, engineers and administrators, educators and so forth. These parties have different in-

terests in the data, potentially conflicting ones. Also, the subsets of data that interest, for example, a particular group of researchers might be irrelevant to network administrators or a specific task they aim to accomplish using the same set of network logs. At the same time, data sharers must carefully preserve a balance between the security needs of their organization and the usefulness of the anonymized/obfuscated logs they provide.

The problem with promoting network data sharing is therefore twofold. Firstly, there is the issue of analyzing and understanding the fundamental reasons behind the lack of trust between different parties. Reasons that go far beyond the scope of this thesis, which addresses the technological part, into the social and/or economic interests of each organization. Secondly, deploying the mechanisms necessary to battle this reluctance by providing tools and knowledge that will help in sharing network data without exposing sensitive information.

2.3 Anonymization as a Means to Promote Data Sharing

Raw network data can be used for a variety of tasks. One could use them to determine weak points in a network, vulnerable hosts and single points of failure. They can also indicate potential bottlenecks, which in turn can be used to launch more effective DoS attacks against the infrastructure. Host activity can also be used to find unpatched servers and exploit them. Packet payloads may contain all sorts of sensitive information; and sometimes sensitive information is hard to identify because it depends on special circumstances or requires context knowledge which is not contained in the network data.

To enforce control over the data, one could have a careful screening process for potential recipients, or even centralized repositories with a high level of physical as well as additional code security measures could be used to

address some of these concerns. However, anonymization techniques provide a clearly more flexible solution when access to raw logs is needed (otherwise, privacy preserving data mining techniques already address the issues when access to data is done through specialized query interfaces).

Several of the tools available today lack efficiency in many ways; they consider only a small subset of network data types, or focus on few packet fields and ignore the rest. It is very clear that IP addresses and abstract payload views are not the only sensitive fields, and they are even not the only ones directly identifying hosts. For instance, a web server could very easily be identified as a host that only accepts incoming connections on port 80 and has no further interaction with other hosts. Lots of other behavioural criteria, often being heuristics, can help attackers extract useful information from network data. Tools that provide flexible and efficient solutions are desired in order to provide the mechanisms required to deal with the practical issues of anonymization and take research to the next level, quantifying the information loss tradeoff and detailed evaluation of anonymization policies. Indeed, even if there are some problems with sharing of network data, the solution comes through extended use of anonymization; one can always share what won't harm them.

Anonymization is the process of removing, hiding, or obfuscating information in data logs which might be considered sensitive. Past work [46, 53, 55, 70] has identified an inherent tradeoff between the amount of information one is willing to reveal and the utility of the data. In the context we described above, the main objectives of the anonymization process are three. First, protecting the privacy of monitored users. Revealing information about users, transient or otherwise, within monitored networks is absolutely unacceptable. Examples of such information are the URLs of the web pages a user has visited in the past, unencrypted sessions that might reveal passwords, credit card numbers or other information about the user

(e.g. command history in a telnet session), peer-to-peer connections, email sent or received and so on. Privacy protection on its own is already complicated enough, so that major organizations choose to play safe and erase any parts of network traffic that might reveal such information. For example, the National Laboratory for Applied Network Research (NLANR) Project [18] releases packet trace datasets which only contain header information, skipping payloads in their entirety.

Secondly, and closely related to the aforementioned, is the objective to hide, or reasonably obfuscate, information about the internal infrastructure and configuration of the monitored network(s). Ideally, an anonymized trace should reveal neither "alive" hosts inside the monitored network nor any of their characteristics that distinguish them from the rest, such as operating system configuration, activity patterns and so forth. A naive first approach would be to randomize the IP addresses of hosts, thus hiding host identity and subnet information, however there are also plenty of other ways to profile a host, such as its open or active port list. We already gave an example above of how such information might directly identify a web server.

Lastly, an anonymized network trace aims to be as realistic as possible, which essentially means being close to the original network trace in terms of usefulness to the particular study for which they are intended. Evaluation experiments and network measurements are two good examples of such data. This requirement is generally conflicting with the other two, and is being analyzed in detail in various areas of application [63], [52], [70], [73].

2.4 Areas of Application

Anonymization techniques should be able to be applied to any part of a network packet that one might wish. The aforementioned anonymization objectives dictate the need to obfuscate any and all fields present inside a packet, regardless of application, transport, or network layer protocols used,

according to application needs. Anonymization operations could range from a simple modification of a field like the IP address to elaborate obfuscation of timestamp values across many packets or string modifications inside application protocols like HTTP and SMTP. As discussed earlier, sensitive information could be present anywhere within a packet, depending on circumstance and malicious intent. One important factor in gaining some *a priori* knowledge or rather insight into anonymization needs is protocol popularity, as it is almost certain that dominant protocols carry larger amounts of information, which in turn is easier to exploit even based on sheer brute force. Popular protocols today include, but are not limited to, the Internet Protocol (IP), the TCP and UDP transport layer protocols and a wide variety of application protocols such as DNS, SMTP, HTTP, NetFlow, the various file transfer/sharing protocols and so forth. Ideally, we would like a mechanism to anonymize all of these protocols, proprietary or not. Many current anonymization tools or solutions do not extend past the transport layer headers, although there are some exceptions to this case. AAPI [49] and subsequently *Anontool* are being developed with that idea in mind, not trying to provide one universal mechanism for all known protocols (which might not even be possible), but rather providing a flexible and extensible implementation that can be easily extended to support any new protocol.

3

Related Work

This chapter will present a categorized overview of the available anonymization solutions or tools which are based on past research efforts and comprise the related work in the area of network data anonymization. A short description and evaluation is included for each tool or approach that is used today to anonymize network data. This listing will provide the reader with an excellent opportunity to place our work with *Anontool* in context. Table 3.1 shows a comparative summary of each tool's capabilities.

3.1 TCPdpriv

TCPdpriv [45] is a well-known anonymization tool that takes as input traces written in tcpdump [25] format and removes sensitive information by operating only on packet headers. TCP and UDP payload is simply removed,

while the entire IP payload is discarded for protocols other than TCP or UDP. The tool provides multiple levels of anonymization, from leaving fields unchanged up to performing more strict anonymization, like mapping IP addresses to integers or prefix-preserving anonymization. Tcpdpriv works only on TCP/IP headers, thus it does not provide any functionality for Netflow anonymization. Ip2anonip [40], a tool based on TCPdpriv, is a simple filter that turns IP addresses into host names or anonymous IPs. Ipsumdump [42] dumps packets into ASCII format and uses TCPdpriv to anonymize IP addresses if specified by the user.

3.2 Prefix-preserving Anonymization

Peuhkuri in [59] addressed the problem of IP address anonymization. Cryptographic algorithms that require small amount of memory are applied in order to provide consistent anonymization across different sessions. Xu, Fan and Ammar in [72] and [71] also applied cryptographic algorithms to provide prefix-preserving anonymization. Crypto-PAn [37] is a cryptography-based sanitization tool for network trace owners to anonymize the IP addresses in their traces in a prefix-preserving manner. Crypto-PAn is provably as secure as the TCPdpriv scheme and provides consistent prefix-preservation in a large scale distributed setting. Slagell, Wang and Yurcik in [32] extended the Crypto-PAn module, which provides a well-known prefix-preserving anonymization scheme, with an integrated passphrase-based key generator and support for NetFlow logs.

3.3 NetDuDe

NetDuDe [50] is a GUI-based tool for interactive editing of packets in tcpdump files. NetDuDe itself does not perform parsing of application-level protocols in the payload, but offers the option for plug-ins to perform packet processing such as recomputing checksums.

3.4 SCRUB-tcpdump

SCRUB-tcpdump [70] is a set of functions that are used to anonymize a packet trace in libpcap format so that it can be shared without jeopardizing the anonymity of the network represented by the captured trace. It too does not perform payload inspection, or application-level protocol decoding.

3.5 CANINE

CANINE (A Combined Conversion and Anonymization Tool for Processing Netflows for Security) is a tool which combines conversion and anonymization capabilities. As a converter, CANINE augments existing flow tools as it enables tools working exclusively with one type of NetFlows to operate on data from NetFlows in other formats. It supports the following Netflow formats: Cisco NetFlow versions 5 & 7, NFDUMP, and two proprietary NCSA formats derived from Cisco NetFlows and Argus [4] NetFlows. As an anonymizer, CANINE addresses problems with sharing sensitive logs by providing its users with multiple methods of anonymizing the following fields: IP address, timestamp, port number, protocol number and byte count. It is implemented in Java, which makes it relatively hard to script and use in conjunction with automated tools for network management.

3.6 NFDUMP Tools

NFDUMP [20] is a set of tools for collection and processing of NetFlow data. The *nfdump* tool among them reads NetFlow log files stored by nfcapd and performs prefix-preserving anonymization on them. It is worth noting that *nfdump* uses the Crypto-PAn module to perform this kind of anonymization, and the key for the cryptographic algorithm is user-supplied. A basic principle of the NFDUMP suite is the separation of the storing process from analyzing the data. As a result, a limitation of NFDUMP is the inability to perform anonymization on live traffic (ie. NetFlow export records as sent

by Cisco routers etc.), since it can only process stored log files. The current NFDUMP version is 1.5.6, currently offering support for Cisco NetFlow versions 5, 7 and 9.

3.7 FLAIM

FLAIM [66](Framework for Log Anonymization and Information Management) is a general framework, created to support the anonymization of heterogeneous logs to multiple levels. FLAIM was developed by the Log Anonymization and Information Management (LAIM) Working Group [15] to overcome the limitations of other tools, such as CANINE [54], which could not be scripted from the command line, and did not offer support of multiple types of logs. FLAIM includes an anonymization engine containing a broad set of anonymization algorithms for various datatypes, an XML based policy engine which validates and parses users' XML policies against a variety of schemes and finally an API governing how parsing modules can pass records back and forth with FLAIM's anonymization engine. At this time, the FLAIM nfdump module supports anonymization of netflows contained only in NFDUMP version 1.4.x logs and not 1.5.x ones, due to changes in the internal NFDUMP format. As a result, it does not support NetFlow version 9. FLAIM provides several anonymization primitives to choose from, such as prefix-preserving anonymization, random permutations of field values and specialized operations on time-related fields. FLAIM focuses on providing generality rather than performace; we believe that *Anontool* can provide the same, if not greater, degree of generality while also achieving the maximum performance, similar to very specialized tools with limited functionality, such as NFDUMP. The latest version of FLAIM is 0.7.0.

3.8 Other Approaches

Paxson and Pang in [58] introduce a way to anonymize the payload of a packet and remove sensitive information instead of removing the entire pay-

load as the other approaches do. Packets are reconstructed into data stream flows and application level parsers modify the data streams as specified by a policy written in a high-level language. The user can specify the field to be altered using regular expressions and the modification to be done. To the best of our knowledge, this work has not yet been extended to Netflow protocol (currently only HTTP and FTP are supported). tcpmkpub [57] is a tool for anonymizing packet headers in trace files. As such, it is subject to the same limitations as several of the tools we already discussed, being unable to operate on the application level. Its latest version is 0.1, as of August 2007, so unfortunately it may be considered a largely inactive project.

3.9 Privacy-Preserving Data Mining Methods

The area of privacy-preserving data mining methods is closely related with the anonymization goals and practices that apply in the area of network data sharing. Sweeney in [67] introduces the model of *k-anonymity*, where the information for one entity cannot be distinguished from at least k-1 other entities whose information also appears in the same release. Meyerson and Williams in [33] have proved that achieving optimal k-anonymity is an NP-hard problem. However, optimizations have been proposed through the years [64].

Machanavajjhala et al. in [56] have shown that k-anonymity still suffers from severe privacy concerns and proposed the definition of *l-diversity*, and have shown it to be practical and able to be implemented efficiently.

Another novel approach to such privacy preserving data mining algorithms was proposed where the individual datum in a data set is perturbed by adding a random value from a known distribution. In these applications, the distribution of the original data set is important and estimating it is one of the goals of the data mining algorithm. This distribution is estimated via an iterative algorithm such as the Expectation Maximization [39] (EM)

algorithm which was shown to have desirable properties such as low privacy loss and high fidelity estimates of the distribution. This method applies to the primitives we are going to present in Section 4.4. Wu in [35] proposes further ways to reduce its computational costs.

Other alternatives to k-anonymity have also been proposed, such as [47], which enables us to limit the maximum confidence for sensitive inferences. All these approaches are not antagonistic to the technical aspect of anonymization which we focus on. Rather, they are complementary, in the way that the theoretical knowledge behind the aforementioned concepts helps us construct policies which enable low privacy risks with increased utility for anonymized traces. Our work focuses more on the technical aspect which enables such policies to be defined in a practical, real-world anonymization tool rather than the database systems on which the above references focus on.

	TCPdpriv	Crypto-PAn	SCRUB-tcpdump	tcpmkpub	FLAIM	Anontool
IP addresses	Yes	Yes	Yes	Yes	Yes	Yes
TCP/IP headers	Yes	No	Yes	Yes	Yes	Yes
Payload	No	No	No	Yes	Yes	Yes
Protocol decoding	No	No	No	Yes	Yes	Yes
Variety of functions	Yes	No	Yes	Yes	Yes	Yes
Function limitations	Yes	Yes	No	No	Yes	No
Extensible	No	No	No	No	Yes	Yes
HTTP	No	No	No	Yes	Yes	Yes
FTP	No	No	No	Yes	Yes	Yes
NetFlow v5	No	No	No	No	Yes	Yes
NetFlow v9	No	No	No	No	No	Yes

TABLE 3.1: Comparative presentation of available anonymization tools.

4

Anontool

Anontool aims to provide a flexible, yet easy to use, platform to deal with anonymization on multiple layers of network packet traces. Another goal is to combine its flexibility with high performance and efficiency, so no compromises are made when trying to choose a tool for anonymization. *Anontool* is built on top of the Anonymization Application Programming Interface (AAPI) [49] which is described as a generic and flexible framework which provides extended functionality, covers multiple aspects of anonymization needs and allows for fine-tuning of the desired privacy protection level. *Anontool* uses and extends AAPI with support for the NetFlow versions 5 and 9 and IPFIX protocols. Leveraging our experience with NetFlow trace anonymization, we discuss two potential attacks which may be used to com-

promise the privacy of anonymized traces and propose two primitives which help defend against them. *Anontool* also incorporates these primitives as part of its distribution. Furthermore, to stimulate sharing of malware traces, we provide a proof-of-concept implementation of a mechanism which can be used to detect and obfuscate sensitive information within executable payloads, and is also analysed in detail below.

4.1 AAPI

4.1.1 AAPI Description

The Anonymization Application Programming Interface (AAPI) is an API based on the C programming language. It allows users to apply anonymization primitives on traffic, live or stored. The C language was chosen to interface directly with traffic capturing libraries, avoiding the use of complex scripting languages from the user's point of view and for performance reasons.

It is clear that a generic global anonymization scheme can not exist since different organizations and applications have different needs. Network administrators should be able to specify their anonymization policies at varying levels of detail. As we have seen, most existing anonymization tools are not adequate enough to provide such flexibility and are not capable to address all anonymization needs, since most of the times they were built having a specific application domain, or a limited range of anonymization policies in mind. In all cases they work on predefined fields and most of them perform only header-level anonymization, ignoring the application layer. AAPI, on the other hand, offers a wide range of anonymization functions that can be applied to any field of a packet or a record, up to the application level. The expressiveness of the framework allows creation of anonymized traffic that is able to express any balance between privacy protection and realism.

A central notion of AAPI is that anonymization is a series of functions

that are applied to a traffic stream. The core functions of AAPI are divided into three main categories. First, there are the anonymization functions that *alter fields* of the packets or records in the given traffic stream, e.g randomize them or replace them, do prefix-preserving anonymization on IP addresses, etc. Secondly, there are *filtering* functions, including BPF filters and string searching. Filtering functions allow to distinguish parts of the traffic stream and apply complex policies such as *"leave all the UDP packets unchanged but randomize the payload of all TCP packets"* or *"anonymize all packets that contain the GNUTELLA-CONNECT pattern"*. Finally, we have *application-level stream* functions, which provide the ability to compose and decompose application-level streams from individual packets.

The main function call of AAPI is the *add_function(set, function, ...)*, where "..." denotes a variable number of arguments, depending on the specific function to be applied. AAPI expresses each anonymization policy as a single or multiple sets of functions. Each set is a logical group of functions that are executed sequentially one after the other, in the order they had been applied. Sets are created through the *create_set()* function. Once a packet is captured, it is passed through each set and for each set is processed by its functions. We should note here that a function can prevent the traversal of a packet in the subsequent functions by simply returning zero. This behavior enables filtering either by BPF filters or string matching algorithms. The combined flexibility of sets and filtering functions allows the user to express "if-else" scenarios or even express different anonymization policies within the same program. The function arguments define which specific function will be applied. Natively, AAPI supports "ANONYMIZE" (field anonymization), "BPF FILTER" (BPF filtering), "STR SEARCH" (string searching), "COOK" (stream reassembly) and "UNCOOK" (splitting a stream to its original packets). User functions can also be added in order to extend the function support. Note that the "COOK" and "UNCOOK"

functions can be applied transparently in the case of TCP streams without user-explicit action.

AAPI offers a wide range of anonymization functions that can be applied on any field of a packet, up to the application layer. All the widely used primitives that appear on similar tools are included. The expressiveness of AAPI allows creation of anonymization policies that are able to balance privacy protection and application utility for any given packet trace. In particular, anonymization functions include hashing (MD5, SHA1, SHA2), block ciphers (CRC32, AES and DES algorithms), random number generators for generic usage and random strings for filenames/URIs, mapping to either sequential values or based on some distribution (uniform, Gaussian, etc.), replacing with constant integers or strings, prefix-preserving for IP addresses (cryptographic and map based), regular expression substitution, checksum adjustments for all protocols, and removing fields mainly used for application-level protocols, thus providing adequate functionality for all user needs. Internally, AAPI performs sanity checks for each function applied before starting to process packets and informs the user in case a function is used in a not meaningful manner. For example, it does not make sense to remove (using the "STRIP" function) the IP header version field, because packets then become corrupt and unusable.

Prior to the work done as part of this thesis, AAPI supported IP, TCP and UDP, ICMP, HTTP and FTP protocols, including HTTP/1.1 features such as persistent connections.

AAPI is simple enough to use, since any anonymization policy is expressed as a set of function calls. Furthermore, the API's implementation is modular and extensible, so the user is given the ability to implement and incorporate her own anonymization functions into the framework. In Section 4.4 we make use of this ability to provide two new primitives for the purposes of anonymizing NetFlow records. For purposes of I/O, AAPI

currently uses *libpcap*. However the I/O code is modular, so it is easy to incorporate code that handles other input sources, such as text logs used by other tools like NFDUMP [20], Snort [65] alert logs or proprietary binary formats.

With AAPI, the user can split traffic into one or more "streams" with BPF filters, and apply a different anonymization policy on each. Application-level stream reconstruction is also possible, and is a transparent process. AAPI preserves stream information internally, so that it is able to deconstruct a stream to its individual packets accurately, after the anonymization policy is applied. Consider the following anonymization policy, which we will show how it can be implemented with AAPI. The policy is: *"remove the TCP payload for TCP packets, remove of IP payload for all other packets, all packets must have their IP addresses anonymized by mapping them to random integers"*. Before we proceed to the actual code, we should observe that this policy divides the packets into two categories, TCP and non-TCP. It is thus very useful to apply filtering functions to distinguish the packets and then for each category apply the appropriate anonymization functions. The "BPF FILTER" function returns zero if the filter does not match, elsewhere returns one and the packet is processed by subsequent functions. The given anonymization policy is implemented as follows with AAPI:

```
int set1=create_set();
int set2=create_set();
add_function(set1,"BPF_FILTER", "tcp");
add_function(set1,"ANONYMIZE", IP,SRC_IP,MAP);
add_function(set1,"ANONYMIZE", IP,DST_IP,MAP);
add_function(set1,"ANONYMIZE", TCP,PAYLOAD, STRIP);
add_function(set2,"BPF_FILTER", "ip and not tcp");
add_function(set2,"ANONYMIZE", IP,SRC_IP,MAP);
add_function(set2,"ANONYMIZE", IP,DST_IP,MAP);
```

```
add_function(set2,"ANONYMIZE", IP,PAYLOAD, STRIP);
```

Note that each packet will match to only one set (it can be either TCP or not) and in the case of TCP the "STRIP" function is applied to the TCP payload.

AAPI also implements function (re-)ordering to automatically detect common pitfalls when applying anonymization functions and ensuring the correctness of the semantics of the anonymization process. It accomplishes three main tasks:

- All anonymization functions except "CHECKSUM ADJUST", which adjusts the checksums to correct value, that are applied on IP, TCP, UDP or ICMP level are moved first. If they were placed between a "COOK" and an "UNCOOK" function, then the headers stored by "COOK" would not be anonymized and "UNCOOK" will emit non-anonymized packets.

- "CHECKSUM ADJUST" and functions that alter the packets length fields are applied at the end of the anonymization. "CHECKSUM ADJUST" is called last in order to reflect all changes, after the rest of the anonymization functions have been applied. Updating the packet length is also applied at the end because other anonymization functions may modify the original packet size. As a result, explicit modifications to the packet length must be performed at the end.

- If the policy requires to use functions that modify an application-level protocol (HTTP, FTP, etc.), they are all grouped together in order to apply "COOK" and "UNCOOK" only once. This means it also detects duplicate reconstruction attempts.

- Lastly, reordering detects and fixes common logic pitfalls of function usage. For instance, in a policy that hashes a URL string and then

removes it from the packet, the first modification is useless and never seen in the final result. It is therefore safe (and more efficient, as a side effect) to skip it.

The key point in AAPI is configurability, meaning the user can define any anonymization policy as a series of functions that are applied on packets. The main design goal is to facilitate the development of custom anonymization tools, that are able to implement both simple and complex policies, in a relatively small amount of lines of code. Furthermore, the framework is implemented in a modular way so it is fully extensible in terms of functionality, protocols and new traffic sources.

AAPI has also been successfully integrated with a large monitoring platform [17]. All the features of AAPI, as well as a performance analysis, are described in greater detail within [49]. The authors have confirmed that with the most commonly used policies, AAPI outperforms similar applications which offer but a fraction of the AAPI functionality.

4.1.2 Anontool

Anontool is a console application, written using the C language, that uses the AAPI library to perform anonymization of packet traces. It is implemented as a console application to allow for easy scripting from the command line, a feature which has hindered other tools, like CANINE [54] from being widely adopted. *Anontool* does not implement any anonymization functions in itself; it is much more transparent to the user to place all the anonymization functionality inside the AAPI implementation, which is a standalone library that can be used by any application. What *Anontool* does do, however, is to provide the user with the choice of protocols and functions to apply in order to create her anonymization policy for a packet trace. It is worth noting, that to maintain simplicity and not overwhelm the user with the vast amount of choices, we have not added support for every primitive AAPI provides.

On the contrary, we have provided a few preset policies which are commonly used and can be selected by a single command line parameter, and we are currently in the process of supporting predefined policies which are stored in files in an XML-derivative language. For instance, a user can invoke a predefined policy which will set IP source and destination addresses' bits to zero, set the values of the TCP port field into new random ones and replace the values of the Uptime field with a random value, and finally generate new checksums for the NetFlow datagrams before writing them to a file named *anon_trace.pcap* by invoking the tool as follows:

```
./anontool -i eth0 -d anon_trace.pcap
```

Enabling *Anontool* to use another primitive made available by AAPI and not currently implemented is as simple as adding another command line option for it, which any user with knowledge of the C language is able to do.

Anontool then enables users to select the desired anonymization function per field. It can read traffic either from a live interface or from a tcpdump [25] trace file. The anonymized packets may be written to disk, again in tcpdump format, . The choice of the tcpdump format was made based on the popularity of the format and the fact that can be given as input to many security and network management applications. Other useful options of *Anontool* are to automatically fix checksums of anonymized packets (the checksum will be changed once all other anonymization functions are applied on a packet, as we saw) and its ability to print packets on screen – in human readable form – for manual inspection. An example invocation of *Anontool* is the following:

```
./anontool -i eth0 -t ZERO -c /dev/null
```

The above invocation will open the NIC named "eth0" for packet capturing, will zero the TCP port numbers of NetFlow records in the packets captured

and recompute checksums, then write each packet to */dev/null*. Alternatively, a user could write:

```
./anontool -i eth0 -a PREFIX --NF5_TOS RANDOM -c 42.pcap
```

which would open the interface named "eth0" for capturing, then in every NetFlow datagram captured it would perform prefix-preserving anonymization on source and destination IP addresses and replace the value in the TOS field with a random value, then recompute checksums before writing the packets to a pcap file named *42.pcap*

In the following sections, we will expand on the non-trivial enhancements we incorporated inside AAPI and made available through *Anontool* .

4.2 NetFlow & IPFIX Anonymization

Since the emerging use of Netflow data, we decided to extend AAPI with support of the Cisco NetFlow packet export format. Taking advantage of the extensibility feature of AAPI we implemented decoding and anonymization functions for both version 5 and the newly defined version 9 of the NetFlow format. Table 4.1 shows all the fields and anonymization primitives available regarding NetFlow v5. Bear in mind the names of the functions are merely indicative, and most are highly configurable with extra parameters. The table does not contain all the NetFlow version 9 fields, which are in the vicinity of a hundred.

Exploiting the template-based nature of the NetFlow version 9 format, *Anontool* provides the user with complete control of every field made available from information export nodes, be it Cisco routers or network monitoring applications which support the NetFlow export format.

The implementation consists of two major components. The first is the decoding facility which identifies NetFlow information within packet payloads, and the second is the mechanism which enables anonymization. Due to the modular architecture of AAPI, implementing these two components

Protocol	Fields	Functions
NETFLOW_V5	VERSION	UNCHANGED
	FLOWCOUNT	MAP
	UPTIME	MAP_DISTRIBUTION
	UNIX_SECS	STRIP
	UNIX_NSECS	RANDOM
	SEQUENCE	HASHED
	ENGINE_TYPE	PATTERN_FILL
	ENGINE_ID	ZERO
	SRCADDR	REPLACE
	DSTADDR	PREFIX_PRESERVING
	NEXTHOP	PREFIX_PRESERVING_MAP
	INPUT	CHECKSUM_ADJUST
	OUTPUT	REGEXP
	DPKTS	BD_MAP
	DOCTETS	VALUE_SHIFT
	FIRST	
	LAST	
	SRCPORT	
	DSTPORT	
	TCP_FLAGS	
	PROT	
	TOS	
	SRC_AS	
	DST_AS	
	SRC_MASK	
	DST_MASK	

TABLE 4.1: Table presents the NetFlow 5 fields that AAPI makes available for anonymization and the functions which can be applied on them.

and putting them to work is a relatively simple process.

The decoder for NetFlow v9 datagrams iteratively tries to identify a NetFlow datagram by effectively laying the packet structure over the payload, starting with the NetFlow header which provides basic information about the packet - such as the NetFlow version, number of records contained within the packet, and sequence numbering, which enables lost packets to be detected. Following the packet header, an exported datagram contains information that must be parsed. These are one of two types of records, which Cisco refers to as "flowsets", template or data. A datagram may contain both types of flowsets, interleaved within the payload. Flowset headers allow us to distinguish between these two types and parse them accordingly, keeping track of all their fields in the process, to make them available to the user during the actual anonymization phase. Based on the header information, *Anontool* is able to enumerate all flowsets inside a NetFlow datagram and process them iteratively. For each flowset, a header structure similar to the basic NetFlow header is laid over the data and the flowset is correctly identified as template or data. Template flowsets have a simple, list-like structure and are the easiest to decode - we simply enumerate the fields present, check them against the Cisco specifications to verify field lengths and store them for future reference. Note that, for manufacturer-specified fields the previous verification step is skipped. In the case of option or data flowsets, the first field is the template flowset ID that describes the data contained. We use the previously mentioned list to recover the template with the corresponding ID and identify each field inside these flowsets, so that the anonymization component can traverse it and perform the actual anonymization process.

At any time during the decoding process, if the application encounters an unrecoverable error or corrupt data inside the payload data, the process is aborted for the malformed packet and an appropriate error message is

```
00000836  8A06          mov al,[esi]
00000838  3C99          cmp al,0x99
0000083A  7505          jnz 0x841
0000083C  46            inc esi
0000083D  8A06          mov al,[esi]
0000083F  2C30          sub al,0x30
00000841  46            inc esi
00000842  3499          xor al,0x99
00000844  8807          mov [edi],al
00000846  47            inc edi
00000847  E2ED          loop 0x836
```

FIGURE 4.1: A typical XOR decoder, used by the Wuerzburg shellcode.

generated for the user.

The anonymization component, having all the information and internal structure of the NetFlow datagram at its disposal, can track the desired field by enumerating all the flowsets, if necessary, and simply choose the relevant pointer to the packet data. The anonymization function for the NetFlow data is then, essentially, a simple iterative *switch* selection. There are naturally many optimizations which may be done, such as grouping fields over each flowset for faster access due to cache locality, and so forth, which can greatly increase efficiency and performance. Such optimizations have not yet been implemented, and are the subject of future work on our tool.

4.3 Binary Payload Anonymization

Traces containing executable code are collected and produced by a variety of applications. From simple traces can be collected from intrusion detection systems (IDS) [65] containing an attack that has passed through the network, to network traffic logs from the deployed honeypots and honeypot infrastructures [2], [9], [26], [21], as well as other related tools [60]. One such location can be found in [19].

In their effort to hide from simple payload level signature matching identification, attackers tend to use polymorphic or metamorphic techniques [13],

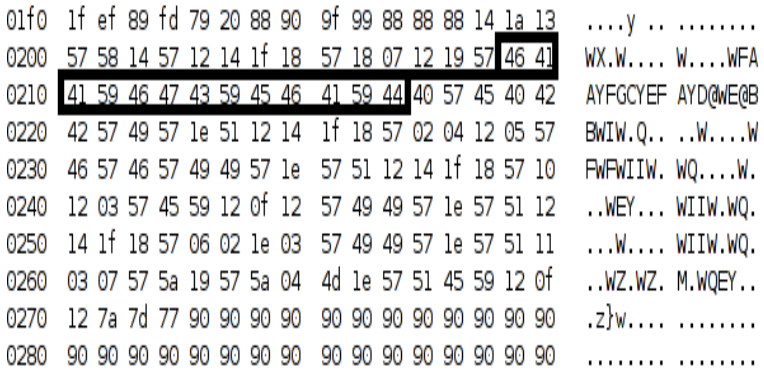
[62], [69], usually by encrypting the attack payload. Usually, a small decryptor precedes the shellcode in the packet payload. When the attack succeeds, it first executes the decryptor over the packet data to get the unencrypted payload that is going to be executed. An example of such a piece of code is the Wuerzburg shellcode [1] which contains a XOR decoder – pictured in Figure 4.1 –, and a connect-back file transfer code segment which connects to a host and downloads a file named *ftpupd.exe*. In Wuerzburg, the IP address and port are XOR'ed with a secondary key (which is a static value of *0xAAAAAAAA*) inside the XOR'ed exploit.

In the above case, sharing the attack trace as-is may leak information that the organization or owner of the host considers confidential. Any individual can, with little effort considering the available tools, decrypt the payload and get information from it. This information may be used for a subsequent attack to the organization or for naming the organization as vulnerable.

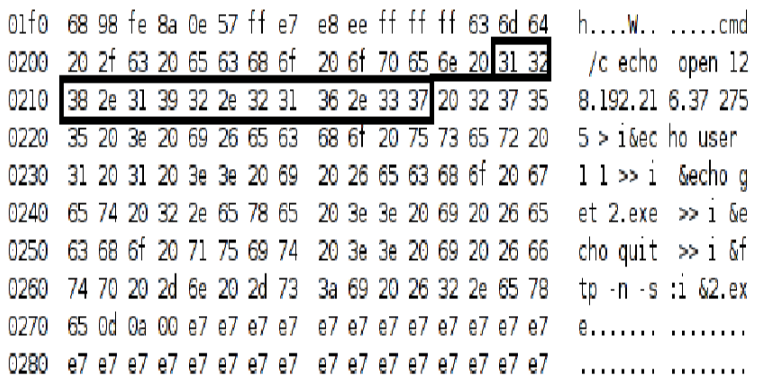
We further explain the problem by observing the actual data of the Wuerzburg shellcode. Figure 4.2(a) shows the original payload of the attack as seen in the wire and captured by a *tcpdump* session. Because of the encryption, the whole payload appears as almost random bytes and seems to contain no interesting information. However, if we decrypt the payload (Figure 4.2(b)), we can clearly recognize the actions of the attack. The payload first builds a small file with FTP commands and the executes the file to download a binary file (which is probably a malware). In the case of the example, the IP [1] address of the FTP server (128.192.216.37) might be information the corresponding organization would not like to be leaked, in order to not expose vulnerable hosts or not be advertised as a vulnerable organization.

The case with modern shellcodes is that they are compact, self-contained

[1] IP address is anonymized for the example above.



(a) Actual Shellcode



(b) Decrypted Shellcode

FIGURE 4.2: (a) An encrypted shell command, before decryption. (b) The same shell command after decryption. The IP and port numbers are clearly seen.

and lightweight pieces of code which exploit a vulnerability in the target service, acquire superuser access privileges and often connect back to a predefined host which is defined within the payload and transfer a rootkit or trojans, or anything that serves the will of the attacker.

The methods used by attackers in order to obfuscate their shellcode are several and range from simple XOR encoding to metamorphic payloads. Ways to identify malicious payloads range from simple regular expressions of invariant strings present within the payload (as is most usually the case with the Snort [65] intrusion detection system) to complex taint analysis within a sandbox environment, such as Argos [61].

It becomes clear that there is a gap to be filled when it comes to packet trace anonymization. There are major advantages to promoting attack trace sharing for the computer security industry, and providing the means to perform anonymization on them could lead to that direction. In the remainder of this section, we're going to relate our work with the modern methods that aim to identify binary payloads and explain our method of choice that will aid us in filling the aforementioned gap.

The first important design decision to be made was in deciding what mechanism to use in order to detect the various types of binary payloads and the sensitive information that may appear within. While there is a vast variety of ways to do that, we chose to implement regular expression matching to identify binary payloads combined with limited emulation to seek and match sensitive information such as IP addresses, URL's, and so forth.

The most important reasons that led to that decision are two. Firstly, regular expression matching is fast, and can be successfully used in deep packet inspection. It can also be made significantly faster, in case speed is an issue [74]. This provides the user with the option to anonymize attack traces on-the-fly, as they are produced by analysis and detection algorithms

```
"\\xEB\\x27(..)(....)\\x5D\\x33\\xC9\\x66\\xB9..\\x8D"
"\\x75\\x05\\x8B\\xFE\\x8A\\x06\\x3C.\\x75\\x05"
"\\x46\\x8A\\x06\\x2C.\\x46\\x34.\\x88\\x07"
"\\x47\\xE2\\xED\\xEB\\x0A\\xE8\\xDA\\xFF\\xFF\\xFF";
```

FIGURE 4.3: A regular expression matching the Wuerzburg shellcode.

and tools. Secondly, regular expressions are expressive enough so as to cover both the case where sensitive information such as an IP address appears within the payload of an attack, as well as when that information is masked by an encoder or packer which has to be executed first, before the actual payload is executed. In both cases, it is important to note that we are not aiming at providing a detection framework; several ways to do so have already appeared in the bibliography. We assume that our input is a trace of the network activity which is the result of such a framework, when it detects suspicious behaviour. That means the user already knows the specifics of the attack inside the trace and can therefore produce a regular expression to match the encoder, shellcode and private information she wishes to anonymize.

Such regular expressions are easy to produce, share, and acquire. There are many examples of databases of such regular expressions. Most notable are the Snort rule database [24], the *nepenthes* project page [21], as well as several others affiliated mostly with honeypots [2].

Table 4.2 mentions the names and short descriptions for each binary payload currently supported by *Anontool* , as well as the number of attack traces each signature matched. We don't make mention of few other signatures which did not match any of the traces processed. We chose the *nepenthes* project information on shellcodes as a point of reference because of its detailed information related to the code provided. In Figure 4.3, you can see the regular expression that matches the code in Figure 4.1.

The core of our implementation uses the PCRE library [22], to search

Name	Type	Matched traces	Comment
Bielefeld	connectback shellcode	9552	None
Metasploit PexEnvSub	xor decoder	2608	None
rbot 256 byte	xor decoder	2575	None
adenau	xor decoder	1133	None
halle	xor decoder	987	None
schoenberg	xor decoder	129	None
langenfeld	xor decoder	21	None
Leimbach	xor decoder	16	TFTP download
kaltenborn	xor decoder	15	None
Wuerzburg	connectback file transfer	1	None

TABLE 4.2: List of regular expressions incorporated into *Anontool* for binary payload anonymization.

for a given set of regular expressions characterizing different kinds of binary payloads within a packet trace. When found, we provide the user with the option of anonymizing the potentially sensitive information that may be contained in that piece of code. Instances of such information may be a hardcoded IP address inside the shellcode, which is frequently the case. The external host information may as well be obtained at runtime, so further inspection might be needed, or modification of the respective instructions.

This first pass can handle straightforward shellcodes which implement a reverse shell technique. If the matched regular expression identifies a decoder/packer, we need to emulate its behaviour, and then search for host information in the decrypted parts of the payload. The emulation process is carried out on a per-decoder basis. We do not use any emulation frameworks or external processes for this task, because most decoders are currently very

simple in their operation. For the decoders in our prototype implementation, we simply emulate in the application level the operations carried out by the decoder. We do this in a very similar way to the nepenthes low interaction honeypot. When the decryption process finishes, another scan is necessary to identify any possible information that may leak information; IP addresses, port numbers, URLs or anything else that may be used in order to fingerprint a host on the Internet (Fig.4.2(b)).

The user is then given the opportunity to manipulate all of this information as she deems fit. One should also take into account that there's the possibility for sensitive information to be leaked even when a hostname inside a shellcode is anonymized. For instance, if the shellcode executed on an infected machine opens a connection to a given IP address, say $a.b.c.d$, it is possible that the flow between these two hosts is also captured. Should the sequence of packets that comprise the conversation between these two hosts is included in the attack trace, an attacker may infer that $a.b.c.d$ is a host that quite possibly plays some part in spreading malware or is part of a botnet, and so on. It's obvious that the IP address $a.b.c.d$ also needs to be anonymized. One needs to be aware of the semantics of an attack trace to apply anonymization policies effectively and efficiently.

We already mentioned of faster alternatives to *libpcre* when it comes to regular expression matching. We choose not to incorporate them into the prototype implementation for the following reason. Packet trace anonymization is at the moment an offline process. Indeed, when it comes to anonymizing traffic as it appears on a network interface, speed is critical. However, when it comes to binary payloads, there are two processes which precede anonymization: the first is detection of the executable content, malicious or not, and the second is the analysis needed to reverse-engineer, classify and produce a signature for it. Although all of them could be considered time-critical components of computer security, analysis and classification is

a lengthy process, and is usually done manually and takes even few days for newly observed payloads (not variants of existing ones).

4.4 New Primitives

In this section, we describe in detail two attacks against conventional anonymization policies which are commonly used today and outline related work in both packet traces and flows.

We assume that the anonymization process is applied onto NetFlow traces which are generated by a router at the border of a monitored network, and export information for traffic entering and exiting this particular network. The network could be of any size or topology, from a small home network to larger networks belonging to research institutes, universities, and so on.

Our threat model assumes an active adversary that is able to direct traffic to the monitored network at will, has knowledge of the address space it occupies and can potentially compromise hosts inside it. We assume a rational attacker, for whom it is less costly or more useful, to "probe" and profile the monitored network before mounting attacks against it, to increase her chances for a successful attack. The adversary may also have several external hosts under her command. She is also able to gain access to the anonymized traces, which will most likely be publicly released.

The first attack, which we call *"Active Fingerprinting"*, aims to break the mapping algorithm when used on IP addresses. Mapping takes the set of IP addresses in a trace and performs a simple mapping function onto another totally different set. The second set may be the output of a deterministic function seeded by a random quantity, such as the *drand48()* family of functions [8], or a very simple sequential assignment of unique IP address numbers which results in a one-to-one mapping. This choice is irrelevant; the attack is independent of this decision.

The second attack aims at using the information about flow sizes contained inside NetFlow traces in order to deduce information about either hosts inside the monitored network or hosts that may be outside it, such as their IP address, network usage profiles, etc. We name this attack *"Statistical Signature Inference"*.

4.4.1 Attacking Applied NetFlow Anonymization Policies

Active Fingerprinting

The idea that active fingerprinting exploits is that the mapping between real and anonymized IP addresses is one-to-one. Consequently, if the mapping on one flow is discovered, the mapping on the whole trace is compromised. This attack has been described on packet traces in [48, 57] and we demonstrate its applicability on NetFlow records below.

Using this idea, an adversary can establish flows from a host under her control, which resides outside the victim network, to one or more victim hosts inside it. These flows will appear in the anonymized trace. The challenge for the adversary is to construct those flows in such a way that they will be easily distinguished in the final trace. This can be accomplished in a variety of ways; she can craft a flow with specific attributes which are known not to be anonymized (the list is as large as the potential fields listed in a NetFlow record, and may usually include TCP flags, IP ToS, and so forth), or in the unlikely scenario where flows are fully anonymized, she can use temporal patterns which are easy to detect. Even a specific packet size can be used as an identifier for the packets involved in a dictionary attack. If the traces from the monitored network span a wide enough time period, the latter scenario is very feasible as the trace contains a large number of attack packets.

This information can then be exploited in many ways. Consider this scenario: the attacker identifies a host inside the monitored network and

the mapping is therefore considered “broken”. She can now gain information such as, ”which sites does that host visit?”. She can acquire the IP address mapping information as described in [48] or by compromising a machine inside the monitored network and performing the aforementioned technique for hosts external to the monitored network as well.

A trivial way an adversary could create these flows is to perform a SYN scan on the victim network’s address space. In this case, even if there is a clear temporal pattern (handshake initiation - response from potential victim’s host) which is easily detectable in the anonymized trace, it can be defeated in an easy way. Setting only the SYN bit in TCP flags and setting the number of bytes to a specific quantity makes the adversary unable to distinguish live hosts from unused address space. Generally, there are many ways to craft these packets that can’t be known a priori.

We discuss a more general measure to defend against this kind of attack in Section 4.4.2.

Statistical Signature Inference

The idea behind this kind of attack is that each web page has a unique and complex enough structure which allows them to be identifiable despite our best efforts to anonymize their presence in NetFlow logs and preserve useful information in them as well.

A naive first effort would be the following. Consider the web sites **interesting.com** and **newssite.com**, and that a web session with each of them is **n** and **m** bytes long, respectively. The adversary can use one of the hosts under her command to initiate a web session to these sites and view the NetFlow records for source and destination IP addresses, port numbers, and the total size of traffic exchanged (which would reflect **n** and **m**). Assuming web page sizes do not radically differ from one session to another, and that NetFlow data records TCP traffic in its entirety, it is possible to filter out the set of web browsing sessions from an anonymized trace and construct

a frequency histogram with the number of bytes transferred in each flow. According to our assumptions above, it is possible to see a great deal of flows around the values of **n** and **m**. The adversary can employ the same tactic to find those flows, and then gather further information about hosts inside the monitored network, which can then be used to answer questions such as: *"What web sites does host A visit?"*, *"Which hosts do frequently visit www.google.com?"*, and make user profiles.

Past work [48] has demonstrated this kind of attack on packet traces. Recent work [38] has extended and demonstrated this attack on NetFlow logs as well. We argue that the fundamental property of web sessions that allows this kind of analysis to be exploited is the fact that web sessions to different hosts produce flows with similar characteristics, especially in flow size. In Section 4.4.2 we are going to view our proposal of a primitive which deals with this issue.

4.4.2 Countermeasures

The previous section described two attacks for revealing sensitive information from an anonymized NetFlow trace. In this section, we will describe our proposed ways to deal with the aforementioned attacks, and evaluate their consistency.

Bidirectional Mapping

We propose a way to deal with the issue that does not iteratively consider all the combinations of fields an adversary may use to craft her flows. Instead, we aim to eliminate the one-to-one host mapping property without losing all of the information the trace can provide. To this goal, we propose a bidirectional mapping to be used, that is different mapping for each traffic direction. Let **A** be the IP address of a live host inside the monitored network, and **B** the IP address of a host outside the monitored network. Conventional mapping functions would map a flow *(A, B)* to *(A", B")* and

a flow *(B, A)* to *(B", A")*. Bidirectional mapping maps a different address to **A** according to the direction of the flow that involves it. In the case of our example, the flow *(A, B)* would be mapped to *(A", B")* yet the flow *(B, A)* would receive a different mapping, say *(C, D)*.

Using this anonymization scheme by applying the function "BD_MAP" with AAPI, we prevent the attacker from identifying her own network flows inside the anonymized trace. Thus, it is made impossible to correlate her data with the trace information and reveal any sensitive data inside it. Also, most of the statistic information derived from the trace remains the same. We can still gather information about the incoming and outgoing traffic of the organization and identify the producers and the consumers of the network. Correlation of incoming and outgoing traffic for a specific IP address can not be done, but we argue that this is a general trade-off of the anonymization process and is up to the organization to decide whether to sacrifice sensitivity for usability in the data it makes public.

The implementation of such a primitive is quite easy. Regardless of the way a mapping for the host IP is produced, we basically want a different mapping based on whether the host initiates a given flow or not. Essentially, if the IP address is present in the source IP address, or the destination IP address field. We therefore preserve two tables with mappings; one with mappings for IP addresses that appears in the source IP field and another for the destination IP addresses. These tables' construction and update are completely independent, and even the algorithm used to obtain the mappings can differ between them. Bidirectional mapping is then very easy to implement, performing a hash table-like insertion on the aforementioned tables whenever an IP address is encountered. So, for any IP address, if that address is already present within the table, the mapping is taken and written on the corresponding field of the packet's IP header. Otherwise, a new mapping is generated using the appropriate algorithm selected by the user

and the IP header is modified in the same manner. The implementation of bidirectional mapping is already included in the stable version of *Anontool* .

Random Value Shifting

In order to diminish the viability of a statistical identification approach, but still be able to calculate some basic representative statistics about a NetFlow log, we propose a randomized shifting of values, which we will describe below.

Given a NetFlow data field with a given value range, our intent is to "scramble" its values across the NetFlow log to the point that we make an adversary unable to distinguish between two web sessions with the same web site and two web sessions with web sites that have similar web pages in size, but not as much as to destroy all the useful information a log may provide. More specifically, we intend to preserve metrics such as arithmetic averages and standard deviation, as well as other descriptive statistics. On the other hand, we wish to obfuscate inferential statistics, so that an adversary would be unable to reach conclusions that extend beyond the immediate data alone.

For clarity, we are going to use the flow size NetFlow field as an example from here on.

Our method is to add to the value of the flow size field a random value. This value is chosen uniformly at random from a fixed range *[-d, d]*. One of the basic properties of our choice is it allows us to directly preserve the arithmetic average and standard deviation of the original distribution of flow size values. The parameter d may be chosen arbitrarily, but we will demonstrate the importance of an educated choice with an example. Consider, as an elementary example, three flows with sizes of *15*, *17* and *25* bytes, which repeatedly occur within a NetFlow log. Choosing d to be equal to two, this is what happens in the anonymized trace: The flows with the initial size of *15* now occur with flow sizes from thirteen to seventeen. Flows with the size of *17* bytes now occur with sizes from fifteen to nineteen. These two

groups of flows are now "mixed up"; what happens is that the confidence intervals for the random variables which represent the flow sizes of each flow are now different, and they overlap. On the other hand, that is not the case for the flow with size *25* bytes. It now occurs on the anonymized trace with values from 23 to 27. An adversary is still able to distinguish this flow from the other two with relative ease. Now we can easily conclude that a proper choice for the parameter d will have to take the entirety of the NetFlow log into consideration. This is an interesting topic for future work if random value shifting is to be used properly and gain acceptance, however we will not further explore it in the rest of this thesis. We will, however, evaluate that our assumptions hold true in Section 5.2, with experimental data.

Currently, *Anontool* performs some basic trace pre-processing when random value shifting is going to be used. It processes all packets in a trace in order to extract the information it needs to calculate d. This information is dependent on the field that random value shifting is being applied on. After the value of d is calculated and chosen according to the user's method of choice, the actual anonymization process takes place. It is possible to estimate d during the anonymization process, however, as knowledge of the whole trace is impossible to have until the whole trace has been processed, we believe the estimated value will not yield as good a result as when a trace can be preprocessed and the value of d calculated on it.

5

Evaluation

In this Section, we will evaluate the contributions of this thesis in detail. Subsections 5.1.1 and 5.1.2 analyze the benefits and shortcomings of the NetFlow and IPFIX anonymization implementation. Section 5.2 discusses in short the results from the application of the newly proposed primitives and lastly, Subsection 5.3 outlines the methodology followed when evaluating binary payload anonymization.

5.1 Evaluating NetFlow Anonymization

5.1.1 Functionality Comparison

Before proceeding with the performance evaluation of the available tools, we are going to briefly discuss the choices they present to the user who wishes

to perform anonymization on NetFlow records using each of these tools. In our discussion, we will include the NFDUMP, CANINE and FLAIM tools, as they are the only ones in the bibliography that support anonymization of NetFlow data in any form of logs.

Before proceeding, it is essential to define the notion of "flexibility" in the context of the anonymization process. As mentioned earlier, AAPI was developed by having flexibility in mind. What this means, is that we feel a potential user, who wishes to perform anonymization on any kind of network data, should have the potential to do so in any way she deems fit. As different organizations, institutions and different groups of people like researchers or network engineers tend to have different views and interests over network data, it is most likely that they would wish to "hide" or obfuscate different aspects of their owned network data traces or logs. Therefore, providing them with the ability to do so, is a very important factor an anonymization tool developer should bear in mind. We therefore believe that the maximum degree of flexibility in anonymization policy definitions is when the user has complete control over what primitives she can use over all of the data. This is ensured because *Anontool* operates on the granularity of protocol fields, and this is the most fine-grained choice a user can have.

Moreover, supporting packet traces as a source of network data is important for two reasons. Firstly, the information in packet traces is complete and does not bear any information loss over logs. During our work with anonymization tools we came along with log formats which, in order to achieve storage and computation efficiency, discarded certain packet contents; we feel this should not be imposed implicitly on a user. Secondly, anonymized packet traces can be further processed by tools meant for accounting, intrusion detection or other tools such as NFDUMP which operate on packet traces, without the need for another application that would reconstruct a packet trace from logs. We feel that this is another factor that gives a user

the maximum degree of choice between all the different protocol fields a packet may contain, and this contributes to achieving maximum flexibility as defined in the previous paragraph.

NFDUMP provides the user with the simplest and most rigid anonymization policy of the three tools; prefix-preserving anonymization of all the source and destination IP addresses inside the log file. Remember that this is due to the integration of the Crypto-PAn tool in the *nfdump* application. Regarding the supported formats, NFDUMP handles the collection of NetFlow export packets versions 5 and 7, as well as the newest version 9. The log files it stores, however, are not in the packet export format Cisco has defined. The single user-configurable parameter in this setup is the choice of the key used for the cryptographic algorithm which Crypto-PAn implements. While it may prove useful for specialized applications, NFDUMP offers no flexibility when a user wants to consider alternative anonymization policies.

FLAIM offers support for NetFlow versions 5 and 7. Although its modular nature should make adding support for new protocols or log formats easy, at the moment of this writing, it does not support NFDUMP version 1.5 logs, and therefore cannot process NetFlow v9 records. Note that, when it comes to NetFlow anonymization, FLAIM also operates on NFDUMP log files, and not on the Cisco packet export format. Nevertheless, FLAIM presents the user with choice between all of the fields a NetFlow record contains. The user may then choose the desired primitive to be applied on any field of each record, through the use of XML-based documents which describe her anonymization policy. FLAIM has a wide variety of anonymization primitives for the user to choose; wiping field values clean (Black Marker primitive), truncating fields, several types of permutation of a field, hashing, partitioning and a specialized partitioning for time-based fields called Time Unit Annihilation, and enumeration. While a lot in themselves, FLAIM imposes certain

restrictions on the algorithms a user can select to apply on each field. For instance, only the BinaryBlackMarker and Annihilation primitives are valid to apply on the Packets field of a NetFlow structure. It is worth noting, the FLAIM user can change the module schema in order to lift those restrictions, but at the same time she is advised not to do so. We feel only experienced users with FLAIM and XML would be able to perform such changes; such assumptions about a user or anonymization policies should, in our opinion, be avoided. Although it may not seem important, it should essentially be up to the user to decide the optimal anonymization policy to apply in each case, which could certainly vary from sharing of network activity logs, to obfuscating certain parameters of the network which could be inferred from the log, if not anonymized properly.

CANINE supports different kinds of NetFlow formats. Among them, the NetFlow v5 and v7, the NFDUMP format, and two NCSA internal formats derived from them. It can anonymize IP addresses, port and protocol numbers, timestamps and the byte counter on each flow record. The algorithms supported on each field resemble closely the ones used by FLAIM; truncation, random permutations, and prefix-preserving anonymization. For the timestamp, it can annihilate certain parts of it, perform random time shifts, or perform an enumeration. There's also a bilateral classification algorithm available for port numbers. Unfortunately, CANINE was considered non-extensible and difficult to script from the command line, so its developers proceeded with the definition and implementation of FLAIM. Due to these factors, but also because FLAIM is a later tool which addresses these difficulties, we will also not consider CANINE in our performance comparison, as it was indeed quite difficult to evaluate its behavior.

Anontool preserves the basic principle of AAPI, which is bent on being generic and flexible. It offers support for NetFlow version 5, which is the most used version supported on routers, and NetFlow version 9, the latest

addition to the series, which has an extensible design and is currently the IETF standard for information export. We chose not to implement support for NetFlow v7, because its a specialized enhancement which is incompatible with the majority of Cisco routers, and therefore not quite popular. As an application based on AAPI, the *Anontool* user has complete control over every field which may be present in a NetFlow packet. We have already mentioned in Section 4.2 the available choices of fields a user has, and there are no restrictions regarding the operations which a user may apply on them. Regarding the anonymization operations a user can apply, *Anontool* offers a wide variety of primitives to choose from. Starting with the simplest deletion of a field value, or setting it to a fixed value, a user can also choose mapping a field's values to new ones, which may or may not follow a probability distribution, she can strip certain parts of a field, or replace them with a specified value (binary or string). The popular prefix-preserving algorithms are also supported, and so are various hash functions, cryptographic and not. Also, the user can set fields according to a pattern, and specify regular expressions to match and change a part or whole of a field. This last feature is particularly useful when a user would want to eliminate potentially sensitive information which could appear on the packets of an HTTP transaction, such as part of a URL being requested by a browser.

At this point, we believe that having discussed the capabilities of each tool, *Anontool* presents a user with the maximum amount of flexibility, offering complete control over the NetFlow packet export structure. FLAIM also offers a significant amount of choices to the user, yet it places restrictions which a user may find limiting. NFDUMP offers the least capabilities of the three tools. Also, we argue that since *Anontool* operates on packets using libpcap, its output can be used as input to other tools for network management, monitoring, or accounting, and thus it can be used in conjunction with other tools, including FLAIM and NFDUMP. This is not the

case with the NFDUMP log format, unless there are specialized converters which perform this task. Yet the process of conversion takes time and makes the whole process tedious and prone to error.

5.1.2 Performance Analysis

Nowadays, NetFlow data are being used for security purposes and anomaly detection([34, 51, 68]). In the field of computer security, high performance and timely response to threats are of paramount importance. Therefore, if anonymized network flow data are to be used and shared for security purposes, we should explore how fast the anonymization process can be completed. Moreover, in the case the user wants to anonymize live traffic on the fly, it is important to explore *Anontool* 's capacity for doing so.

In this section, we present a performance evaluation of the tools available with NetFlow anonymization capabilities, referenced and described in Section 5.1.1. In order to perform the performance evaluation we used a real traffic trace collected from a monitoring sensor located at the University of Crete. The trace was collected from 26/03/2008 morning through 27/03/2008 afternoon and contains 7328264 flows presenting total traffic of 94.1 GB. The trace itself was 857 MB large.

To perform our evaluation, we used the most recent versions of the tools available; *1.0* for *Anontool* , *1.5.6* for NFDUMP, and *0.7.0* for FLAIM. All the figures we present are means calculated over 20 iterations. The test machine was an Intel(R) Pentium(R) 4 CPU clocked at 2.53GHz, with 1 GB of RAM, running a Kubuntu OS with Linux kernel version 2.6.24.

Since both NFDUMP and FLAIM require the collection of NetFlow data from the network before the actual anonymization process can take place, we used the nfcapd daemon, supplied with the NFDUMP tools, to convert the trace into the NFDUMP format and calculated the sum of user and system time needed for the conversion in the total time needed for the trace anonymization. As this collection/conversion process is required for log

processing by the aforementioned tools, we feel this extra cost should be taken into consideration, as the result is a very close approximation of a "direct" comparison.

In this point we would like to argue, once more, that sharing network level traces is more useful than sharing logs, since the user can use the traces for several purposes; i.e. she can translate the trace into flows in the format that is more suitable for the analysis she wants to perform, or she can use the anonymized network trace in order to evaluate NetFlow collection or translation tools.

In our first experiment we choose to follow the NFDUMP anonymization policy, which we implemented both with AAPI and FLAIM. NFDUMP deploys prefix preserving anonymization only in the flow source and destination IP addresses. Figure 5.1 shows the sum of system and user time taken for a single trace to be anonymized by the three tools, and Figure 5.2 shows the average CPU load during anonymization. We note that all three tools use a high amount of CPU time, which is easily explained because prefix-preserving anonymization, and the AES cipher in particular, involves a lot of expensive cryptographic operations. As we can see the performance of our implementation and NFDUMP is similar but our tool has the ability of deploying anonymization in all fields of the Netflow implementation. FLAIM presents 5 times worse performance, which is attributed to high memory consumption which led to excessive swapping operations (as indicated by the *vmstat* tool). This performance problem appears in FLAIM's current release (0.7.0) at the time of this writing as well as a previous release we tested (0.5.2), and the developers were made aware of this behaviour.

In our second experiment we choose to implement a different anonymization policy. We zero both source and destination IP addresses in all Netflow records. Since NFdump has a single anonymization policy we can compare only with FLAIM. As the results from figures 5.3 and 5.4 indicate,

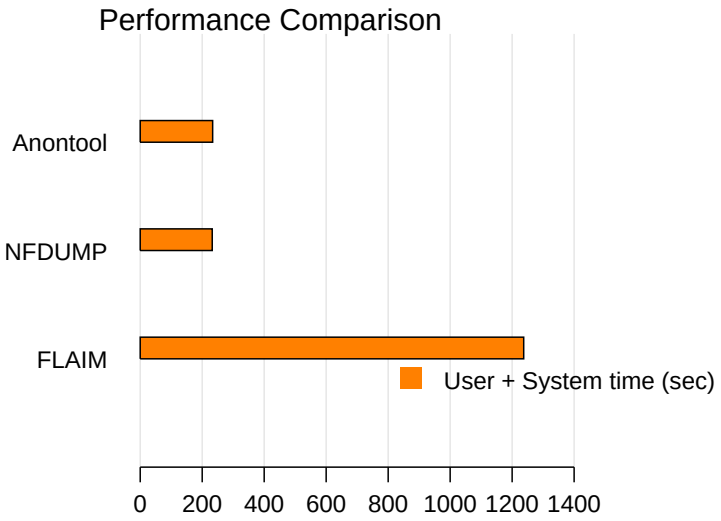


FIGURE 5.1: Performance comparison (user & system time) deploying prefix-preserving IP address anonymization

our implementation is again one order of magnitude faster than FLAIM and also requires half the CPU utilization that FLAIM does. This shows that our tool would be able to anonymize NetFlow datagrams on the fly without any loss even in high bandwidth rates (the 20 seconds needed by *Anontool* are translated to approximately 350 Mbps throughput), while other tools would require capturing the data first and then follow the anonymization procedure.

Finally, we measured the time it took *Anontool* and FLAIM to execute the predefined anonymization policy we saw at the end of Section 4.2. This policy instructs that source and destination IP addresses are set to zero, the TCP port fields are set to a random value and the Uptime field is also set to a random value. Again, we note that NFdump cannot perform these operations as it only applies prefix-preserving anonymization of IP addresses, it is therefore not included in the results. The results appear in figures 5.6

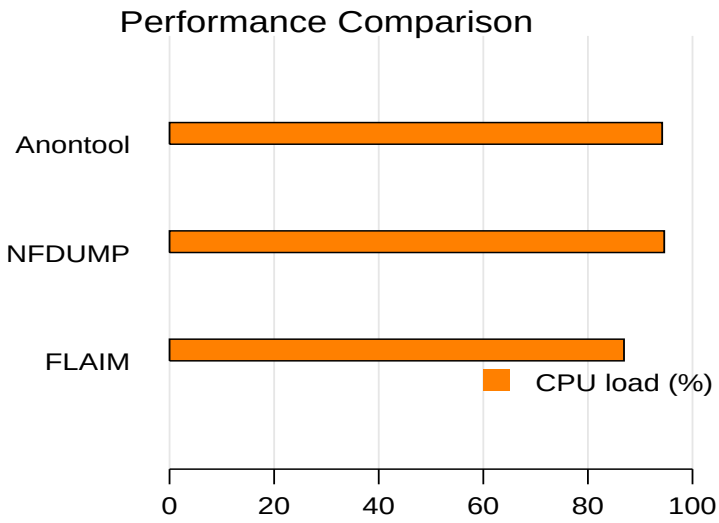


FIGURE 5.2: Performance comparison (CPU load) deploying prefix-preserving IP address anonymization

and 5.5, and they are consistent with our previous experimental results.

One might think this comparison with respect to the results on FLAIM's poor performance shows nothing but the effect of swapping operations on application performance, but this is not the case; it stresses the need to stop focusing on providing a plethora of different tools for performing anonymization and focus research efforts on anonymization policies and the open theoretical issues about tradeoff quantification, etc. Spending time and effort in rediscovering the wheel and solving bugs which other implementations lack is a waste; it would be more productive to unify the various implementations into one and support it. This is still a long way ahead, though.

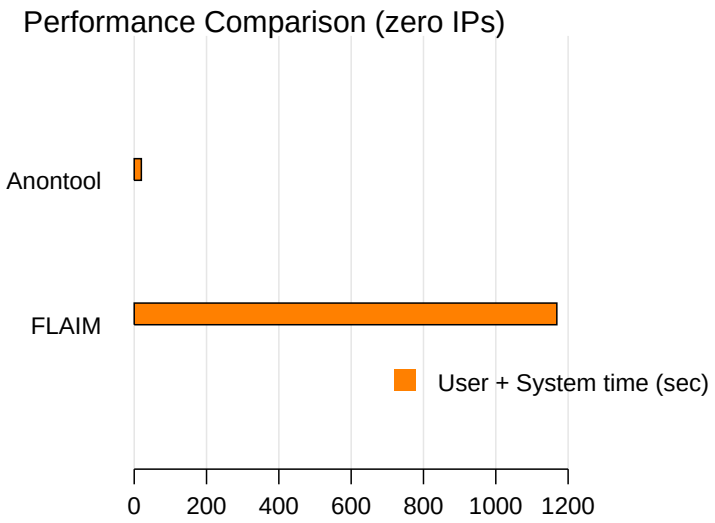


FIGURE 5.3: Performance comparison (user & system time) deploying zero IP address anonymization

5.2 Evaluating New Primitives

While describing our motivation behind the design and implementation of bidirectional mapping, we also presented our evaluation of it. Bidirectional mapping is straightforward enough to evaluate - a simple verification stage for its operation is enough, in order to confirm it works as expected.

To verify our assumptions about the descriptive statistics of a NetFlow log being preserved after the application of random value shifting, we implemented it in *Anontool* and proceeded to process a NetFlow packet trace with it. Our choice for the parameter d was the minimum flow size observed, divided by 2. As we previously mentioned, this is most likely not a good choice for real world applications, but it is good enough for the experimental evaluation we describe. For a given packet trace, we computed its cumulative statistics and then used them to compute the value for d; we underline

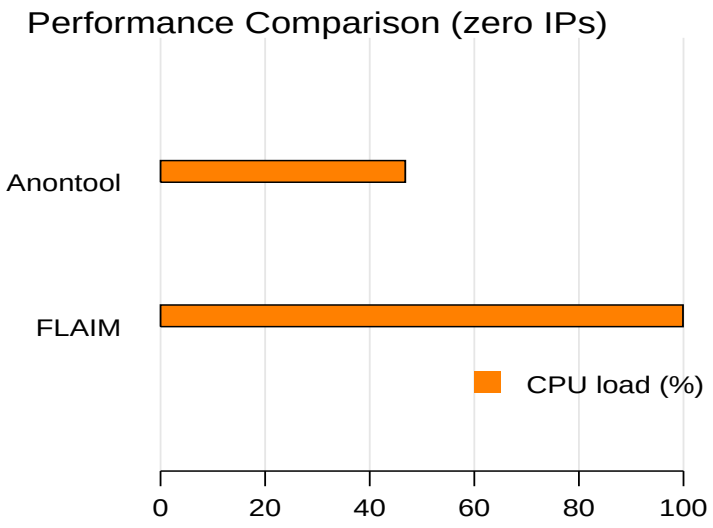


FIGURE 5.4: Performance comparison (CPU load) deploying zero IP address anonymization

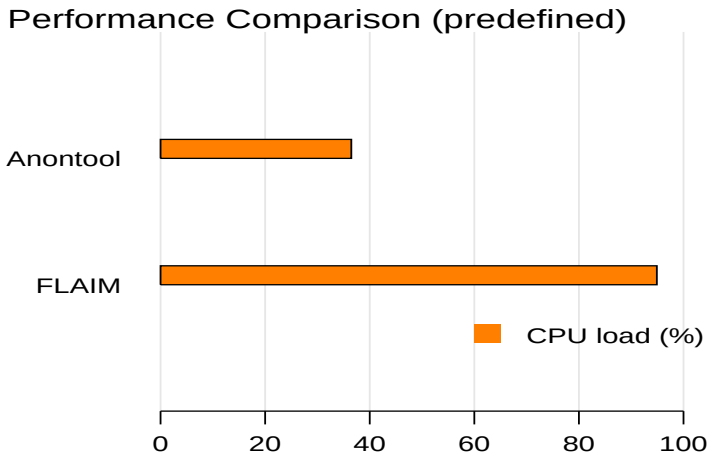


FIGURE 5.5: Performance comparison (CPU load) deploying a predefined policy (zero source and destination IP addresses, random TCP port numbers and random Uptime)

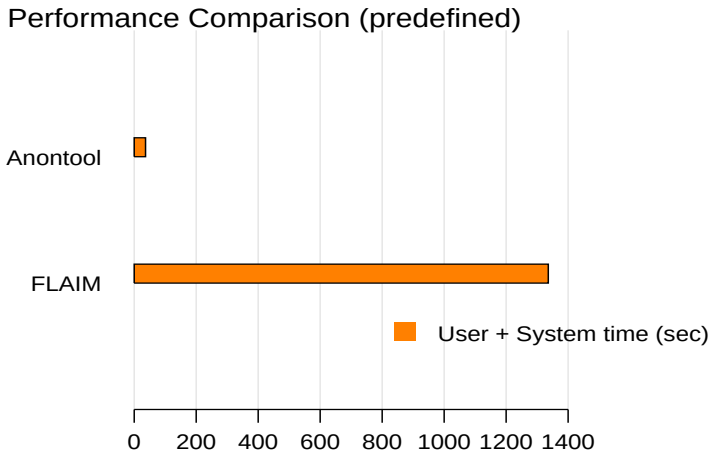


FIGURE 5.6: Performance comparison (user & system time) deploying a predefined policy (zero source and destination IP addresses, random TCP port numbers and random Uptime)

here once again that the choice for d opens a wide topic for discussion in itself, and real world applications should consider it in great length. The value for d specifies how certain values in the original trace will overlap with the others, therefore "scrambling" the value range and making it harder for the attacker to distinguish between them.

We then calculated the arithmetic average and the standard deviation for both the original and the anonymized trace, which we present in the first row of table 5.1. The NetFlow trace spanned a time period of three minutes and a bit more than 150.000 bytes transferred. Given the above choice of d, we subsequently applied random value shifting to our packet trace and verified that cumulative statistics were indeed largely preserved, with a difference of 0.072%. Table 5.1 presents the values calculated for both traces. We can see that the average and standard deviation do not largely differ. This supports our initial hypothesis, that we can preserve

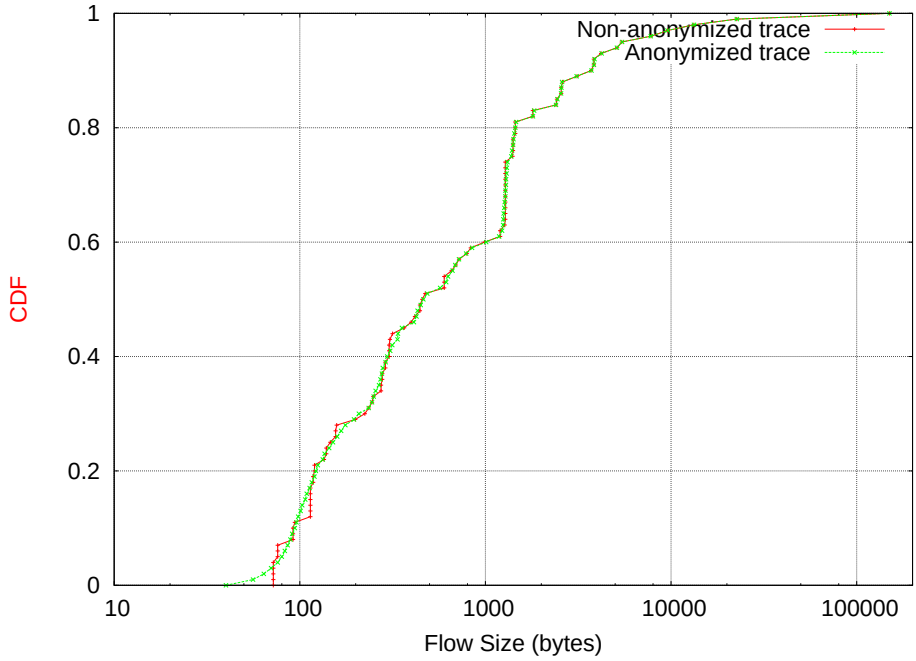


FIGURE 5.7: Cumulative distribution function of flow sizes

some amount of general information about NetFlows in the trace even after performing random value shifting. In the case the reader is wondering on the use of such information like basic cumulative statistics, we should note that it is of great importance to network administrators, which take interest in basic traffic patterns in their networks without necessarily wondering about the specifics of each flow.

Trace	**Average Flow Size (bytes)**	**Std. Deviation**
Original	1843.69	7336
Anonymized	1845.02	7335.52

TABLE 5.1: Some basic descriptive statistics regarding a NetFlow trace before, and after anonymization.

Figure 5.7 presents the cumulative distribution function of the distribution of flow sizes in the original, and the anonymized traces, for comparison and reference. As we can see the distribution remains almost identical after the anonymization process. This enforces our initial argument that the information that can be derived by the anonymization process are not affected by the value scrambling.

For such a value of d, which was not chosen with the purpose of mixing flow sizes in mind, it is not useful to examine each flow individually and attempt to identify it in the resulting trace after anonymization. This should be the object of future work, as it is being done with existing anonymization primitives [55] and the anonymization process in general [70], [53].

5.3 Evaluation of Binary Payload Anonymization

In order to test our implementation against a set of packet traces containing malicious payloads, we obtained using the prototype implementation of the Network-level Emulation attack detection method formally described in [43], [60], which identifies the presence of self-modifying polymorphic shellcode in network streams. The alerts generated contain full payload traces in libpcap format. Each trace corresponds to a single attack attempt and contains all packets of the network flow (quintuple) of the particular attack instance, including the initial TCP 3-way handshake.

In total, the number of attack traces generated by NEMU was 21726, spanning a time period from January 11, 2007 to April 6, 2008. Anontool detected and anonymized sensitive data within 17036 of those traces, a 78.4% of the total number of alert-generating traces.

For verification purposes, we manually checked and inspected some of the attack traces, to determine two things: firstly, whether we anonymized the bytes at the correct offset(s), and secondly to determine whether exploits that weren't anonymized did not contain any sensitive data or we just

```
01f0  1f ef 89 fd 79 20 88 90  9f 99 88 88 88 14 1a 13   ....y .. ........
0200  57 58 14 57 12 14 1f 18  57 18 07 12 19 57 00 00   WX.W.... W....W..
0210  00 00 00 00 00 00 00 00  00 00 00 00 57 45 40 42   ........ ....WE@B
0220  42 57 49 57 1e 51 12 14  1f 18 57 02 04 12 05 57   BWIW.Q.. ..W....W
0230  46 57 46 57 49 49 57 1e  57 51 12 14 1f 18 57 10   FWFWIIW. WQ....W.
0240  12 03 57 45 59 12 0f 12  57 49 49 57 1e 57 51 12   ..WEY... WIIW.WQ.
0250  14 1f 18 57 06 02 1e 03  57 49 49 57 1e 57 51 11   ...W.... WIIW.WQ.
0260  03 07 57 5a 19 57 5a 04  4d 1e 57 51 45 59 12 0f   ..WZ.WZ. M.WQEY..
0270  12 7a 7d 77 90 90 90 90  90 90 90 90 90 90 90 90   .z}w.... ........
0280  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90   ........ ........
```

FIGURE 5.8: The anonymized Wuerzburg shellcode as the final output of *Anontool* .

happened to lack a regular expression for the corresponding decoder.

For the first case, we chose a random trace which *Anontool* reported it anonymized an IP within it, and tried to determine whether the IP was correctly anonymized. Figure 5.8 shows the anonymized output for the example explained in Section 4.3, taken from the resulted trace of *Anontool* . Our tool managed to find and anonymize the sensitive information contained in the encrypted payload. The outlined bytes show the offset at which the IP address was identified and subsequently masked. We therefore were able to verify the anonymization process was correct.

We then proceeded to examine in detail a few attack traces which *Anontool* did not anonymize. Among them we discovered an remote root exploit for the Knox Arkiea Server (*arkiead*) [14] which did not connect back to any hosts and additionally its payload did not contain any sensitive information. By choice, we do not include and regular expressions for this kind of exploits inside our tool. It is by definition they lack the sensitive information that may expose an innocent host to attackers, which means they can be shared without that risk in mind.

Our search for remote exploits which used connect-back methods or sub-

sequently downloaded another binary did not prove fruitful. That, however, does not by any means prove there are no such exploits out there. The user needs to be aware that regular expressions within the tool do not constitute a panacea. On the other hand, given the fact that we designed and implemented our tool with modularity and extensibility in mind, adding support for a new kind of shellcode/binary payload is easy and intuitive. We expect that, in order for our tool to be widely deployed and used, this is a necessary quality, especially since we cannot predict future advances in the area of polymorphic and metamorphic shellcode construction. Experts in that field, however, should be given the convenience of easily implementing a code module for our tool to anonymize their traces of choice.

5.4 Availability

Stable versions of *Anontool* are made available on [3], and is mirrored from the Community Resource for Archiving Wireless Data (CRAWDAD [7]). It has already been downloaded from at least 98 different users since it was first mirrored from the CRAWDAD download page, and we have received constructive input and debugging information which has aided us in debugging, profiling and further developing *Anontool* , to maintain a high level of code quality, performance and usability. We have also ported *Anontool* to the OpenBSD [29] and Mac OS X [16] operating systems.

6

Conclusions and Future Work

This thesis provided a detailed overview of some open problems regarding network data anonymization. Based on a widely accepted framework technology, *Anontool* provides a complete, flexible and efficient basis for performing anonymization on packet traces. Moreover, the architecture and implementation of *Anontool* was designed to be extensible in order to accomodate future needs. *Anontool* aims to provide a standard codebase to meet the privacy needs of parties sharing network traces.

We described the implementation of *Anontool* , which is built on top of the AAPI framework, while focusing on our three major contributions. Firstly, the implementation of modules to support anonymization on NetFlow (versions 5 & 9) and IPFIX protocols. *Anontool* gives the user com-

plete control over every field of flow datagrams made available by an information export node, without imposing limitations of other tools. Furthermore, our results have shown that *Anontool* is a lot faster than other tools with similar flexibility, and on par with very specialized and limited approaches.

Continuing with NetFlow traces, we described two attacks on existing trace anonymization schemes which cannot be resolved with the primitives already implemented in popular tools, and provided countermeasures for each of them, namely bidirectional mapping and random value shifting which preserves cumulative statistics about a trace while obfuscating inferential statistics. Lastly, we designed and implemented a proof-of-concept mechanism for identifying and anonymizing executable payloads, with emphasis on malicious payloads such as shellcodes and remote exploits and the polymorphic and metamorphic components of them. The mechanism has still a lot of potential for improvement and extensibility but still is a significant step towards sharing attack traces for the benefit of the computer security community.

Anontool is under deployment and new features are constantly being added and refined. It has the privilege of enjoying usage in real world scenarios, a process which has also helped a lot with debugging and making it more stable and faster. Overall, the traits that make *Anontool* popular, extensible and fast are the same traits that provide its users with lots of future capabilities to support new network protocols and policy-related implementations for verification and evaluation on the various metrics of anonymization, which is still an area with many open problems and lots of opportunities for novel research.

Bibliography

[1] Amun Honeypot: breakdown of the Wuerzburg shellcode. "http://zero.ram.rwth-aachen.de/amun/shellcode.php?id=1".

[2] Amun: Python honeypot. "http://zero.ram.rwth-aachen.de/amun/shellcodes.php".

[3] Anontool Download Page. http://www.ics.forth.gr/dcs/Activities/Projects/anontool.html.

[4] Audit Record Generation and Utilization System (Argus). http://www.qosient.com/argus/index.htm.

[5] CAIDA - COMMONS Project. http://www.caida.org/projects/commons/.

[6] CAIDA - Cooperative Association for Internet Data Analysis. http://www.caida.org.

[7] Community Resource for Archiving Wireless Data (CRAWDAD). http://crawdad.cs.dartmouth.edu/.

[8] drand48() Linux Manual Pages. http://www.manpagez.com/man/3/drand48/.

[9] European Network of Affined Honeypots. "http://www.fp6-noah.org/".

[10] Internet World Stats. http://www.internetworldstats.com/stats.htm.

[11] IP Flow Information Export (IPFIX). "http://www.ietf.org/html.charters/ipfix-charter.html".

[12] It industry's 12-point cyber-security plan. http://www.vnunet.com/vnunet/news/2126395/industry-point-cyber-security-plan.

[13] K2, admmutate. "http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz.".

[14] Knox Arkiea Server Backup local/root remote exploit. "http://archives.neohapsis.com/archives/fulldisclosure/2005-02/att-0397/arksink2.c".

[15] Log Anonymization and Information Management Working Group (LAIM). http://laim.ncsa.uiuc.edu/.

[16] Mac OS X Homepage. http://www.apple.com/macosx/.

[17] MAPI official homepage. "http://mapi.uninett.no/".

[18] National Laboratory for Applied Network Research (NLANR) Project. http://www.nlanr.net.

[19] Network traces of attacks captured at various LOBSTER passive monitoring sensors. "http://lobster.ics.forth.gr/traces/".

[20] Nfdump tools collection. http://nfdump.sourceforge.net/.

[21] Official Nepenthes website. http://nepenthes.mwcollect.org/contact".

[22] Perl Compatible Regular Expressions (PCRE) library. "http://www.pcre.org/".

[23] Protected Repository for the Defense of Infrastructure Against Cyber Threats (PREDICT). https://www.predict.org/.

[24] Sourcefire VRT Certified Rules. "http://www.snort.org/pub-bin/downloads.cgi".

[25] Tcpdump/libpcap official site. "http://www.tcpdump.org".

[26] The HoneyNet Project. "http://www.honeynet.org/".

[27] The Internet Measurement Data Catalog (DatCat). http://imdc.datcat.org/help/general.

[28] The National Strategy to Secure Cyberspace. `http://www.whitehouse.gov/pcipb/`.

[29] The OpenBSD Operating System. `http://www.openbsd.org/`.

[30] The SANS (SysAdmin, Audit, Network, Security) Institute. `http://www.sans.org/`.

[31] Ten Things Lawyers Should Know About Internet Research, August 2008. `http://www.caida.org/publications/papers/2008/lawyers_top_ten/`.

[32] J. W. A. Slagell and W. Yurcik. Network Log Anonymization: Application of Crypto-PAn to Cisco Netflows. *NSF/AFRL Workshop on Secure Knowledge Management (SKM)*, 2004.

[33] A.Meyerson and R.Williams. General k-anonymization is hard. Carnegie Mellon School of Computer Science Tech Report,2003; 03-113.

[34] P. Barford and D. Plonka. Characteristics of Network Traffic Flow Anomalies, 2001. `"citeseer.ist.psu.edu/barford01characteristics.html"`.

[35] Chai Wah Wu. Privacy Preserving Data Mining: A Signal Processing Perspective and a Simple Data Perturbation Protocol. In *In Proc. of the IEEE ICDM Workshop on Privacy Preserving Data Mining*, pages 10–17, 2003.

[36] Cisco Systems, Inc. Netflow Specification. `"http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml"`.

[37] College of Computing, Georgia Tech. Cryptography-based Prefix-preserving Anonymization. `"http://www.cc.gatech.edu/computing/Telecomm/cryptopan"`.

[38] S. Coull, M. Collins, C. Wright, F. Monrose, and M. Reiter. On Web Browsing Privacy in Anonymized NetFlows. In *16th USENIX Security Symposium*, pages 339–352, 2007.

[39] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS '01: Proceedings of*

the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 247–255, New York, NY, USA, 2001. ACM.

[40] Dave Plonka. ip2anonip. "http://dave.plonka.us/ip2anonip".

[41] A. Desjanun. Hacker Teams Breach Powerful Research Networks, 2004. http://www.usatoday.com/tech/news/computersecurity/2004-04-14-synchronized-hacking_x.htm.

[42] Eddie Kohler. ipsumdump. "http://www.cs.ucla.edu/~kohler/ipsumdump".

[43] M. P. et al. Emulation-based Detection of Non-self-contained Polymorphic Shellcode. In *Proceedings of RAID'07*, 2007.

[44] A. Ghose and K. Hausken. A Strategic Analysis of Information Sharing Among Cyber Attackers. In *SSRN Working Paper Series*, 2006. http://ssrn.com/abstract=928138.

[45] Greg Minshall. Tcpdpriv. "http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html".

[46] Jian Xu et al. Utility-Based Anonymization for Privacy Preservation with Less Information Loss. In *12th ACM SIGKDD*, 2006.

[47] K. Wang and B. C. M. Fung and P. S. Yu. Handicapping Attacker's Confidence: An Alternative to k-Anonymization. *Knowledge and Information Systems: An International Journal (KAIS)*, 2006.

[48] D. Koukis, S. Antonatos, and K. G. Anagnostakis. On the Privacy Risks of Publishing Anonymized IP Network Traces. In *Communications and Multimedia Security*, pages 22–32, 2006.

[49] D. Koukis, S. Antonatos, D. Antoniades, P. Trimintzios, and E. Markatos. A Generic Anonymization Framework for Network Traffic. In *Proceedings of the IEEE International Conference on Communications (ICC 2006)*, June 2006.

[50] C. Kreibich. NetDuDe: Network Dump data Displayer and Editor. "http://netdude.sourceforge.net".

[51] A. Lakhina, M. Crovella, and C. Diot. Characterization of Network-Wide Anomalies in Traffic Flows, 2004. `"http://citeseer.ist.psu.edu/715839.html"`.

[52] K. Lakkaraju and A. Slagell. Evaluating the Utility of Anonymized Network Traces for Intrusion Detection. *ArXiv e-prints*, 712, Dec. 2007.

[53] K. Lakkaraju and A. J. Slagell. Evaluating the Utility of Single Field Anonymization Polices by the IDS Metric : Towards measuring the trade off between Utility and Security. In *Proceedings of the 4th Annual SECURECOMM Conference*, September 2008.

[54] Y. Li, A. Slagell, K. Luo, and W. Yurcik. CANINE: A Combined Converter and Anonymizer Tool for Processing NetFlows for Security. In *Proceedings of the International Conference on Telecommunication Systems - Modeling and Analysis (ICTSM)*, Nov. 2005.

[55] M. Burkhart, D. Brauckhoff, M. May and Elisa Boschi. The Risk-Utility Tradeoff for IP Address Truncation. In *1st ACM Workshop on Network Data Anonymization (NDA 2008)*, October 2008.

[56] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 24, 2006.

[57] R. Pang, M. Allman, V. Paxson, and J. Lee. The Devil and Packet Trace Anonymization. *ACM Computer Communication Review*, 36(1):29–38, Jan. 2006.

[58] R. Pang and V. Paxson. A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In *Proceedings of the ACM SIGCOMM Conference*, August 2003.

[59] M. Peuhkuri. A Method to Compress and Anonymize Packet Traces. *Internet Measurement Workshop (San Francisco, California, USA: 2001)*, pages 257–261, 2001.

[60] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. Network-level Polymorphic Shellcode Detection using Emulation. In *Proceedings of DIMVA'06*, 2006.

[61] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.

[62] P.Szor. Hunting for Metamorphic. In Proceedings of the Virus Bulletin Conference, 2001.

[63] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley. Analyzing Privacy in Enterprise Packet Trace Anonymization. In *Proceedings of NDSS 2008*, February 2008.

[64] Roberto J. Bayardo and Rakesh Agrawal. Data Privacy through Optimal k-Anonymization. *Data Engineering, International Conference on*, 0:217–228, 2005.

[65] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. November 1999. (available from *http://www.snort.org/*).

[66] A. J. Slagell, K. Lakkaraju, and K. Luo. FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs. In *LISA*, pages 63–77, 2006.

[67] L. Sweeney. k-anonymity: a Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[68] T. T.Dbendorfer, A.Wagner and B.Plattner. Flow-Level Traffic Analysis of teh Blaster and Sobig Worm Outbreaks in an Internet Backbone. In *Proceedings of DIMVA 2005, Springer's Lecture Notes in Computer Science*, 2005.

[69] Y. T.Detristan, T.Ulenspiegel and M.V.Underduk. Polymorphic Shellcode Engine Using Spectrum Analysis. Phrack magazine, `"http://www.phrack.org/issues.html?issue=61\&id=9#article"`.

[70] W. Yurcik et al. SCRUB-tcpdump: A Multi-Level Packet Anonymizer Demonstrating Privacy/Analysis Tradeoffs. In *Proceedings of SECOVAL '07*, 2007.

[71] J. Xu, J. Fan, M. Ammar, and S. Moon. Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme. *ICNP 2002*, 2002.

[72] J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization. *Internet Measurement Workshop (San Francisco, CA, USA: 2001)*, pages 263–266, 2001. "citeseer.nj.nec.com/462352.html".

[73] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu. Utility-based anonymization for privacy preservation with less information loss. *SIGKDD Explor. Newsl.*, 8(2):21–30, 2006.

[74] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102, New York, NY, USA, 2006. ACM.

[75] C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis, 2002.