UNIVERSITY OF CRETE
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCES AND ENGINEERING

Identification of Events on Encrypted Network Traffic and Characterization of Malicious Servers on the Internet

by

Eva Papadogiannaki

PhD Dissertation

Presented

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Heraklion, July 2023

UNIVERSITY OF CRETE

DEPARTMENT OF COMPUTER SCIENCE

Identification of Events on Encrypted Network Traffic and Characterization of Malicious Servers on the Internet

PhD Dissertation Presented

by **Eva Papadogiannaki**

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

APPROVED BY:

Author: Eva Papadogiannaki

Supervisor: Sotiris Ioannidis, Associate Professor at Technical University of Crete

---

Committee Member: Xenofontas Dimitropoulos, Associate Professor at University of Crete

---

Committee Member: Polyvios Pratikakis, Assistant Professor at University of Crete

---

Committee Member: Maria Papadopouli, Professor at University of Crete

---

Committee Member: Panagiota Fatourou, Professor at University of Crete

---

Committee Member: Elias Athanasopoulos, Associate Professor at University of Cyprus

---

Committee Member: Michalis Polychronakis, Associate Professor at Stony Brook University

---

Department Chairman: Antonis Argyros, Professor at University of Crete

Heraklion, July 2023

To my grandma and all the girls that are banned from their right to education

Acknowledgments

I would like to dedicate some lines to acknowledge the people that contributed in this long and challenging doctoral journey.

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Sotiris Ioannidis for his continuous guidance, encouragement and support. Over the years, he helped me accumulate a wealth of knowledge and experience. I am deeply thankful to the members of my PhD advisory committee, Prof. Xenofontas Dimitropoulos and Prof. Polyvios Pratikakis, for their invaluable insights and critical feedback. Also, I would like to extend my sincere appreciation to the examination committee members, Prof. Maria Papadopouli, Prof. Panagiota Fatourou, Prof. Elias Athanasopoulos and Prof. Michalis Polychronakis, for their thoughtful comments and rigorous examination of this thesis.

In addition, I would like to thank the Niometrics team and especially Dr. Periklis Akritidis, Dr. Konstantinos Chalevidis and Lazaros Koromilas, for our influential collaboration during my internships. I would like to express my sincere gratitude to all my present and former colleagues, the Parasecurity team members, and especially, Christos Papachristos, Manos Athanatos, Despina Kopanaki and Maria Mastoraki. Special thanks to Dimitris Deyannis, George Christou, Kostas Solomos and Dimitris Karnikis. I should not omit to extend my appreciation to the MSc students I have had the privilege of co-supervising, Giannis Giakoumakis and Andreas Theofanous.

Finally, I would like to acknowledge all the people that helped me preserve my mental equilibrium. I am deeply thankful to my family Michalis, Alexia and Sifis, for their unwavering pride and support. I feel indebted to my grandmother Anthoula that always provided me with ταπεράκια (phonetically taperákia, lunchbox). I would like to extend a special thank you to my loyal companions, Rocky and Jela, who served as a reminder to take breaks and appreciate the simple pleasures in life. I am deeply grateful for the un-

wavering support and friendship of Georgianna, Eirini, Xenia, Maria, Ioanna, Elina and Rafaella. Last but not least, I want to thank Antonis, for his unwavering belief in me that was a constant source of strength, throughout the ups and downs of my academic pursuit.

As a final remark, I want to acknowledge that this work has been performed at the Computer Science Department at University of Crete, the Foundation for Research and Technology - Hellas (FORTH), the Technical University of Crete (TUC), and Niometrics. The work was supported by the projects AI4HealthSec, CyberSANE, CyberSURE, I-BIDAAS, Ideal-Cities, and SENTINEL, funded by the European Commission under Grant Agreements No. 883273, No. 833683, No. 734815 , No. 780787, No. 778229 , and No. 101021659, respectively.

Abstract

The growing adoption of network encryption protocols, like TLS, has altered the scene of network traffic monitoring. With the advent and rapid increase in network encryption mechanisms, typical deep packet inspection systems that monitor network packet payload contents are gradually becoming obsolete, while in the meantime, adversaries abuse the utilization of the TLS protocol to bypass them.

In this work, we propose a pattern language to describe packet sequences for the purpose of fine-grained identification of events even in encrypted network traffic. The first use case for our pattern language is the identification of application-level events in encrypted network traffic. We demonstrate its expressiveness with case studies for distinguishing messaging, voice, and video events in Facebook, Skype, Viber, and WhatsApp network traffic. The second use case for our pattern language is the identification of intrusions and suspicious events in encrypted network traffic. Similarly, we investigate its expressiveness with case studies for distinguishing events originating from penetration tools, such as password cracking, or botnet communications. We provide an efficient implementation for the proposed pattern language, which we integrate into two different DPI systems. We evaluate the proposed pattern language with respect to the level of expressiveness and the processing performance. Finally, we demonstrate that the proposed language can be mined from traffic samples automatically, minimizing the otherwise high ruleset maintenance burden.

Except for our passive analysis approach, we actively contact IP addresses known to participate in malicious activities, since we aim to understand the botnet ecosystem in the wild. We utilize an open-source tool for active probing and TLS fingerprint construction. Based on packets acquired from TLS handshakes, server fingerprints are constructed dur-

ing a time period of 7 months. The fingerprints express servers' responses to a sequence of several ‘‘`TLS Client Hello`’’ packets with different TLS attributes and we investigate if it is feasible to detect suspicious servers and re-identify other similar within blocklists with no prior knowledge of their activities. Based on our findings, we can see that fingerprints originating from suspicious servers are repetitive among similarly configured servers, while it is rare to overlap with fingerprints that correspond to legitimate domains. The findings of our measurement study encourage the utilization of actively generated TLS fingerprints for detecting malicious command and control servers in the wild.

Subsequently, we present the literature that manages to perform network traffic analysis and inspection after the ascent of encryption. We observe that the research community has already started proposing solutions on how to perform inspection even when the network traffic is encrypted and we review these works. We present the techniques and methods that these works use and their limitations.

Lastly, we do not omit to examine the countermeasures that have been proposed to circumvent traffic analysis and we discuss about our system's limitations related to traffic analysis resistance.

Keywords: Encrypted traffic analysis, Network monitoring, Packet metadata, Passive traffic inspection, Mobile applications, Intrusion detection, Active probing, TLS fingerprinting, CnC server characterization, Traffic analysis resistance

Supervisor:

Sotiris Ioannidis

Associate Professor

School of Electrical and Computer Engineering

Technical University of Crete

Περίληψη (Abstract in Greek)

Η συνεχώς αναπτυσσόμενη καθιέρωση των πρωτοκόλλων για την κρυπτογράφηση της κίνησης του δικτύου (όπως το TLS πρωτόκολλο), έχει αλλάξει τα δεδομένα στην εποπτεία του δικτύου. Με τη ραγδαία αύξηση των μηχανισμών κρυπτογράφησης, τα παραδοσιακά συστήματα για επιθεώρηση της κίνησης του δικτύου που στηρίζονται στην επεξεργασία των περιεχομένων των πακέτων, σταδιακά χάνουν την αποτελεσματικότητα τους, καθώς παράλληλα, κακόβουλοι χρήστες του δικτύου εκμεταλλεύονται την κρυπτογράφηση για να κρύψουν τις δραστηριότητες τους και να αποφύγουν την ανεύρεση της παρουσίας τους.

Σε αυτήν την εργασία, προτείνουμε μία γλώσσα προτύπων για να περιγράψουμε τα μοτίβα που υπάρχουν στις ακολουθίες πακέτων δικτύου, με σκοπό τη λεπτομερή ανίχνευση συμβάντων ακόμα και σε κίνηση δικτύου που έχει κρυπτογραφηθεί. Η πρώτη περίπτωση χρήσης που εξετάζουμε είναι η λεπτομερής ανίχνευση δραστηριότητας εφαρμογών δικτύωσης χρηστών και επικοινωνίας. Δείχνουμε πως είναι δυνατή η δημιουργία μίας τέτοιας γλώσσας που να περιγράφει με εκφραστικότητα ενέργειες όπως την ανταλλαγή μηνυμάτων και την επικοινωνία μέσω κλήσης ή βιντεοκλήσης χρησιμοποιώντας τις διαδεδομένες εφαρμογές Facebook, Skype, Viber, WhatsApp. Μία δεύτερη περίπτωση χρήσης για τη γλώσσα προτύπων που προτείνουμε είναι η ανίχνευση περιστατικών εισβολών σε συστήματα εποπτεύοντας κίνηση δικτύου που είναι κρυπτογραφημένη. Όπως και στην προηγούμενη περίπτωση, εξετάζουμε αν είναι εφικτό, και αν ναι σε τι λεπτομέρεια, να αναγνωρίσουμε τη δραστηριότητα που προέρχεται από εργαλεία διείσδυσης (penetration tools) ή επικοινωνία δικτύων προγραμμάτων ρομπότ (botnets). Παρέχουμε μία αποδοτική υλοποίηση για αυτήν την γλώσσα προτύπων, την οποία ενσωματώνουμε σε δύο διαφορετικά συστήματα επιθεώρησης κίνησης του δικτύου. Αξιολογούμε την υλοποίηση μας χρη-

σιμοποιώντας τα κριτήρια της απόδοσης τόσο σε επίπεδο ορθότητας και ακρίβειας, όσο και σε επίπεδο επιδόσεων. Τέλος, δείχνουμε πως η γλώσσα που προτείνουμε μπορεί να 'εξορυχθεί' (data mining) αυτόματα, περιορίζοντας το φόρτο εργασίας για τη διαμόρφωση και συντήρηση ενός μεγάλου συνόλου από πρότυπα.

Η διαδικασία που αναφέραμε στην παραπάνω παράγραφο, περιγράφεται από μία παθητικού τύπου εποπτεία και ανάλυση των πακέτων δικτύου. Επιπρόσθετα, χρησιμοποιήσαμε μία πιο παρεμβατική μέθοδο, έτσι ώστε να επικοινωνήσουμε με εξυπηρετητές μέσω διευθύνσεων IP που βρίσκονται διαθέσιμες σε δημόσιες λίστες με πληροφορίες για κακόβουλες δραστηριότητες. Ο σκοπός μας είναι να καταλάβουμε το οικοσύστημα των δικτύων προγραμμάτων ρομπότ (botnets). Συγκεκριμένα, χρησιμοποιούμε ένα εργαλείο ανοικτού κώδικα για να παράξουμε αποτυπώματα TLS για κάθε ένα από τους εξυπηρετητές που επικοινωνούμε. Στην έρευνα μας καταφέραμε να συλλέξουμε πληροφορίες σε ένα διάστημα 7 μηνών. Τα αποτυπώματα αυτά εκφράζουν τις απαντήσεις των εξυπηρετητών σε μία αλληλουχία από 10 ''TLS Client Hello''μηνύματα με διαφορετικά χαρακτηριστικά (π.χ. έκδοση TLS , επιλεγμένος αλγόριθμος κρυπτογράφησης κ.τ.λ.). Αυτό που θέλουμε να εξετάσουμε είναι η δυνατότητα να αναγνωρίσουμε την ιδιότητα και δραστηριότητα αυτών των εξυπηρετητών χρησιμοποιώντας μόνο αυτά τα αποτυπώματα, μέσα σε λίστες από αποτυπώματα για τα οποία δεν έχουμε πρωτύτερη γνώση. Μέσω των αποτελεσμάτων μας, βλέπουμε πως τα αποτυπώματα που υπολογίζονται, επαναλαμβάνονται μεταξύ εξυπηρετητών που συμμετέχουν στην ίδια οικογένεια δικτύων προγραμμάτων ρομπότ (botnets). Επίσης βλέπουμε πως σπάνια αλληλοεπικαλύπτονται με αποτυπώματα από εξυπηρετητές για τους οποίους είναι γνωστή η ορθή χρήση τους. Τα ευρήματα αυτά μας παροτρύνουν να χρησιμοποιήσουμε αυτή τη μεθοδολογία για να αναγνωρίζουμε και να ταυτοποιούμε εξυπηρετητές που εμπλέκονται σε κακόβουλες δραστηριότητες.

Στη συνέχεια, παρουσιάζουμε μία εκτενή βιβλιογραφική μελέτη σχετικά με τα εργαλεία για την εποπτεία του δικτύου μετά την καθιέρωση των πρωτοκόλλων κρυπτογράφησης. Αυτό που παρατηρούμε είναι πως παρόλο που η επιστημονική κοινότητα έχει προτείνει μία πληθώρα τεχνικών για το σκοπό αυτό, υπάρχουν ακόμα ελλείψεις

και μειονεκτήματα.

Τέλος, δεν παραλείπουμε να εξετάσουμε τις τεχνικές που υπάρχουν για την αποτροπή της ανάλυσης των κρυπτογραφημένων επικοινωνιών σε περιπτώσεις κακόβουλης χρήσης και λογοκρισίας. Συζητάμε αν το σύστημα μας μπορεί να ανταπεξέλθει σε τέτοιες περιπτώσεις.

Λέξεις Κλειδιά: Κρυπτογραφημένη κίνηση δικτύου, Εποπτεία δικτύου, Μεταπληροφορίες πακέτων δικτύου, Επισκόπηση δικτύου με παθητικές τεχνικές, Εφαρμογές κινητών συσκευών, Εντοπισμός εισβολών, Παρεμβατική διερεύνηση δικτύου, Αποτύπωμα TLS πρωτοκόλλου, Περιγραφή εξυπηρετητών CnC , Συστήματα που αντιστέκονται στην ανάλυση κίνησης δικτύου

Επόπτης:
Σωτήρης Ιωαννίδης
Αναπληρωτής Καθηγητής
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The adoption of network encryption is rapidly growing. The 2019 Annual Report of *Let's Encrypt* [195] states that in just four years, global HTTPS page loads have increased from 39% to more than 80% [185]. In 2019, one year after TLS 1.3 been published as an RFC [184], IETF reports that its adoption is rapidly growing with a 30% of Chrome's Internet connections to negotiate TLS 1.3 [187]. Even though network encryption is crucial for the protection of users and their privacy, it naturally introduces challenges for tools and mechanisms that perform deep packet inspection and rely heavily on the processing of packet payloads. Typical applications of deep packet inspection are packet forwarding and l7 filtering [105, 174], while it is a vital operation in intrusion detection and prevention systems [202–204]. In addition, the majority of content and service providers perform network analytics using deep packet inspection (DPI) to improve network performance and provide good quality of service and experience to their users. However, with the widespread adoption of network encryption protocols, solutions that rely on retrieving meaningful information from packet payload contents are becoming less and less effective and new mechanisms must be employed to keep up with network encryption.

In the meantime, network encryption continues to be abused by malicious actors. Again in the 2021 TLS Telemetry Report [212], we can see that the proportion of phishing sites using HTTPS and valid certificates has risen to 83%. The ThreatLabz State of Encrypted Attacks Report estimates that more than 85% of attacks were encrypted in 2022 [217]. Moreover, although DNS over HTTPS (DoH) and DNS over TLS (DoT) have been proposed to promote user privacy [52], they have been exploited by Command and Control (CnC) servers that hide their communications [211]. Malware continues to be a crucial problem

in the Internet [60] and network encryption makes it more difficult for malware detection systems to identify them. Typical network intrusion detection systems (NIDS), such as Snort, inspect packet headers and payloads to report malicious or abnormal traffic behavior. In encrypted packets[1] though, the only information that makes sense is (i) TLS handshake packets and (ii) TCP/IP packet headers (the data transmitted in packet payloads is encrypted). So, even popular intrusion detection systems seem to inadequately inspect encrypted connections. The SSL Readme page of Snort, for instance, reports that when inspecting port 443, "only the SSL handshake of each connection will be inspected" [201].

Recently, machine learning techniques are widely used for traffic classification, network analytics and malware detection [7, 53, 70, 74, 106, 124]. The majority of these works show that despite having encrypted payloads in network packets, we are still able to classify network traffic even in a fine-grained manner [14,23,90]. Packet headers contain information like IP addresses, port numbers and packet data sizes. Flow duration and packet inter-arrival times are time-related features that are relevant in encrypted traffic analysis and can be easily computed. When properly combined, packet metadata can offer valuable traffic insights [7].

In this work, we thoroughly examine the state-of-the-art in network traffic analysis and inspection after the vast adoption of encryption and we identify the limitations of the proposed solutions. The vision of this work is to address the shortcomings that exist in the literature and propose approaches for practical and effective traffic analysis in the era of fully encrypted communications. In this dissertation, we employ passive and active traffic scanning techniques to investigate network characteristics after encryption. The main goal for our work in Chapter 2 is to enable practical network inspection for encrypted network packets (using passive network analysis), while the goal for our work in Chapter 3 is to collect valuable information and gain insights related to the activity of C&C servers (using active network scanning).

In Section 2.1, we focus on analysing encrypted traffic generated by Over-The-Top (OTT) mobile applications. Traditional DPI implementations can only extract very coarse-grained information for the majority of such traffic. Its analysis, however, is an integral operation for many network systems and needs to be improved to offer detailed traffic metrics for OTT applications. We implement a system that is able to extract essential in-

[1] With encrypted packets, we refer to TCP packets that are secured using the TLS protocol.

formation from encrypted traffic generated by mobile applications. Packets contain *metadata* usable even with encrypted traffic, such as packet sizes—information that can be extracted from the packet headers.

In Section 2.2, we focus on the identification of intrusion attempts on encrypted network traffic using packet metadata patterns. We examine the automatic generation of expressive signatures, which are compiled into an Aho-Corasick automaton that enables simultaneous multi-pattern matching. We evaluate the effectiveness of the signatures and we integrate them into an intrusion detection engine.

In Chapter 3, we generate TLS fingerprints using JARM [194], an open-source tool for active server probing. JARM fingerprints are used by popular Internet scanners, such as shodan [219] and censys [220]. Our goal is to provide a long-term measurement study of botnets, using C&C server information that is available in public datasets [214, 216].

In Chapter 4, we present the works that we find in the literature that are able to perform traffic processing and inspection even when the network is encrypted. We examine the use cases of these works (e.g., network analytics) and how authors achieve to implement such systems. Having no visibility over the packet payload contents introduces major challenges. Thus, goal of this literature examination is to identify the means to achieve encrypted network traffic analysis and inspection effectively. This study will help the reader of this dissertation to (i) understand the challenges of traffic inspection when the network traffic is encrypted or tunnelled, (ii) discover the uses cases and applications of encrypted traffic analysis, (iii) acquire knowledge on the methods that are used to achieve encrypted traffic analysis, (iv) deduce which techniques are appropriate respecting the objectives of a system, (v) recognize the constraints each method presents, and finally, (vi) come across with the publicly available datasets.

1.1 Objectives and Thesis Statement

Since this work is divided into passive and active network monitoring and analysis, we partition this section into two pieces.

First, the objectives of our work on passive traffic analysis are the following: (i) process encrypted network packets, (ii) identify the underlying activity in a fine-grained manner with high accuracy, (iii) maintain an expressive yet simple enough language to facilitate

automated mining, and (iv) take advantage of modern hardware architectures to enable real-time processing. In this part of the dissertation, we aim to provide valuable insights on the network traffic shape after the growth of network encryption. We have examined the literature and we have identified that there is a gap in practical solutions that can process network traffic and report events in real-time using simple and straightforward techniques. Thus, in this dissertation, we propose a methodology that enables fine-grained inspection of encrypted network traffic by monitoring packet metadata patterns to indicate specific events or activity. We show that our methodology can be applied in different use cases and applications, from application analytics to intrusion detection. We integrate the patterns and signatures produced into two high performance DPI systems and we evaluate the methodology effectiveness and the processing performance.

Second, the objectives of our work on active network scanning and analysis are the following: (i) contact servers characterized by malicious activity, (ii) collect the packets exchanged during this communication and construct a fingerprint that corresponds to a specific server, (iii) export communication patterns between servers of the same botnet family, (iv) re-identify servers of specific botnet families, (v) study the evolution of such servers in time. In this part of the dissertation, we aim to build a database with TLS characteristics and configurations of malicious servers on the Internet with ultimate goal to re-identify them without prior knowledge on the fly. Also, we examine the evolution of such servers to evaluate this approach.

Thesis Statement In this dissertation, we demonstrate that traffic inspection is still possible, even after the vast adoption of network encryption, simply by monitoring the sequences of packet payload sizes. We show that integrating this functionality into a DPI engine enables real-time processing, which is applicable in diverse test cases, such as usage analytics and intrusion detection. Finally, we take advantage of the fact that TLS fingerprints can reveal the identity and activity of CnC servers to perform a measurement study of botnet configurations based on those fingerprints.

1.2 Contributions of this Dissertation

The key contributions of this dissertation are the following:

- We present a practical methodology to collect, label, and analyse encrypted traffic generated by popular mobile applications and vulnerability scanners to identify usage events and intrusion attempts (Chapter 2).

- We propose an expressive pattern language to describe packet metadata sequences that signify such events and we confirm its effectiveness experimentally. We demonstrate that our pattern language is amenable to automated mining (Chapter 2).

- We integrate our pattern language with two DPI engine implementations to evaluate its performance against real, high-volume network traffic (Chapter 2).

- We probe malicious servers and we construct a database of fingerprints based on the exchanged client-server TLS handshake packets. Fingerprints are actively produced on a daily basis for 7 months (Chapter 3).

- We present the evolution of IP addresses that participate in botnets and the TLS fingerprints constructed during a 7-month period (Chapter 3).

- We classify unknown TLS fingerprints from suspicious server IP addresses found in two blocklists (with no prior knowledge of activity), based on our botnet fingerprint database (Chapter 3).

- We compare the constructed fingerprints from malicious servers against the fingerprints of legitimate servers. We show that the effectiveness of threat hunting with outdated fingerprints can be reduced. (Chapter 3).

- We present an extensive literature examination and propose a taxonomy for the state-of-the-art on encrypted traffic analysis (Chapter 4).

1.3 Outline of Dissertation

In Chapter 2 we present our work with respect to encrypted traffic inspection using packet metadata patterns. Specifically, we evaluate our approach using two different use cases: (i) network analytics in Section 2.1 and (ii) network security in Section 2.2. Then, in Chapter 3, we perform active probing for C&C server characterization. In Chapter 4, we review

the state-of-the-art in encrypted traffic analysis. Finally, in Chapter 5 we discuss several aspects of this work and its limitations.

Chapter 2
Identification of Events on Encrypted Network Traffic

In this section, we examine the analysis of encrypted traffic generated by diverse applications to demonstrate that it is still possible to inspect network traffic, even after the vast adoption of network encryption. Specifically, we focus on two different use cases: (i) mobile application usage analytics and (ii) network intrusion detection. Traditional DPI implementations can only extract very coarse-grained information for the majority of such traffic. Its analysis, however, is an integral operation for many network systems and needs to be improved to offer (i) detailed traffic metrics for mobile applications and (ii) effective intrusion detection. Network packets contain *metadata* usable even with encrypted traffic, such as packet timestamps and sizes—information that can be extracted from the packet headers or timed. To this end, we perform DPI over encrypted traffic generated by mobile applications, vulnerability scanners and botnets by inspecting patterns in network packet metadata.

2.1 Use Case: Application Usage Analytics

First, we focus on using patterns of packet size trains to identify OTT application events such as messaging, voice and video calls over encrypted traffic (Section 2.1.1). We evaluate the effectiveness of our approach in Section 2.1.2. We provide a full implementation as part of a DPI engine supporting rulesets with packet train patterns—matched using an automaton consuming packet sizes—on top of traditional substring and port number pat-

terns, to efficiently match and report events in encrypted network traffic (Section 2.1.3). Finally in Section 2.1.4, we discuss about the automation of the pattern generation. Figure 2.1 shows a high-level overview of the approach that we follow.

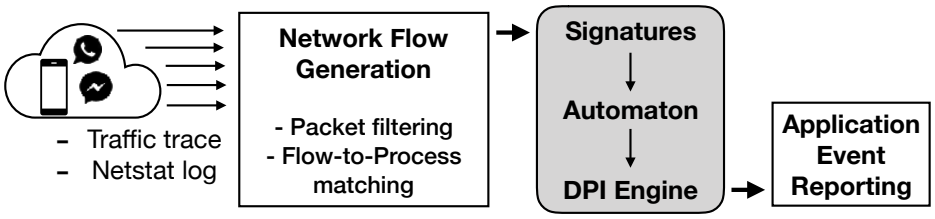

Figure 2.1: High-level overview: Traffic samples are collected offline and then signatures are created either manually or using data mining. The signatures are fed to our DPI engine and compiled into an automaton for execution on live traffic keeping only usage statistics.

2.1.1 Pattern Language

During our analyses, we observed that specific sequences of packet payload sizes reliably signify discrete events inside an application. In this subsection we describe our proposed pattern language to express such patterns in network traffic.

Design Goals

We aim for an expressive yet simple enough language to facilitate the automated mining of rules. While offline mining techniques can be involved during the construction of the rules, we need to support very efficient and low-latency evaluation of the rules at runtime on live traffic for use in a production quality DPI system. Another consideration for a practical system is to minimize the amount of state information that a DPI engine needs to maintain per flow in order to evaluate patterns across packets of the same flow. These requirements led us to a simple regex-inspired formulation applied on trains of observed packet sizes. The advantage of our approach is that it can be implemented with an automaton without the need to retain previously observed packet sizes to support backtracking, and that it is expressive enough to capture the traffic features of interest.

Table 2.1: Examples of application event rules.

Application	Event	Rule
WhatsApp	Voice call	3{1,3}, 56-60{1,3}, 400-800
WhatsApp	Video call	3{1,3}, 56-60{1,3}, 3{1,3}, 117 OR 3{1,3}, 56-60{1,3}, 3{1,3}, 144
WhatsApp	Chat message	3{1,3}, 52

Pattern Language Specification

Table 2.1 displays some examples of rules that we extracted during our analysis phase. The proposed pattern language uses a regex-inspired syntax, and is easy to follow, since it resembles standard regular expressions. When a network flow contains such sequences of these pre-defined payload packet sizes, expressed through a rule and in conjunction with any other traffic characteristics such as port numbers or substrings, then the application event is reported. For instance, when a captured network flow contains a series of two packets with payload sizes 3 bytes and 52 bytes respectively, then our system reports the existence of an outgoing chat message.

In order to deal with retransmitted TCP packets we could either (i) normalize traffic before applying the rule by discarding such packets or (ii) form the expression to handle the retransmitted packets, accordingly (like the rules in Table 2.1). The displayed expressions are able to handle retransmitted packets having a repeat range {1,3}, where 3 is the upper bound (the maximum number of retransmissions). However, handling retransmissions through the expression might be risky. Having retransmitted packets is an unpredictable network behaviour, so we might lose an application event reporting solely due to a not properly defined upper bound in the repeat range of an expression. Thus, we choose to handle retransmitted TCP packets by discarding them in a packet filtering phase.

2.1.2 Effectiveness Evaluation

In this subsection we demonstrate the expressiveness of the proposed pattern language by manually generating pattern signatures for a set of application events and evaluating their accuracy. We used 25% (randomly chosen) of the ground truth samples as a reference for the human analyst, and the remaining 75% for the accuracy evaluation (§2.1.2).

Flow Sample Collection Mechanism

We divide the mobile application network traffic into flows. A network flow is represented by the standard *5-tuple* containing (i) the source IP address, (ii) the source port number, (iii) the destination IP address, (iv) the destination port number and (v) the protocol. A network flow then, consists of the packets matching a certain *5-tuple*. To categorise the flows generated by different mobile applications, we need further information. This information should include either the domain, the process name or the process id that relates to the specific network connection. There are multiple ways to achieve this. For instance, other approaches, like [23], do domain filtering, leveraging the WHOIS protocol. We chose to employ the process id in order to obtain the required information about each network flow. In the following subsection we present how we implemented the network flow filtering.

Flow-to-Process Matching `Netstat` [176] is a command-line network utility that can display among others, information about network connections. Having superuser privileges, someone can use `netstat` to determine the process id (PID) and process name of the process that owns the connection socket. In Android devices, `netstat` is available via the `BusyBox` application [175].

To collect all necessary information about each connection established during the network traffic trace collection, we continually invoke `netstat` and store the output to a file that will be later used for the flow-to-process characterisation phase (flow/process correlation), during which flows are assigned to their process and (and the corresponding PID), generating a *7-tuple*, with the following format: *{process name, process id, source IP address, source port number, destination IP address, destination port number, protocol}*.

Packet Filtering In order for the TCP protocol to deliver data reliably, it offers many mechanisms to detect and avoid unpredictable network behaviour, like packet loss, duplication or reordering. In the proposed methodology, we choose to discard packets that do not offer substantial information to the flow (e.g. retransmitted packets). In our proposed method, we focus entirely on handling and processing packet metadata. This means that we do not take into consideration the packet payload, since we assume that it is encrypted. The information that we handle lays solely on packet metadata, such as the packet direc-

Table 2.2: The characteristics of the mobile devices that we used to collect our dataset.

Device Model	**Android Version**	**Kernel Version**
Sony Xperia D5503	Android v.5.1.1	3.4.0-gd26777b
Xiaomi Redmi 3s	Android v.6.0.1	3.18.20-g76f906f
Xiaomi MI Note LTE	Android v.6.0.1	3.4.0-gf4b741d
Xiaomi Redmi Note 3 Pro	Android v.6.0.1	3.10.84-gda78349

tion and payload size. Thus, packets lacking payload do not provide any valuable information to our method. To this end, we filter out ACK-flagged packets[1].

Sample Traffic Generation

To avoid extracting overly specific application event patterns, we analysed traffic traces generated during realistic usage of such applications. In addition, we used devices on both fixed and mobile networks.

Device Variations To ensure variation, we make use of different devices, vendors, Android and kernel versions, as shown in Table 2.2. We used four different Android mobile devices, a Sony Xperia D5503 (Android v.5.1.1, kernel v.3.4.0-gd26777b), a Xiaomi Redmi 3s (Android v.6.0.1, kernel v.3.18.20-g76f906f), a Xiaomi MI Note LTE (Android v.6.0.1, kernel v.3.4.0-gf4b741d), and finally a Xiaomi Redmi Note 3 Pro (Android v.6.0.1, kernel v.3.10.84-gda78349). In order to obtain full functionality and privileges, we used exclusively rooted Android devices, with developer options enabled. Thus, we were able to install the `BusyBox` application from Google Play store and take advantage of Unix utilities provided through a single executable [175], as well as the Android `tcpdump` tool to locally capture network traffic on the device [173]. In addition, we used Android Debug Bridge (ADB) version 1.0.39 and Wireshark 2.4.2. Due to toolset limitations, we did not include Apple devices in our study.

OTT Application Events We chose four of the most widely used OTT Android applications to evaluate our methodology: (i) WhatsApp, (ii) Skype, (iii) Facebook Messenger and (iv) Viber[2]. The applications' versions are presented in Table 2.3. Since these applications

[1] We discard the TCP packets with only the ACK flag set. PUSH/ACK packets are kept.

[2] Through the dataset collection we make use of different application versions per application. This allows us to verify the generalisation ability and scalability of our methodology.

Table 2.3: The Android OTT applications' versions for each one of the devices that we used.

Device Model	Facebook Messenger	Skype	WhatsApp	Viber
Sony Xperia D5503	146.0.0.33.136	7.46.0.596	2.17.427	7.9.4
Xiaomi Redmi 3s	155.0.0.14.93	8.16.0.6	2.18.65	8.4.0.4
Xiaomi MI Note LTE	155.0.0.14.93	8.16.0.6	2.18.65	8.4.0.4
Xiaomi Redmi Note 3 Pro	155.0.0.14.93	8.16.0.6	2.18.65	8.4.0.4

are mainly used for communication purposes, we focused on identifying (i) outgoing chat messages, (ii) voice and (iii) video calls through the encrypted network traffic. Of course, our work can be extended to support other OTT application events, such as media exchange (e.g. photo sharing), as well as iOS devices.

Overall, we collected a set of over 350 samples[3]. Each individual sample simulates either an exchange of an arbitrary number of outgoing messages (messaging), or a single voice or video call using one of the aforementioned OTT applications. Then, for each sample we collected (i) a network packet trace, (ii) a file with the information of every TCP socket that was open during the traffic capture and the process information that created it, (iii) a screen recording and (iv) a file with the device's system logs reported by the Android ADB tool, named `logcat`. Each sample contains only a single application event type (e.g. sample0: Skype/messaging).

To validate, we compare the detected application events to the device's system logs that are included in the `logcat` output and screen recordings. Using the `logcat` file and the screen recording we are able to cross-check the reported events with the actual ones. `Logcat` is a command-line tool that dumps a log of the device's system messages. We extracted information such as audio hardware on/off, camera on/off and incoming chat messages. Unfortunately, we were not able to identify a system event that matches an outgoing chat message. Thus, we had to use the screen recordings to inspect the actual time of an outgoing chat message departure, as well as the quantity of the outgoing messages.

Accuracy Evaluation

Hit Rate Table 2.4 shows the resulting true positive (TP) rates. Each sample contains only a single within-application event type (e.g. sample0: Skype/messaging, sample1:

[3]These samples were generated using dummy accounts and non-personal mobile devices.

Table 2.4: TP rates of our methodology. The percentages presented are extracted through the comparison of the results of our methodology to the actual ground-truth dataset.

Application	**Messaging**	**Voice**	**Video**
Facebook Messenger	83%	96%	96%
Skype	88%	100%	75%
Viber	100%	54%	88%
WhatsApp	100%	92%	75%

WhatsApp/voice). When a signature reports a within-application event (messaging: 0 or 1, voice: 0 or 1, video: 0 or 1), then we compare it to the actual event of the application. If the event is correctly reported, then the TP counter is increased. Otherwise, we have a false positive (FP).

The TP rate of our methodology individually for each event is (i) 93% for outgoing chat message, (ii) 86% for voice and (iii) 84% for video calls. The slightly lower TP rate for voice and video calls, is due to a trade-off with FPs[4]. We discovered that, for all applications under investigation except Viber, video-related flows included voice-related flows as well, and, thus, a video event includes also a voice event. On the other hand, our signatures for Viber voice and video events do not follow this trend as they are not complementary to each other. Thus, we can reach the interesting conclusion, that the core implementation of the Viber application is different from all the other applications under investigation.

False Discovery Rate In addition to true positives, another metric necessary for the evaluation of our methodology is the false positive rate for each application event. Reporting mobile application events using only encrypted network traffic can be considered risky since no easy cross-validation can be made. It is not only significant to correctly report the existence of events, but also to not mistakenly report absent events as existent. Table 2.5 shows the false discovery rates of event reporting using our signatures[5]. False discovery rates are always below 8%.

The choice of signature can significantly affect the trade-off between true positive and false discovery rates. Having a relaxed signature definition leads to almost intact TP rates,

[4]In the following subsection, we discuss about how the signature formation affects the balance between TP and FP rates.

[5]False discovery rate can be calculated as $FDR = FP/(TP + FP)$

Table 2.5: This table presents the false discovery rates of our methodology. The "Messaging FDR" column shows the percentages of erroneous messaging reporting in voice or video samples. Respectively, "Voice / video FDR" column shows the percentages of erroneous voice/video reporting in messaging samples.

Application	Chat FDR	Voice/Video FDR
Facebook Messenger	0%	1%
Skype	5.5%	4.2%
Viber	1%	2%
WhatsApp	8%	0.6%

with the cost of high false positives. Similarly, a more strict signature definition gives satisfactory TP rates, keeping the false positives low. We settled on signature definitions that result in hit rates over 84% and false discovery rates below 8%.

Granularity of Messaging Event Reporting Using our signatures for messaging reporting we achieve a total hit rate of 93%—again, compared to our ground truth data collection. This rate covers the correct identification of the existence of messaging events (i.e. outgoing text messages) within a mobile OTT application. Moving to a more fine-grained granularity, we are able not only to show that there is messaging activity within a network traffic trace, but also to accurately report *when* an outgoing text message is sent, and *count* the number of text messages sent during a messaging session, something we demonstrate in Section 2.1.4.

2.1.3 Implementation and Performance

In this subsection, we discuss and evaluate an implementation of our proposed pattern language.

Efficient Automaton

We implemented a data structure to efficiently match packet trains in a streaming fashion against sets of patterns. It is inspired by string searching algorithms such as Aho-Corasick [3] but instead of characters, it operates on packet sizes represented as 16-bit integers.

The Aho-Corasick algorithm is a string searching algorithm that locates elements of a finite set of strings within an input text. It matches all strings simultaneously, so its complexity does not depend on the size of the searched set. It works by constructing an automaton executing transitions for each character of the input text. To adapt the algorithm for matching packet trains, we replaced the 8-bit characters with 16-bit packet sizes.

The algorithm constructs a finite state machine that resembles a trie with additional "failure" links between the internal nodes. These failure links are followed when there is no other matching transition and allow for fast transitions to other branches of the trie that share a common prefix, without the need for backtracking using earlier inputs. This allows for interleaving a large number of concurrent searches, such as in the case of network connections, because the state of the matcher can be preserved across input data observed at different points in time by storing a pointer to the current state of the automaton with the state maintained for each connection. Otherwise, backtracking would require us to maintain expensive per-flow state for previously-seen packet sizes.

For additional performance, a Deterministic Finite Automaton (DFA) can be built by unrolling the failure links in advance and adding appropriate transitions to map each failure directly to an appropriate node without the need to follow multiple failure links at runtime. Expanding the automaton in this way did not provide an advantage in our case where the automaton is executed for each packet size as opposed to each byte when searching for substrings, and where the length and number of patterns is much less than typical substring-based rulesets, so we opted for the more compact data structure where the failure links are followed at runtime. For a very large number of patterns, however, this optimization may be worthwhile.

We implemented packet-size repetitions with a range $m - n$ as required by our pattern language by expanding them to $n - m + 1$ separate patterns. To implement packet ranges, we attempted at first to expand them into multiple individual 16-bit characters, leading to excessively large automata in the presence of wide packet size ranges, such as `100-200{3}` which would expand to 100^3 distinct sequences. To avoid this we use ranges instead of individual 16-bit characters for the arcs of the automaton. To simplify the implementation, we preprocess the expressions to collect possibly overlapping ranges used in them and extract a set of non-overlapping ranges that we use as the alphabet for the automaton constructed. For example, rule `152-156{1,5},150-600` contains two over-

lapping ranges, 152–156 and 150–600, which are expanded to an alphabet of three non-overlapping ranges: 150–151, 152–156, and 157–600. Subsequently, the repetitions in this example are expanded as shown in Figure 2.2.

```
152-156,150-151
152-156,152-156
152-156,157-600
152-156,152-156,150-151
152-156,152-156,152-156
152-156,152-156,157-600
152-156,152-156,152-156,150-151
152-156,152-156,152-156,152-156
152-156,152-156,152-156,157-600
152-156,152-156,152-156,152-156,150-151
152-156,152-156,152-156,152-156,152-156
152-156,152-156,152-156,152-156,157-600
152-156,152-156,152-156,152-156,152-156,150-151
152-156,152-156,152-156,152-156,152-156,152-156
152-156,152-156,152-156,152-156,152-156,157-600
```

Figure 2.2: Illustration of the complete expansion of rule `152-156{1,5},150-600` into a set of simple sequences of non-overlapping ranges. An alphabet of size three is used, each character corresponding to the range 150–151, 152–156, or 157–600.

DPI Engine Integration

We integrated the pattern matching data structure with our proprietary DPI engine that uses an extensible signature language by implementing a plugin to add a new condition, that we called `packet_train`. The signature language uses an event-condition-action model. The DPI engine raises different events to which sets of conditions and actions can be associated with. The conditions and actions are implemented as plugins, and are free to interpret their arguments and construct the necessary state objects that are evaluated on each event. The rule engine itself handles the logic of the ruleset as a whole, and the plugins are consulted for individual conditions. Each condition plugin declares the pieces of information that it requires (such as payload or flow-tuple information) and the rule engine ensures that the respective conditions are only used in combination with events that provide the required information. One such event is the `packet` event, which contains

information about packet payload and therefore packet size, that we make use of in our extension. Other events include `connection`, which is raised by the connection tracker. Information can be communicated across events by means of tags stored in the connection state, assigned by an action called `tag` and checked by a condition also called `tag`. These can be used to chain together rules triggered on distinct events, for example a rule could match a substring in a certificate to detect the application and tag the connection, while later the tag can be used in the rule that uses the `packet_train` condition to avoid evaluating flows from irrelevant applications.

Figure 2.3 illustrates a rule example. The conditions are evaluated as a conjunction. Disjunctions can be expressed using multiple rules, or (if the condition itself supports it, such as ours), with a list of arguments (Figure 2.4). The extension API provides hooks for populating individual condition arguments into a shared object that is consulted once per event and communicates back to the rule engine any matching rules. This facilitates conditions performing simultanous matching such as those based on Aho-Corasick or hash-tables.

```
facebook_video:
  event: packet
  conditions:
  - port: 443
  - packet_train: '399{1,2}, 51{1,2}, 1000-1260{1,2}, 38'
  actions:
    ...
```

Figure 2.3: Example of rule. The underlying data representation language used is YAML.

```
whatsapp_video:
  event: packet
  conditions:
  - packet_train:
    - '3{1,3}, 48-60{1,3}, 3{1,3}, 117'
    - '3{1,3}, 48-60{1,3}, 3{1,3}, 144'
    - '3{1,3}, 48-60{1,3}, 3{1,3}, 102'
```

Figure 2.4: Example of rule with a disjunction of patterns handled internally by the `packet_train` extension.

Performance Evaluation

We evaluated the performance of the entire system experimentally using our proprietary DPI engine in a live traffic test-bed. We used an HPE Proliant DL380 Gen9 server with two Intel® Xeon® E5-2699 v4 CPUs at 2.20 GHz with hyper-threading enabled, providing us with 88 logical cores (lcores), and configured with 1TB of RAM. The system has 4x40 Gbps NICs, two on each CPU socket. CentOS Linux release 7.4.1708 with kernel RPM version 3.10.0-693.11.6.el7.x86_64 was used.

The DPI engine is configured to use 8 lcores for processing the traffic from the four ports (two lcores per port). These lcores perform just sufficient packet decoding in order to load balance the traffic internally to 58 lcores configured to perform traffic inspection. These are the lcores running our implementation. The rest of the lcores in the system are dedicated to other tasks such as logging and shell access.

The traffic load consisted of real mobile user traffic that varies throughout the day between 52-153 Gbps with an average of 109 Gbps, 20-25 Gpps and between 67-230K new connections per second with an average of 161K/s. Throughout the experiments we confirmed that the system does not exhibit packet loss.

First we measured the baseline CPU utilization of the traffic inspection lcores using `mpstat` over 1 minute intervals. For a traffic of about 130 Gbps at 1pm local time, we measured a CPU utilization of 34.2%. After enabling our DPI engine extension, and making sure it is invoked for all packets, we measured 37.6%, an increase of about 10%. We also took a closer look using the `perf` tool, to narrow down on the specific function performing our checks, called `extension_packet_train_multiset_match`. We measured it at 3%, even without any actual patterns loaded. This number is an upper bound. If the automaton is fed only packets for pre-screened traffic that belongs only to the application (using appropriate signatures), the performance impact of our extension is expected to be less.

Subsequently, we loaded packet train signatures, increasing the number of signatures in each experiment to measure the impact of the number of signatures on the CPU utilization. We tried 1-5,10,15 and 20 signatures. The results were within the 2.7-3% range, with significant variance and without any observable trend. This observation shows that the bulk of the cost comes from the mere interposition of our extension into the DPI engine's pipeline and does not depend on the number of patterns, at least up to a number of 20

patterns.

2.1.4 Pattern Mining

Rule Mining Methodology

In order to illustrate the robustness of our event signature approach as well as to permit fast signature extraction for numerous application - event combinations, we automated the process. The application event rules were extracted from the packet traces by using frequent pattern mining (FPM) to detect frequent packet sequences and then correlating these patterns to the ground-truth events. This approach avoids the dependence on packet statistical measures commonly employed by other studies [1, 72, 127]. In order to extract the rules, the following steps are taken on the training dataset:

1. *Pre-processing*: All packets with a different process id than that of the application under examination are filtered out. Similarly, as mentioned in the above, TCP retransmissions are filtered out. Finally, all local and remote IPs are considered as a single local and a single remote IP, respectively.

2. *Packet statistics*: Afterwards, the absolute frequency of all pre-processed packet (source, destination, payload length) is calculated, and packet tuples whose frequency is greater than a predetermined percentile are mapped to unique identifiers (called items in the following). All other packet tuples are grouped according to their source and destination, as previously, but with the payload length segmented in 4 equally sized buckets, and similarly mapped to identifiers. This step was taken so as to limit the effect of variable payload length on the pattern mining (e.g., a long chat message may have a greater payload length than a shorter one).

3. *Trace splitting*: The packet traces were split to bursts (or sequences) of traffic (i.e., traffic with interpacket temporal distance less than a threshold, in this case set to 1 second) [1, 127]. It should be noted that as one of the type of events investigated is outgoing chat messages, a larger temporal threshold could potentially result in multiple chat messages included in one burst (chat messages sent in quick succession). Furthermore, bursts not containing any of the events under investigation are filtered out. This step is taken in order to divide the traffic to temporally correlated

sequences, which, in turn, will be used as an input to the frequent pattern mining algorithm.

4. *Frequent Pattern Mining*: Frequent pattern mining techniques are used to discern the correct packet patterns corresponding to the events among potential noise. The present methodology utilises closed sequential patterns (i.e., a pattern not strictly included in another pattern of the same support) as potential application event rules in order to avoid loss of information. The patterns are mined using the ClaSP algorithm [47].

5. *Rule Generation*: Finally, the rules are generated by identifying which closed sequential patterns match well with the ground truth events (i.e., the pattern timestamp is within a margin of the ground truth event timestamp).

In order to reduce the number of possible generated rules, the supersets of the above matching patterns are used, and evaluated using the F_1 measure (i.e., placing equal emphasis to both precision and recall). Finally, the generated rule is used to detect application events on the test dataset. The training dataset consists of 25% of the samples (the same samples as those used for training in the main implementation as mentioned in 2.1.2).

It should be noted that the rules generated by the above mining approach differ to those of the main implementation in that they take into account the direction of packet. This can be easily included in the DFA engine by encoding outgoing packets with a preceding minus sign to the payload size.

Rule Mining Evaluation

Table 2.6 shows the true positive rates achieved by the automated FPM methodology as well as the difference to the main implementation results. It can be seen that the FPM methodology outperforms the main implementation in all cases except Facebook where it underperforms. Furthermore, from Table 2.7, it can be seen that the performance of the two approaches on the false discovery rate metric is similar.

The FPM methodology is able to achieve accurate detection of distinct outgoing chat messages with a true positive rate and false discovery rate (FDR) of 98.55% and 3.54%, respectively, across all applications under investigation. Figures 2.5 and 2.6 show randomly

Table 2.6: TP rates of the automated FPM methodology. The difference to the main implementation is given inside the parentheses.

Application	**Chat**	**Voice**	**Video**
Facebook Messenger	42% (-41)	54% (-42)	83% (-13)
Skype	100% (+12)	96% (-4)	100% (+25)
Viber	100% (0)	96% (+42)	100% (+12)
WhatsApp	100% (0)	100% (+8)	100% (+25)

Table 2.7: False discovery rates of the automated FPM methodology. The "Messaging FDR" column shows the percentages of erroneous messaging reporting in voice or video samples. Respectively, "Voice / video FDR" column shows the percentages of erroneous voice/video reporting in messaging samples. The difference to the main implementation is given inside the parentheses.

Application	**Chat FDR**	**Voice/Video FDR**
Facebook Messenger	0% (0)	3% (+2)
Skype	2% (-3.5)	8.4% (+4.2)
Viber	3% (+1)	2% (0)
WhatsApp	2% (-6)	3.3% (+2.7)

chosen packet captures from WhatsApp and Skype messaging activity. We choose not to include the equivalent graphs for the remaining applications due to space constraints. The vertical lines depict the logged timestamp of the outgoing chat messages, while *Main* and *FPM* points show the detected events using the two proposed methodologies. The slight temporal deviation of the detected events from the ground truth timestamp can be explained from the fact that the outgoing message is not truly instantaneous, but rather spans from the transmission to the delivery acknowledgement.

Figure 2.5 shows a case where both our rule generation methods were able to perfectly detect the actual events, as opposed to the case shown in Fig. 2.6 where both false positives and false negatives are present. An interesting observation that can be derived is the increased Skype traffic during the time window 10:39:06 - 10:39:15. During this time, the user attempted to choose emoticons which were not pre-loaded.

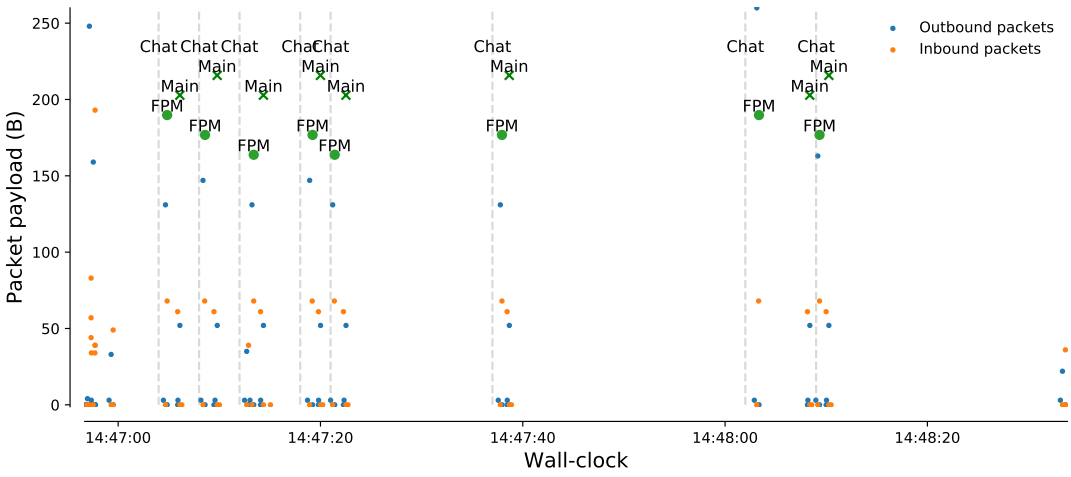

Figure 2.5: Packet capture of WhatsApp messaging activity. The vertical lines depict the actual outgoing chat messages, while *Main* and *FPM* points show the detected events.

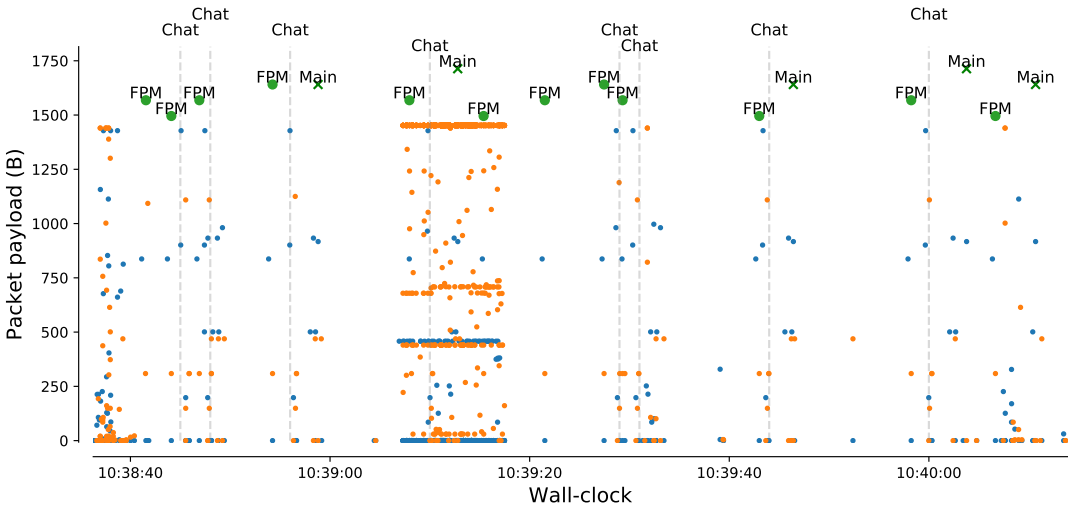

Figure 2.6: Packet capture of Skype messaging activity.

2.2 Use Case: Network Intrusion Detection

In this section, we focus on the identification of intrusion attempts on encrypted network traffic. We examine the automatic generation of expressive and fine-grained signatures. The signatures are tailored for intrusion detection in encrypted networks and are constructed using sequences of packet payload sizes (Sections 2.2.1 and 2.2.4). We evaluate the effectiveness of the signatures and we present the results in Section 2.2.2. In Section 2.2.3 we show how we modify a high-performance intrusion detection engine to support the matching of packet metadata sequences. For the evaluation, we use two different datasets: (i) a dataset with packet captures from several penetration tools that we collected in a controlled environment and (ii) a dataset of packet captures from IoT malware that is publicly available. A high-level overview of this work is presented in Figure 2.7. In the offline phase, we (i) process the ground-truth dataset retrieved from [171] and the *pentool-dataset* that we collected (§ 2.2.2), (ii) we generate signatures and (iii) we build the automaton. In the online phase, we process the input traffic using our intrusion detection engine that reports any suspicious activity identified by our signatures.

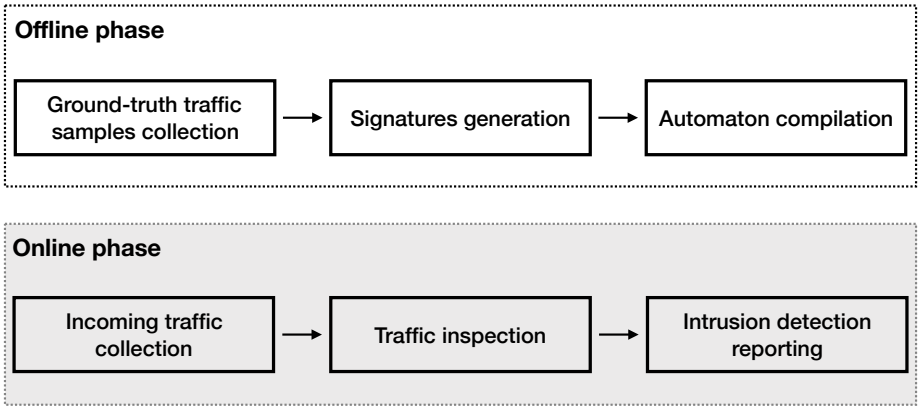

Figure 2.7: A high-level design overview of this work.

2.2.1 Signatures

A thorough examination of the literature and our own analysis, led us to conclude that the inspection of sequences of incoming packet payload sizes can point to discrete events that possibly signify an intrusion attempt within a network [90–92]. Figures 2.8, 2.9 and 2.10 show examples of how discrete events in network traffic can be revealed only by observing their patters of sequences of packet payload sizes. With signatures, we refer to sequences of packet payload sizes within a network flow. A network flow is characterized by the typical 5-tuple {*source IP address, destination IP address, source port, destination port, protocol*}. This mean that a signature is unidirectional. Signatures are matched against incoming traffic anywhere in the network flow.

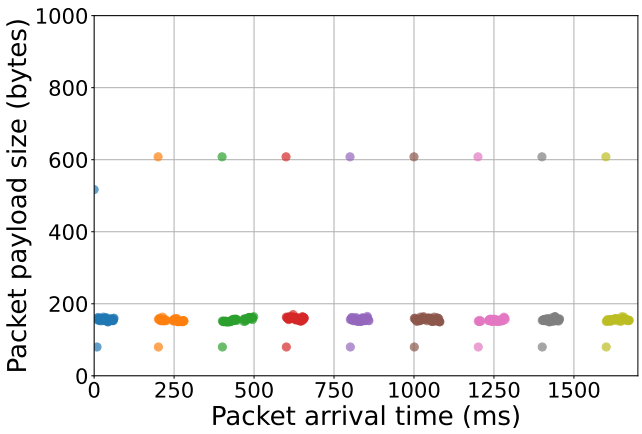

Figure 2.8: Illustration of packet payload size sequences within a network traffic capture of (a) file scanning attempt using the “dirbuster” tool, during the first 1.7 seconds of the active network flows. Each bullet color represents a single network flow.

In this section, we describe the signature language that expresses such network traffic patterns. Our goal is to develop a simple signature language that is expressive enough and enables automated mining, similarly to Section 2.1. Table 2.8 illustrates some signature examples that we construct during our analysis. The proposed format is easy to follow. An intrusion attempt event is reported, right after a network flow matches one or more signatures. For instance, when different network flows contain a sequence of 4 packets with payload sizes “22, 976, 48, 16” bytes respectively then our intrusion detection engine

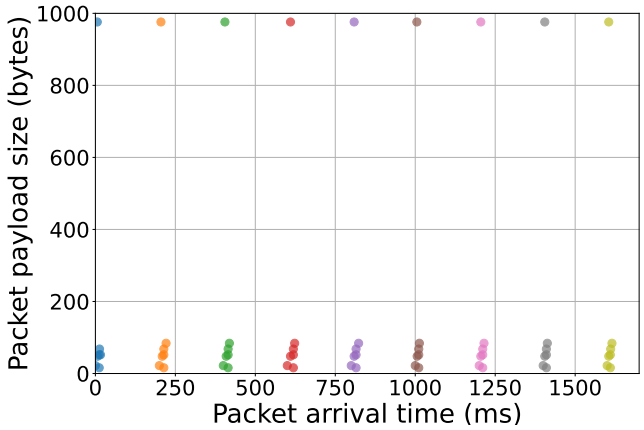

Figure 2.9: Illustration of packet payload size sequences within a network traffic capture of a ssh password cracking attempt using the "hydra" tool, during the first 1.7 seconds of the active network flows. Each bullet color represents a single network flow.

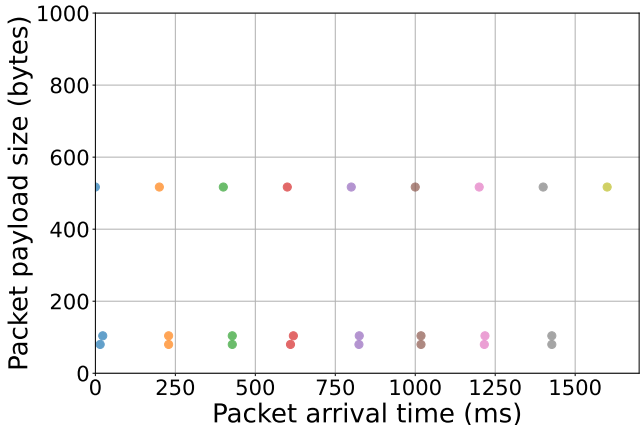

Figure 2.10: Illustration of packet payload size sequences within a network traffic capture of a login attempt to the web server using the "hydra" tool, during the first 1.7 seconds of the active network flows. Each bullet color represents a single network flow.

reports a password cracking attempt originating from the Hydra tool [193].

The proposed signature language can be extended by adding regular expression support for additional expressiveness. Yet, we decide to keep the signature language complex-

Table 2.8: Signature examples. Each signature corresponds to a sequence of packet payload sizes that must be matched against a network flow to report an intrusion attempt event.

Tool/Malware	Activity	Signature
Hydra	SSH Password cracking	22, 976, 48, 16
Dirbuster	File/directory scanning	608, 80, 155, 156
IRCbot	Communication with C&C server	16, 23, 19, 13
Muhstik	Communication with C&C server	78, 15, 31, 47

ity to a minimum in order to facilitate the automatic signature mining procedure. Similarly to 2.1.1, we normalize the network traffic by discarding retransmitted TCP packets in a filtering phase, before traffic inspection.

2.2.2 Effectiveness Evaluation

In this section, we evaluate the effectiveness of the proposed signature language. The intrusion events that we examine come from activity originated by (i) penetration tools (i.e., *pentool-dataset*) and (ii) IoT malware (i.e., *iot-malware-dataset*). The traffic that characterises the network activity of penetration tools is collected by us in a controlled environment and the traffic that characterises the network activity of IoT malware is retrieved by a public repository [171]. For the *pentool-dataset*, we used 30% (randomly chosen) of the packet captures for the signature generation, and the remaining 70% for the evaluation. For the *iot-malware-dataset*, a big majority of the malware families that exist in the dataset are contained in a single packet capture file. Thus, we split the network flows into separate packet captures into (i) signature generation captures and (ii) evaluation captures. The process is straightforward, since there are files that contain the necessary information (e.g., which network flows contain a certain malware activity). Just like in the *pentool-dataset*, we use the 30% of the total packet captures for signature generation and the remaining 70% for the signature evaluation [6].

[6]While in machine learning approaches, it is common to use 70% of the dataset for training and 30% for testing, we do the reverse to stress the effectiveness of our approach in cases of limited data.

Traffic Processing and Filtering

We divide the network traffic collected during the attacks into network flows. As previously stated, a network flow is characterized by the standard 5-tuple {*source IP address, destination IP address, source port, destination port, protocol*}. Since the network traffic from the *pentool-dataset* is generated within a controlled and isolated environment and the IP address of both machines is known, we can presume that the resulted flows in a packet capture indicate the traffic generated by the malicious machine during each corresponding attack. Regarding the *iot-malware-dataset*, we use the available logs that describe each attack and are available in the repository [171]. The *iot-malware-dataset* contains unencrypted network traffic, containing protocols like HTTP. Even though in HTTP traffic the payloads are not encrypted, we follow the same methodology to produce the intrusion detection signatures using packet metadata. The labels are produced after an analysis using the Zeek network security monitor [204].

In our methodology we discard retransmitted TCP packets, since such packets do not offer additional information to the flow. In addition, we assume that packet payloads are encrypted and thus, our approach proposes processing only packet metadata (e.g., packet payload size, packet direction). Packets that do not contain payload are also not processed (TCP ACK packets), since they do not provide any valuable information for our methodology.

Sample Traffic Generation

For the collection of the penetration tools dataset (i.e., *pentool-dataset*), we setup an environment with two virtual machines. The first machine runs Kali Linux and the second machine runs a vulnerable Ubuntu distribution with DVWA [190] installed using a self-signed certificate to enable HTTPS connections. The two machines are isolated from the network to ensure that no other machine is affected and a safe intercommunication between the two machines is established. The Kali Linux machine (IP address: 192.168.56.101) serves as the malicious entity that communicates with the vulnerable Ubuntu machine (IP address: 192.168.56.103) in order to perform various malicious activities (e.g., port scanning, file/directory scanning, password cracking, sql injection). The tshark tool is installed on the vulnerable machine and captures the incoming network traffic during the intrusion

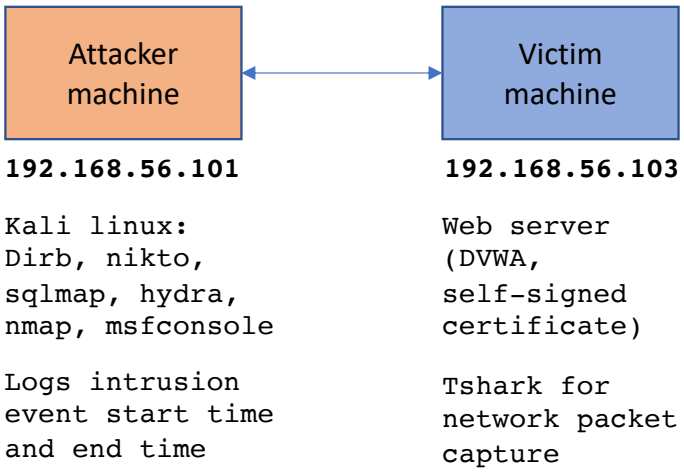

Figure 2.11: Illustration of our testbed setup for traffic collection.

attempts performed by the malicious machine. Figure 2.11 illustrates the testbed setup.

Table 2.9: Activities performed as intrusion attempts to the vulnerable web server.

#	Tool	Activity
1	Dirbuster	Web content scanning in victim machine
2	Nikto	Web server scanning in victim machine
3	Hydra	Admin login attempt to web server in victim machine
4	Hydra	Root login attempt to web server in victim machine
5	Metasploit	Directory scanning to web server in victim machine
6	Metasploit	File scanning to web server in victim machine
7	Sqlmap	SQL injection to web server in victim machine
8	Nmap	Detection of remote services version numbers
9	Nmap	OS detection, version detection, script scanning, traceroute

We choose some popular vulnerability scanners to evaluate our methodology. Some of the tools used for the intrusion events generation are *DIRB* [191], *NIKTO* [197], *SQLMAP* [188], *HYDRA* [193], *NMAP* [198], and *METASPLOIT* [196]. We perform numerous instances of attacks for different times and days within a one-month period. DIRB is a web content scanner. NIKTO examines a web server to find potential problems and security vulnerabilities. SQLMAP is a penetration test tool that enables SQL injection exploitations. HYDRA is used for login cracking. NMAP is a utility that enables network security auditing. METASPLOIT is a penetration testing software (we use the `msfconsole`, which is the metasploit framework console). Table 2.9 presents the events generated. Overall, we collected a set

of over 120 packet captures (half of these packet captures were generated using a different Kali Linux distribution, to ensure that the signatures are resilient across OS/application updates). Each individual packet capture simulates an intrusion attempt as described in Table 2.9. For each packet capture, we log the start time and end time of each intrusion attempt event.

We do not discuss in detail the testbed setup of the IoT malware dataset (i.e., *iot-malware-dataset*) in this section, since a thorough description of the dataset is provided in the public repository [171] by the creators. The events that we examine are presented in Table 2.10.

Table 2.10: Malicious activity as retrieved from the IoT-23 dataset [171].

#	Malware	Activity
10	Mirai	Connection of an infected device and a CC server
11	IRCbot	Connection of an infected device and a CC server
12	IRCbot	Attack from infected device to a host
13	Muhstik	Connection of an infected device and a CC server
14	Muhstik	Attack from infected device to a host

Signature Effectiveness

In this section, we evaluate the effectiveness of the signatures that are generated by our methodology. For the evaluation, we use 70% of the total packet traces that contain a single intrusion event (the remaining 30% packet captures were previously used for the signature generation).

Each packet capture in the dataset contains only a single intrusion event type (as defined in Tables 2.9 and 2.10). When a signature reports an intrusion event, we compare it to the actual event. For instance, when a signature reports that "web content scanning" probably occurs in a specific packet capture, we manually investigate the actual event. If the intrusion attempt event that happens in the specific packet capture is the same as the event that was reported, then we mark this report as correct. When an event is correctly reported, we increase the true positive counter. If the report is incorrect, we increase the false positive counter for the signature. The true positive rate (TPR) for each malicious activity is presented in Table 2.11. For the *pentool-dataset*, the TPR of our signature generation methodology is 100% individually for each event. This means that the signatures that are generated to report a specific intrusion attempt event can correctly identify the

existence of this specific event. For instance, the signature that is generated for the identification of *event no.1* "web content scanning in victim machine" using the `dirbuster` tool, correctly reports the existence of such event in every packet trace that indeed contains such event (signature TPR for *event no.1*: 100%). For the *iot-malware-dataset*, the TPR fluctuates between 62% and 100%. Mostly, the signatures that cause a low TPR are exported by packet captures with limited network flows to contain a malicious activity. For instance, the IRCbot network flows that contain a communication with a CC server are 1530 (*event no.11* TPR: 100%) while the IRCbot network flows that contain an attack are 677 (*event no.12* TPR: 62%).

Besides the true positive rate, we measure the resulting true negative rates (TNR) and false discovery rates (FDR) for each intrusion attempt event. The validation of results is challenging in encrypted network traffic. A network intrusion detection system must be able to report any traffic behavior that is suspicious, while it is equally important to not falsely report events that are not existent in the network. The false discovery rate of our methodology is reported in Table 2.11. False discovery rate is calculated as $FDR = FP/(TP + FP)$. In detail, events *no.1*, *no.2*, *no.7* and events *no.10–no.14* are expressed by signatures that perform very effectively, having high true positive rates and no false positives. Each signature that is generated by our methodology to identify each one of the three intrusion events (events no.1, no.2 and no.7) contains no less than two sequences of packet payload sizes. Thus, to report an event, a network flow must match each one of the sequences that are contained within the signature. This makes the three signatures (events no.1, no.2 and no.7) stronger than the remaining signatures that raise a number of false positives. For each one of the remaining tools (i.e., `hydra`, `metasploit`, `nmap`), our signature mining methodology produces a single signature. Each signature contains only a single sequence of packet payload sizes, which makes it easier for the signature to be matched against a network flow; something that could eventually present false positives. Moreover, we observed that the signatures that were generated by our methodology for two events of the same tool were identical. For instance, *event no.3* and *event no.4* (i.e., `hydra` tool) are described by the same sequence of packet payload sizes. Similarly, *event no.5* and *event no.6* (i.e., `metasploit` tool) share the same signature as well as *event no.8* and *event no.9* (i.e., `nmap` tool). As a result, different events of the same tool are reported simultaneously. For example, the false discovery rate is 11% for the signature of the *event*

Table 2.11: Resulting True Positive Rates (TPR), True Negative Rates (TNR) and False Discovery Rates (FDR) by the signatures examined.

# (Tool)	**TPR (in-dataset)**	**TNR (normal)**	**FDR (in-dataset)**	**FDR (normal)**
1 (Dirbuster)	100%	100%	0%	0%
2 (Nikto)	100%	100%	0%	0%
3 (Hydra)	100%	100%	11%	0%
4 (Hydra)	100%	100%	11%	0%
5 (Metasploit)	100%	100%	11%	0%
6 (Metasploit)	100%	100%	11%	0%
7 (Sqlmap)	100%	100%	0%	0%
8 (Nmap)	100%	100%	11%	0%
9 (Nmap)	100%	100%	11%	0%
10 (Mirai)	100%	100%	0%	0%
11 (IRCbot)	100%	100%	0%	0%
12 (IRCbot)	62%	100%	0%	0%
13 (Muhstik)	100%	100%	0%	0%
14 (Muhstik)	100%	100%	0%	0%

no.3, since the network traffic that is produced during an *event no.3* is falsely reported as *event no.4*, as well. Still, it is correctly reported as an *event no.3*. Minimizing the false positives that an intrusion detection system presents is very important. At this point, we highlight that the false discovery rate that is presented by some events (e.g., *event no.3*, *event no.5*) is negligible, if we consider that these signatures correctly report the existence of the tool and the traffic that it generates. Even thought the granularity of the event is not fine-grained (the generated signature for the `hydra` tool can not distinguish between events no.3 and 4), the signature is still able to correctly identify the existence of the traffic that the tool generates in a network. In addition, we use normal HTTPS traffic samples to measure the FDR for the signatures generated. These results are presented in Table 2.11 under the column *FDR (normal)*. The samples that we used for this experiment are publicly available [208] (packet captures used: CTU-Normal-20 – CTU-Normal-32). Correctly, our signatures do not report any intrusion event in the normal traffic dataset, leading to 100% TNR and 0% FDR. .

Comparison to Snort rules As an additional evaluation step, we compare the signatures generated by our methodology to the most relevant Snort signatures. We download the latest version of the community Snort rules [210] and we extract the rules that match the

Table 2.12: Comparison of the effectiveness of the rules that are generated by our methodology to the effectiveness of the corresponding rules that are used by Snort.

# (Tool)	Our TPR	Snort's TPR
1 (Dirbuster)	100%	0%
2 (Nikto)	100%	0%
3 (Hydra)	100%	–
4 (Hydra)	100%	–
5 (Metasploit)	100%	0%
6 (Metasploit)	100%	0%
7 (Sqlmap)	100%	0%
8 (Nmap)	100%	0%
9 (Nmap)	100%	80%

same tools. More specifically, we have identified 101 *metasploit* rules, six *nmap* rules, one *dirbuster* and one *sqlmap* rule. For *hydra*, we identified eight Snort rules, which seem to target the Hydra malware and not the *hydra* tool – thus, we choose to exclude them from the evaluation. Finally, since there was no rule for *nikto*, we used one rule that was present in an older version of community snort rules. We executed Snort using the equivalent Snort rules against the same network packet captures that we used for the evaluation in Table 2.11. As presented in Table 2.12, only the nmap rule that corresponds to the *event no.9* reported positively. Thus, it is apparent that Snort is not effective when performing against encrypted network traffic, in contrast to our methodology that matches packet metadata sequences and not packet contents. We do not execute Snort for the *iot-malware-dataset*, since it contains network packets that are not encrypted.

2.2.3 Implementation and Performance

A very efficient algorithm that popular signature-based intrusion detection systems use for pattern matching is the Aho-Corasick algorithm. Pattern matching is the core operation of any deep packet inspection system, such as a network intrusion detection system. A deep packet inspection system dives into the network packet payloads in order to extract sequences of characters, namely strings. These strings are compared against well known patterns that describe, for instance, the communication between a known botmaster with its bots.

In our approach, we assume that the network traffic that should be inspected by our intrusion detection system contains encrypted payloads. Thus, we do not extract any payloads and we only process packet metadata. These packet metadata can be derived from the contents of network packet headers. For example, even in a TLS protected connection, the packet headers are not encrypted. As we have already mentioned, our methodology uses packet metadata like the packet payload sizes (i.e., data transmitted in the packet) and packet directions in order to generate signatures. We express the packet direction implicitly, since a signature will match against one-directional network flows. A signature that we produce contains sequences of packet payload sizes. These sequences of packet payload sizes must be matched against the incoming network traffic in order to report an intrusion attempt event that is described by the corresponding signature. Yet, packet payload sizes are integers and can not be expressed as strings. Thus, integrating signatures of packet metadata into a typical signature-based intrusion detection system that performs deep packet inspection in packet payloads, is not trivial. In the following paragraphs, we describe the implementation of our system.

Simultaneous Multi-Pattern Matching and Efficient Automaton

The choice of the pattern matching algorithm is crucial for efficiently matching large data streams against multiple patterns. Inspired by the Aho-Corasick string matching algorithm [3], we implement a finite state machine to efficiently match a set of patterns (i.e., signatures) against streams of network packets. We extend the Aho-Corasick algorithm to enable integer matching, instead of strings, similar to [89–91].

The Aho-Corasick algorithm is a very efficient string searching algorithm that matches the items of a finite set of strings against an input stream. It is able to match a large volume of patterns simultaneously, so its complexity does not depend on the size of the pattern set. It constructs an automaton that performs transitions for each 8-bit ASCII character of the input text. For our approach, we replace the 8-bit characters with 16-bit values that represent the packet sizes. The algorithm builds a finite state machine, resembling a trie with added “failure” links between the trie nodes. When there is no remaining matching transition, we move through the state machine following the failure links, performing fast transitions to other branches of the trie that share a common prefix. In this way, we avoid the expensive backtracking operation, so the algorithm allows the interleaving of a large

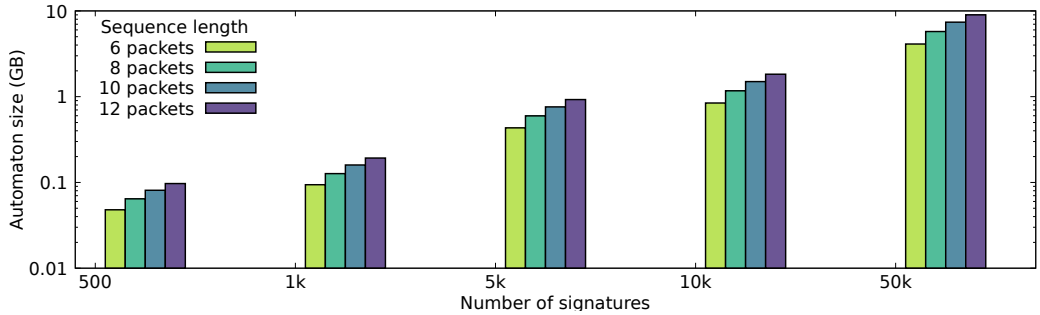

Figure 2.12: Automaton size.

number of concurrent searches, such as in the case of network connections, because the state can be preserved across input data that are observed at different points in time by storing a pointer to the current state of the automaton, with the state maintained for each connection. Backtracking is an operation very expensive since it requires the maintenance of per-flow state for previously-seen packet payload sizes. In order to boost the resulted performance, we build a Deterministic Finite Automaton (DFA) by unrolling the failure links in advance, adding them as additional transitions directly to the appropriate node.

To present our automaton's characteristics, i.e. the automaton size and the compilation time, we generate signature sets out of varying packet sequences, each time increasing the number of signatures and the packet sequence length. Figure 2.12 presents the size of the automaton in regard to different signature sets. More specifically, we present the size of our automaton, using 500, 1K, 5K, 10K and 50k randomly generated patterns of sequence length 6, 8, 10 and 12 packets; for example, the automaton that is generated using 10,000 signatures, where each signature resembles a sequence of 10 packet sizes, is around 1.5 GB. Figure 2.13 presents the compilation time of the automaton based on the same signature sets. The compilation time of the automaton does not affect the end-to-end performance negatively, since the compilation happens offline and only once.

Packet Processing Parallelization

For the implementation of the pattern matching of our intrusion detection system, we use the OpenCL framework (Intel OpenCL 2.1 SDK for Intel CPUs) [177, 199]. The overall architecture of our intrusion detection system is presented in Figure 2.14.

The system utilizes one or several CPU worker threads, assigned to a singe input source

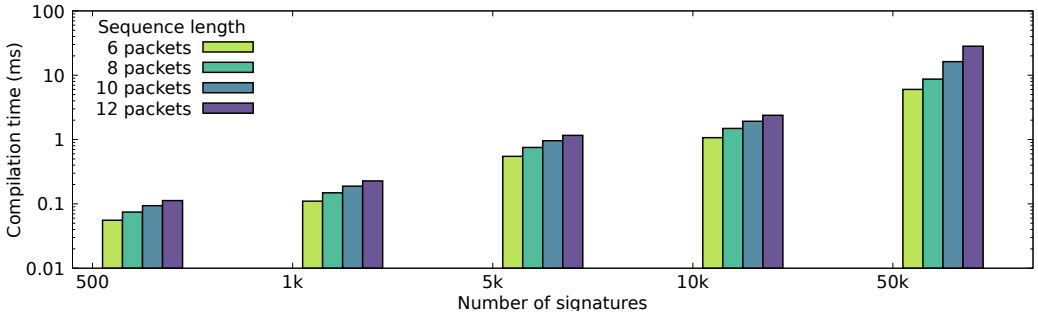

Figure 2.13: Automaton compilation time.

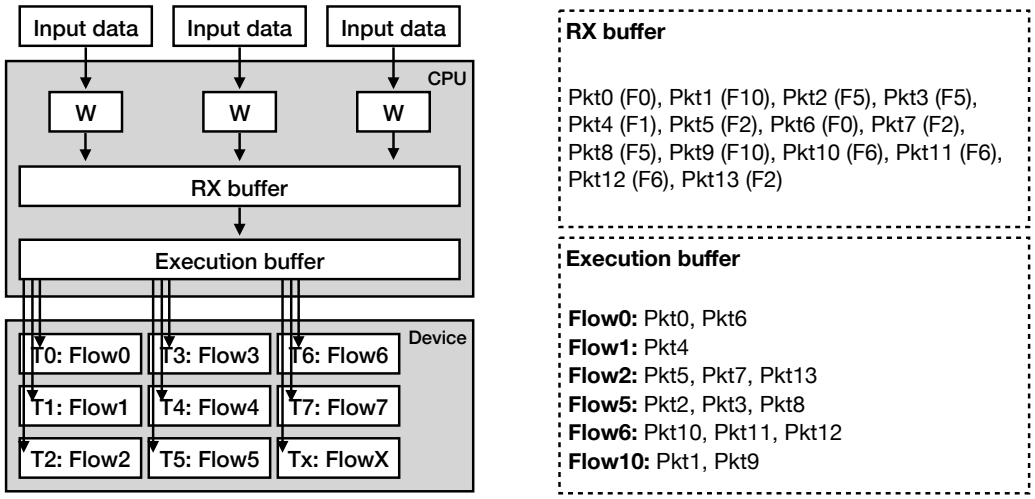

Figure 2.14: Overview of the proposed packet processing architecture.

(NIC or .pcap file). Once a CPU worker thread receives a network packet, it forwards it to a receive buffer. This buffer is responsible for the traffic processing and filtering, discussed in § 2.2.2. At this point, the receive buffer is filled with packets that belong to different network flows. When the buffer is full, our system generates execution batches with the traffic contained in the receive buffer. The execution batches contain the packet payload sizes of the received network packets, divided and ordered by network flows. In this way, we transform the input traffic to sequences of packet payloads. Each sequence refers to a single flow and is ready to be processed by the pattern matching engine. In the meantime, the receive buffer is accepting new incoming packets. Maintaining different buffers for receiving packets and for preparing packets to be processed enables us to avoid packet

losses. We implement the pattern matching engine of our system as an OpenCL compute kernel.

In OpenCL, an instance of a compute kernel is called a work-item; multiple work-items are grouped together and form work-groups. A data buffer required for the execution of a computing kernel has to be created and associated to a specific context. Unlike other relevant works that follow a packet-per-thread processing approach (e.g., [94, 131, 132]), we follow a flow-per-thread approach. This means that each thread reads at least one network flow from the execution batch and then performs the processing. Whenever a batch of packets is received and forwarded for flow ordering and processing by the device, new packets are copied to another batch in a pipeline fashion.

Moreover, in order to fully utilize the SIMD capabilities of the hardware, we represent the packet payload sizes in the execution buffer as `unsigned short` integers. In this way, we are able to access the data using the `ushort16` vector data type in a row-major order. This enables us to fetch information for 16 packets at once [178]. During processing, the pattern matching kernel uses one `ushort` value as input, representing one payload size, at each step in order to traverse the automaton, as described in Section 2.2.3. If a signature is identified, the engine reports the suspicious flow identifier, packed with the packets that matched the signature (using the first and the last packet contained in the signature, together with the signature identifier).

Performance Evaluation

For the performance evaluation of our implementation we use a commodity high-end machine. The hardware setup of our machine includes an Intel i7-8700K processor with 6 cores that operate at 3.7 GHz with hyper-threading enabled, providing us with 12 logical cores, configured with 32 GB RAM. The main processor is packed with an Intel UHD Graphics 630 integrated GPU. In our setup, we use Arch Linux with kernel version 4.19.34-1-lts. We perform offline traffic processing, meaning that the application reads the traffic from memory, thus, for the performance evaluation we focus on micro-benchmarks in order to present the throughput and latency of the engine's execution.

The performance results that are presented in Figures 2.15 and 2.16 display the median values occurring after 30 runs per configuration.

Figure 2.15 presents the processing throughput achieved by our pattern matching en-

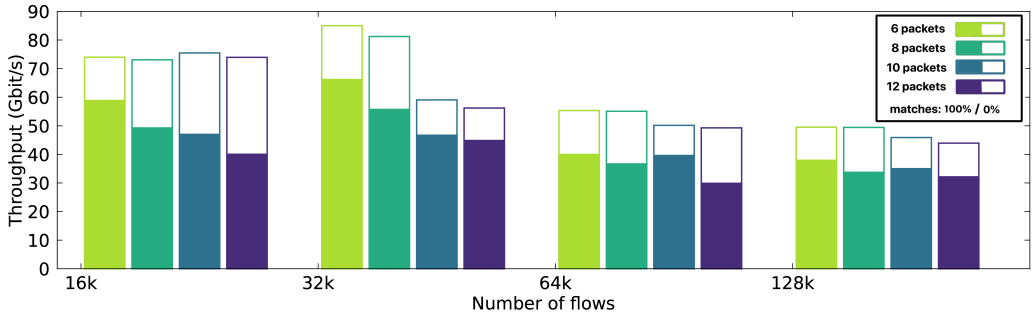

Figure 2.15: Throughput of our pattern matching implementation for different number of flows and varying pattern sizes.

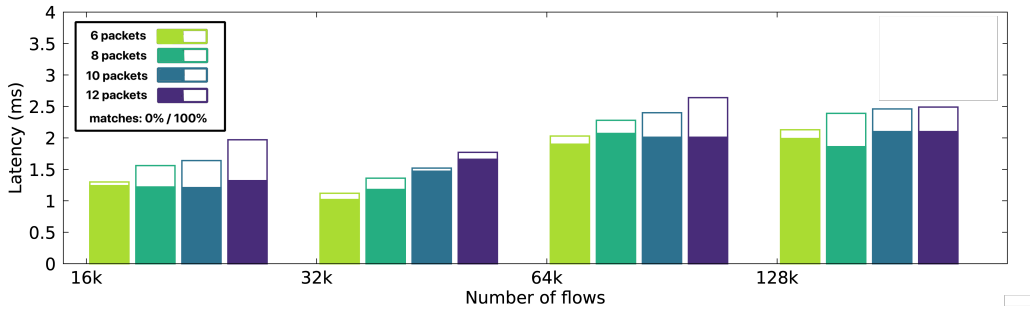

Figure 2.16: Latency of our pattern matching implementation for different number of flows and varying pattern sizes.

gine using an Intel i7-8700K CPU. In Figure 2.15, the color-filled bars indicate the throughput achieved by the pattern matching engine when the selection of signatures and input results to a computationally loaded condition (i.e., the 100% of the traffic reports signature matches). White-filled bars with borders indicate the throughput achieved in a computationally relaxed condition (i.e., less than 10% infected traffic), which is the most realistic scenario. We present the throughput using different packet batch sizes. The pattern matching engine is executed with one automaton of 1000 signatures of varying lengths (i.e., 6-12 packets). The main processor performs better for smaller batch sizes, resulting to 85Gbps processing throughput (i.e., 32k network flows per batch) for the realistic scenario and 69 Gbps processing throughput for the worst-case scenario.

In Figure 2.16, the color-filled bars indicate the performance achieved by the pattern matching engine when the selection of signatures and input, results to a computation-

ally relaxed condition. Again, in the figure, we present the most realistic scenario, where we have less than 10% malicious traffic. White-filled bars with borders indicate the performance achieved in a computationally loaded condition (i.e., 100% malicious traffic), which is the worst-case scenario. We present the latency using different packet batch sizes. Executing on the main processor adds very low latency, making it ideal for real-time, latency-intolerant environments. For instance, a batch of 16k network flows results up to 2ms processing latency, while processing larger batches takes up to 2.5ms. We use the same automaton as the one used in Fig. 2.15.

2.2.4 Signature Mining

To enable fast signature generation for detecting different intrusion attempts, we aim for automating the procedure. We extract the intrusion signatures from network packet traces using frequent sequential pattern mining. More specifically, from our ground-truth sample collection, we detect frequent packet payload size sequences that correspond to specific intrusion attempts. Unlike other works, our approach does not depend on network statistical measures for the encrypted traffic inspection [7, 22]. Figure 2.17 illustrates the workflow of our methodology.

First, we process the traffic captures so as to keep only the network packets that are related to the malicious activity. All the remaining packets other than the malicious activity under examination are discarded. Similarly, as already discussed in §2.2.1, we discard retransmitted TCP packets, as well. Then, we use the `joy` tool [182] to extract per network flow data that are used for signature generation. More specifically, `joy` receives as input a packet capture that contains an intrusion event (§ 2.2.2). `Joy` returns a JSON file with network flow related information, such as the sequence of packet sizes and arrival times, DNS names, HTTP header fields and others. For each network flow originated from the intrusion event under examination, we retrieve the sequence of non-zero packet payload sizes and the packet arrival times. This sequence of non-zero packet payload sizes is later used by the signature mining procedure.

For the signature mining procedure we choose to utilize a frequent sequential pattern mining technique. Such algorithms discover frequent sequential patterns that occur in sequence databases. Benefiting from such techniques, in our proposed methodology we

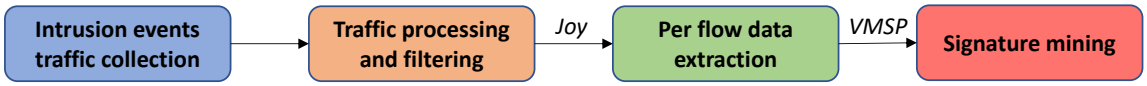

Figure 2.17: Illustration of our methodology workflow. First, we collect a set of ground-truth packet captures from intrusion attempts. Then, we process these captures and keep only the network packets that are related to the malicious activity. We use the tool `joy` [182] to extract sequences of packet payload sizes per flow and with a frequent sequential pattern mining algorithm, we generate signatures with sequences of packet payload sizes.

choose to utilize a maximal sequential pattern mining algorithm. Maximal sequential pattern mining is used to extract the frequent *longest common sequences* of network packet payload sizes contained in traffic. Our methodology uses the resulting sequences as potential signatures that can indicate an intrusion attempt. The resulting signatures are mined using the VMSP algorithm [42], with minimum support 50%. Finally, we select the maximal sequences that match to the ground truth information that we have. For instance, if the time window of the intrusion attempt is close (in time) to the sequence's first occurrence inside the network traffic. The generated signatures, then, are used to report intrusion attempts on a test dataset. The training dataset is the traffic traces that we used to export the frequent longest common sequences and generate the signatures. This training dataset consists of the 30% of the total traffic traces that we collected, infected with different intrusion attempt events (e.g., password cracking). For the generation of the signatures we take into account the direction of the packets (i.e., incoming packets).

How Signature Length Affects Effectiveness One of our design goals is to generate expressive signatures that will be able to detect intrusion attempt events in a fine-grained manner. After a manual examination of the signatures generated by the automatic methodology that we followed (§ 2.2.4), we found out that the majority of resulted signatures consisted of short sequences of packet payload sizes (e.g., median sequence length was 3 packet payload sizes). Having signatures with short sequences of packet payload sizes

results to a more compact automaton (i.e., less memory requirements). Yet, a short sequence can eventually result to high false positive rates, as well. A signature construction decision can significantly disturb the resulting rates of true positives and false positives. A short signature can lead to high true positive rates– yet, there is the cost of a high false positive rate. A large signature, with a more strict definition, can give satisfactory true positive rates and reduce false positives.

To prove our statements we perform an experiment. We use a publicly available dataset that contains network packet traces with several network intrusion events [80]. The events are classified into three categories: "Exploits", "Reconnaissance" and "DoS". For signature generation, we use 40% randomly chosen flows from the ground-truth dataset and the remaining 60% for the evaluation. Table 2.13 shows the resulting true positive rates (TPR) for different attack types found into the ground-truth dataset. Each traffic trace in the test dataset contains a combination of malicious and benign traffic (labeled). When a signature reports malicious activity we compare it with the actual category of the flow. If the activity is correctly reported as malicious, then the TP counter is increased. Otherwise, we have a false positive (FP). The true positive rates, as presented in Table 2.13 show the effectiveness of each signature according to the signature's length. For instance, short signatures result to higher TPR. In some cases, however, a short signature that results to a high TPR is possible to introduce the trade-off of resulting to a high FDR, as well. The false discovery rate[7], as presented in Table 2.14, presents the percentage of signatures that falsely reported malicious activity. Thus, as occurs from Tables 2.13 and 2.14, the signature length can significantly affect the signature effectiveness.

Table 2.13: Resulted true positive rates (TPR) of varying signatures between event category and size.

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Signatures for "Expoits" events	100%	93%	69%	63%	54%
Signatures for "Reconnaissance" events	100%	89%	89%	89%	87%
Signatures for "DoS" events	100%	61%	49%	44%	10%

[7]False discovery rate can be calculated as FDR = FP/(TP+FP)

Table 2.14: Resulted false discovery rates (FDR) of varying signatures between event category and size.

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Signatures for "Expoits" events	0.8%	0.7%	0.6%	0.2%	0.1%
Signatures for "Reconnaissance" events	62%	0.9%	0.7%	0.3%	0%
Signatures for "DoS" events	62%	43%	30%	21%	18%

Minimizing False Discovery Rates Figure 2.8 illustrates the sequences of packet payload sizes that appear within a network packet capture during a scanning attempt with the "dirbuster" tool. Examining the respective signature from the automatically generated signature set, we observe that the packet sequence that is depicted by the corresponding figure is the "608, 80, 155, 156". This packet payload size sequence is the longest common sequence in the dataset that was used for the signature generation (§ 2.2.4). Yet, we speculate that having such a short sequence would probably lead to higher false positives when evaluated in a larger testing dataset. Additionally to the VMSP maximal sequential pattern mining algorithm, we use the CM-ClaSP algorithm to discover closed sequential patterns [41]. Closed sequential pattern mining produces the largest subsequences that are common sets of sequences. So in practice, the resulted number of maximal patterns are less than closed patterns (or all patterns). Following the same signature generation methodology, combining the two frequent pattern mining algorithms, we obtain the following signatures for the intrusion attempt event illustrated in Figure 2.8 (selected set of sequences):

```
608, 80, 155, 155, 152, 152
...
608, 80, 155, 155, 152, 158
...
608, 80, 155, 156, 153, 152
...
608, 80, 155, 156, 153, 158
...
```

Signatures that occur from this optimization are more difficult to match an irrelevant network flow that will lead to false positives. For instance, a network flow with the packet

payload size sequence "608, 80, 155, 156, 90, 32, 60" (this sequence of packet payload sizes does not refer to a malicious event) would have matched against the short signature that was generated by our initial methodology (i.e., "608, 80, 155, 156"). However, if we combine the results of the two frequent pattern mining algorithms we will have a longer, more expressive signature set, which eventually will result to less false positives. Indeed, we noticed that this approach reduces some false positives from those of the initial methodology.

Early Intrusion Detection When it comes to intrusion detection, early reporting of an event is crucial. Figure 2.10 illustrates different packet payload size sequences within a network traffic capture of a login attempt to the web server using the "hydra" tool. We can observe that some packets from the same network flow are received within a negligible inter-arrival time (e.g., sometimes more than 3 seconds). Taking into account packet inter-arrival times, we can generate signatures with sequences of packet payload sizes that are received within a certain time-window. Aiming to keep the signature language as simple as it is at its current form, we add the extra parameter of time in the signature generation methodology. So, instead of having a packet payload size sequence will take almost 8 seconds to match against the network traffic, we settle to a shorter sequence that will report the intrusion event sooner. Using as example the packet capture of Figure 2.10, to quickly detect the intrusion event, we should replace the signature's sequence "517, 80, 104, 285, 229" produced by our methodology with the sequence "517, 80, 104, 285". In this way, our intrusion detection system will report the event after the first 4 seconds, instead of 7 seconds. This optimization offers a detection speedup of $x1.75$.

Chapter 3
Characterization of Malicious Servers on the Internet

In a TLS handshake, the first packets sent remain unencrypted and offer valuable information to traffic analysis tools, enabling fingerprinting of devices, operating systems and applications [96, 101]. While most of the works that perform TLS fingerprinting focus on passive methods, in this work we generate TLS fingerprints using JARM [194], an open-source tool for active server probing, with ultimate goal the effective identification of malicious command and control servers in the wild.

3.1 Background

Transport Layer Security (TLS) is an encryption protocol that is widely used to ensure the security and privacy of user communications online [184]. Specifically, TLS is used to encrypt and authenticate the communication channel between two endpoints, and it is widely adopted among others in browsing, messaging, voice over IP (VoIP) calls, emails. TLS allows endpoints to securely communicate over the Internet, hindering any possible malicious actions like eavesdropping, tampering and forgery.

A communication session between two endpoints starts with the TLS handshake. The TLS handshake mainly kicks-off the decision-making procedure between the two endpoints, about which TLS versions, encryption ciphers and extensions they will actually use during their communication. After this handshake, the two endpoints are able to share data, which is encrypted with the arranged TLS configurations between the two

endpoints. The only part of a TLS communication that is plain in sight, is the contents of the TLS handshake packets exchanged. Aiming to clearly explain how the TLS handshake works, we provide Figure 3.1. As shown in the figure, the endpoint that wishes to initiate a communication session with another endpoint sends a ‘‘TLS Client Hello’’ packet. In the ‘‘TLS Client Hello’’, the endpoint includes the TLS version that it supports, the encryption cipher suites and a string of random bytes. Then, the other endpoint responds with a ‘‘TLS Server Hello’’ packet. The ‘‘TLS Server Hello’’ packet contains the server's SSL certificate, the encryption cipher suit that the server wishes to use and a newly created string of random bytes. After the entity, which acts as client, receives the ‘‘TLS Server Hello’’ packet, it verifies the legitimacy of the server's SSL certificate with the authority that issued it. After the verification of the server's SSL certificate, the communication between the two endpoints starts[1].

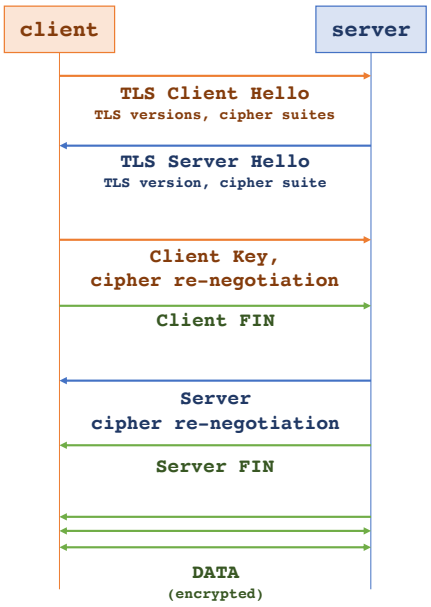

Figure 3.1: The TLS handshake steps.

In encrypted communications, extracting meaningful information about the contents of this communication is difficult. When it comes to preventing user data tampering and

[1]In this work, we utilize the information exchanged between the two endpoints that are present in the ‘‘TLS Client Hello’’ and ‘‘TLS Server Hello’’ packets. More information about the TLS handshake can be found in [184]

forgery by malicious entities, network encryption is of paramount importance. On the other hand, adversaries exploit the characteristics of widely used encryption protocols to hide their activities, making it impossible for security applications like network intrusion detection systems to completely block adversaries' actions. Aiming to stay effective in the current situation of the wide adoption of TLS and encryption protocols in general, the research community investigates new workarounds to deal with the rising problem of adversaries using network encryption to camouflage their existence and actions over the network.

3.2 Data Collection and Preliminary Analysis

As already mentioned, we make use of the JARM tool [194], which is an active TLS server fingerprinting tool. Based on the handshake properties of TLS, JARM actively sends 10 consequent "TLS Client Hello" packets to a server and collects the "TLS Server Hello" packets that come as responses. The 10 consequent "TLS Client Hello" packets that are sent to the target server are specifically generated to force TLS servers to response with unique responses. To be precise, JARM sends "TLS Client Hello" packets with different TLS versions, ciphers and extensions. The "TLS Server Hello" packets, then, contain information with the server's attribute combination of TLS versions, ciphers and extensions. Hashing these 10 server's responses, we receive a JARM fingerprint. The fingerprint is calculated for a single server and it is comprised of 62 Bytes in total.

On October 2021, we started collecting publicly available IP addresses from several blacklists. More specifically, we retrieve IP addresses from the Feodo Tracker Botnet C2 IP Blocklist [214] (CC1), the MalSilo IPv4 feed [216] (CC2), the CINS Score CI-Badguys list [213] (BL1) and the blocklists.de list [206] (BL2). The blacklists with identifiers CC1 and CC2 contain IP addresses, port numbers and activity/botnet name. The remaining blacklists with identifiers BL1 and BL2 only contain IP addresses. Thus, we produce fingerprints for every IP address encountered. Since the botnet lists (CC1 and CC2) contain more information, we classify the botnets based on the produced fingerprints. Then, we search these fingerprints against the blocklists BL1 and BL2. Figures 3.2(a), 3.2(b), 3.3(a) and 3.3(b) show the evolution of the IP addresses and fingerprints that we collect during this study. In the following paragraphs, we set the scene with a preliminary analysis based

on the data that we collect and calculate. As occurres, the different command and control servers that exist in the botnet lists that we download and parse, include the activity of the following botnets: (i) `Dridex`, (ii) `Qakbot`, (iii) `Trickbot`, (iv) `Emotet` and (v) `Downloader`. How the number of unique command and control server IP addresses evolve during our study is illustrated in Figure 3.2(a). Everyday, we download the fresh command and control server IP addresses from the botnet lists and we calculate the server fingerprints that are produced by JARM. The number of unique fingerprints per botnet is shown in Figure 3.2(b).

The number of unique Dridex command and control server IP addresses reach up to 186, while the fingerprints that are produced by actively probing those servers are maximum 8. This means that the majority of the Dridex command and control servers mostly have same TLS configurations. Likewise, the number of unique QakBot command and control server IP addresses reach up to 421, while the fingerprints that are produced by actively probing those servers are maximum 9. These number show that the fingerprints produced by the servers of this botnet are significantly uniform. For TrickBot, we encounter a maximum of 195 distinct IP addresses in a single day, resulting to 20 fingerprints. For the Emotet botnet, we encounter a maximum of 107 unique IP addresses in a single day, resulting to 4 fingerprints. The highest fingerprint diversity is introduced by the `Downloader` botnet, where for only 15 distinct IP addresses we get 6 fingerprints. Finally, our findings make us confident that maintaining a database with these fingerprints can eventually help mitigate cyber attacks related to the examined botnets 5.

Concerning the blocklists that we download and parse, we retrieve around 31K unique IP addresses from BL1 and 15K unique IP addresses from BL2 with only a small number of those IP addresses existing in both lists (Figure 3.3(a)). However, as shown in the following paragraphs 3.2.1, a large number of these IP addresses refused the TLS connections that we initiated using JARM. Yet, the number of the IP addresses that respond is still important and, thus, we continue our analysis for these IP addresses.

In Figure 3.3(b), we can see that the IP addresses that are contained in BL1 produce up to 759 unique fingerprints (out of 31K unique IP addresses), while the IP addresses of BL2 produce up to 413 unique fingerprints (out of 15K unique IP addresses). At this point, we have to stress that we do not have any prior knowledge related to the activities of each IP address that is included in the two blocklists BL1 and BL2. In addition, the

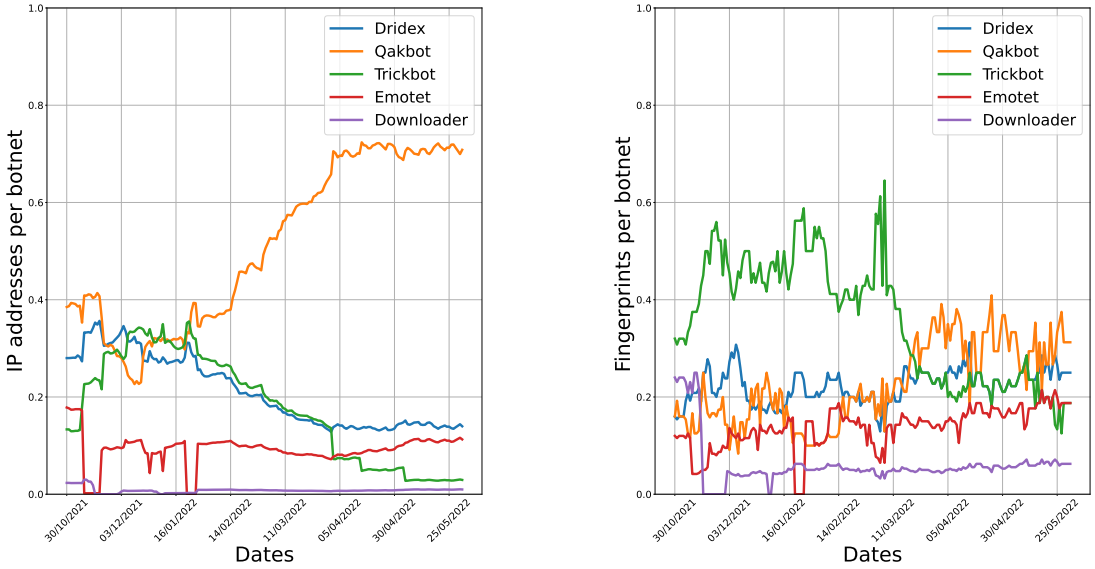

Figure 3.2: (a) The unique IP addresses contained in the list with the botnet command and control servers and (b) the unique fingerprints hashed out of the 10 TLS Server Hello responses, when contacting the IP addresses contained in the botnet command and control server lists that we parse (i.e., CC1, CC2).

two blocklists do not contain any port numbers, so we are just contacting IP addresses using the default port 443. Based on the fact that the servers of each botnet seem to be configured similarly (similar server TLS fingerprints between a large number of different IP addresses), we speculate that the two blocklists may contain IP addresses that serve numerous and different botnets, much more than the five botnets that we examine in Figures 3.2(a) and 3.2(b).

3.2.1 Refused TLS connections

Refused TLS connections from servers are expected. Figures 3.4(a) and 3.4(b) show the number of servers that refused all TLS connections that we started by not responding to the "TLS Client Hello" packets that we sent. From the two figures, we see that there are times that contacted servers refuse an incoming TLS connection. Specifically, in Fig-

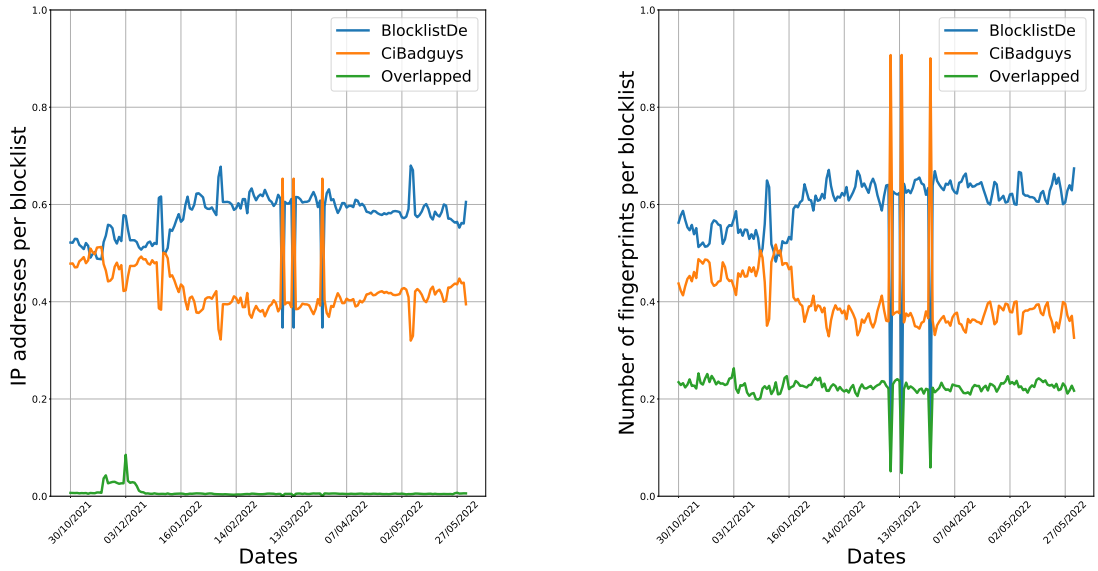

Figure 3.3: (a) The unique IP addresses contained in the blocklists and (b) the unique fingerprints hashed out of the 10 TLS Server Hello responses, when contacting the IP addresses contained in the blocklists that we parse (i.e., BL1, BL2).

ure 3.4(a) we plot the number of the refused TLS connections from known command and control servers (found in CC1, CC2). Servers from the Qakbot and Trickbot botnets mostly accept our connections and respond to the "TLS Client Hello" packets that we send them. Downloader servers also respond with a high ratio. Dridex and Emotet are the botnets that refuse incoming connections more frequently, but not always. For instance, Emotet does not refuse any of our connection during the period 23/01/22 – 27/01/2022. In Figure 3.4(b), we plot the ratio of the the server IP addresses that refuse incoming TLS connection from us and exist in BL1 and BL2. We can see that the TLS refusal ratio is higher and some days it can reach up to 60% of the total IP addresses from lists BL1 and BL2. Yet, taking into consideration that the total unique IP addresses in the two lists can reach up to 31K (BL1) plus 15K (BL2), the number of the fingerprints that is extracted and used for our analysis is more than considerable. When it comes to command and control servers, successful TLS connections are expected, since they are prepared for new connections coming from infected devices that act as bots. Thus, low TLS connection refusals are not a surprise. On

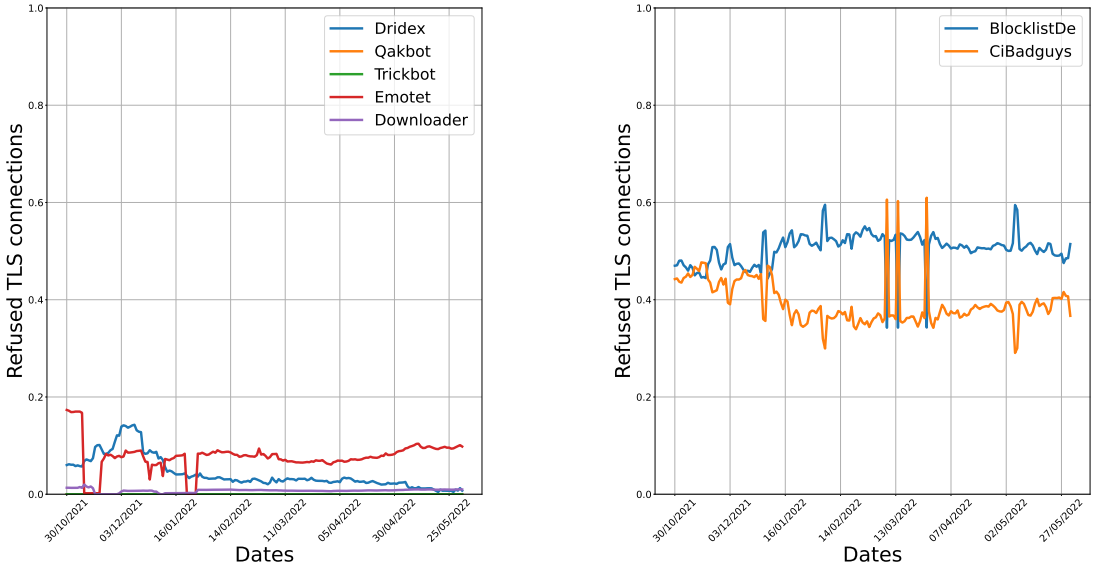

Figure 3.4: The number of refused TLS connections from IP addresses contained in the (a) botnet command and control server lists (i.e., CC1, CC2) and (b) blocklists (i.e., BL1, BL2).

the other hand, we do not have any activity insights regarding the IP addresses that are contained in blocklists BL1 and BL2. We speculate that servers from the blocklists that accept new TLS connections (through port 443) act as command and control servers. In addition, TLS connection refusals could be related to the lack of dedicated port numbers in the respecting lists. Specifically, our default port to probe is 443 and we do not perform any port scanning, since we want to stay as stealth as possible. As a result, in cases that a server responds only to connections that arrive through a dedicated port (other than 443), we get a refusal for initializing a TLS connection.

3.2.2 Fingerprints of benign servers

Our methodology of probing known suspicious IP addresses to produce TLS server fingerprints using the JARM tool would not be either effective or successful, if the fingerprints often overlapped with TLS server fingerprints of known legitimate servers from popular domains. We actively contacted the top-10K benign servers from the The Majestic Million

list [215] one day in October 2021 and one day in October 2022. Figure 3.5 presents the daily overlaps of the contacted server IP addresses retrieved from botnet server lists (i.e., CC1, CC2) and blocklists (i.e., BL1, BL2).

Results presented in Figure 3.5(a) show the daily overlaps of the malicious/suspicious servers with the benign servers of the top-10K from the Majestic list [215][2], with fingerprints calculated in October 2021. Figure 3.5(a) shows that for the server IP addresses that exist in the blocklists, we find some overlapping TLS server fingerprints to legitimate TLS server fingerprints, that could result to 5.3% false positives at maximum. The TLS server fingerprints that are produced using the IP addresses found in the botnet lists result to less than 1% overlapping fingerprints.

One year later, Figure 3.5(b) presents the daily overlaps of the same malicious/suspicious servers with the benign servers of the top-10K from the Majestic list [215][3], with fingerprints calculated in October 2022. Interestingly enough, we observe a rise in the overlaps. For server IP addresses that exist in the blocklists, we find overlaps of TLS server fingerprints with legitimate TLS server fingerprints, that result to 21% false positives at certain days. The overlaps of botnet server are significantly lower (still less than 5%), but higher when compared to the overlaps from Figure 3.5(a).

Based on our measurements the unique fingerprints of servers that exist in all lists examined (i.e., CC1, CC2, BL1, BL2) during the 7-month period are 3242. The unique fingerprints of the legitimate servers (top-10K Majestic) are 2915 in 2021, while in 2022 the unique fingerprints of the legitimate servers (top-10K Majestic) are 1311. The overlaps of malicious server fingerprints with legitimate server fingerprints are 371 in 2021 (13%) and 534 in 2022 (40%). Based on our measurements, we can see that in a year, the variation in TLS configurations of legitimate servers is reduced (i.e., less unique fingerprints). Furthermore, based on Figure 3.5, it is clear that as time passes, the calculated fingerprints of malicious servers are becoming less effective, when not updated.

[2]The list of domains is downloaded in October 2021.
[3]The list of domains is downloaded in October 2022.

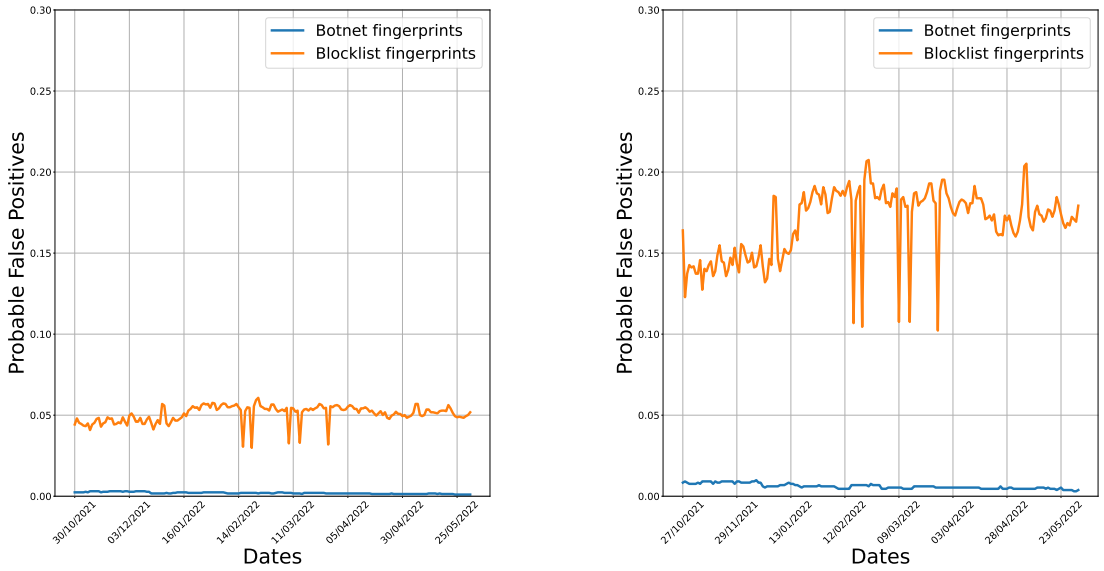

Figure 3.5: TLS server fingerprints (from servers found in CC1, CC2, BL1, BL2) that overlap with servers found in the top 10K domains ([215]).

3.3 Analysis

In this section, we analyse our findings based on IP addresses that we have been contacting for 7 months, and share our insights. On a daily basis we contact more than 30K distinct IP addresses, retrieved from numerous blacklists.

3.3.1 Botnet ports

For each established connection, the port number used can sometimes signify the underlying activity. Aiming to shed some light in the port selection by known botnets, Table 3.1 contains the top-5 ports that we encounter in the blocklists together with the IP addresses[4]. It is really interesting to notice that except for the well known ports (e.g., 443, 8080), the different botnets operate using a diverse list of port numbers. Such ports, along with the fingerprints produced, can offer better accuracy results in the performance of a network

[4]We wish to share the complete list of these port numbers with security researchers and students, upon request.

monitoring tool.

Table 3.1: Most popular port numbers per botnet (top-5 in CC1, CC2).

Botnet name	Popular ports (descending order)	Total ports
Dridex	443, 7443, 4664, 10172, 6225	45
QakBot	443, 995, 2222, 993, 1194	26
TrickBot	443, 447, 449	3
Emotet	8080, 443, 80, 7080, 4001	10
Downloader	6602, 1973, 13786, 29795, 46187	11

3.3.2 TLS Server Configurations

Each fingerprint that is calculated by the JARM tool for a single server is comprised of 62 Bytes in total. The first 30 Bytes present information about the configurations (i.e., TLS version and ciphers) of the server contacted and the remaining 32 Bytes contain information about the extensions. This means that for two different fingerprints that share the same first 30 Bytes, they also share the same configuration of TLS versions and supported ciphers. We believe that it would be very interesting and insightful to examine if there are trends between the servers that participate in the same botnet activity. Then, studying if these trends are distinct between different botnets, we might be able not only to tell if a server looks malicious, but to also indicate the botnet that the server is part of. Figure 3.6(a) shows that concerning the different botnets that we study, the TLS configurations of the servers are always less than the actual server fingerprints. This means that the servers that we probed in order to calculate their fingerprints based on the TLS Server Hello packets sent, share very similar TLS configurations. Examining the fingerprints produced by the IP addresses contained in the two blocklists, we see that the number of servers that share the same TLS configuration is significant. For instance on 23/11/2021, the servers with IP addresses contained in Blocklist.de produced 419 unique fingerprints of 62Bytes, with 206 unique fingerprints of 30Bytes. Similarly, the servers with IP addresses contained in the CI-Badguys blocklist produced 333 unique fingerprints of 62Bytes, with 165 unique fingerprints of 30Bytes. Figure 3.6(b) illustrates these variations.

Aiming to explore the uniqueness of the fingerprints produced, we check for overlapping fingerprints (length of 30Bytes) between the different servers that act as part of a certain botnet family. Table 3.2 shows the fingerprints that are common among the differ-

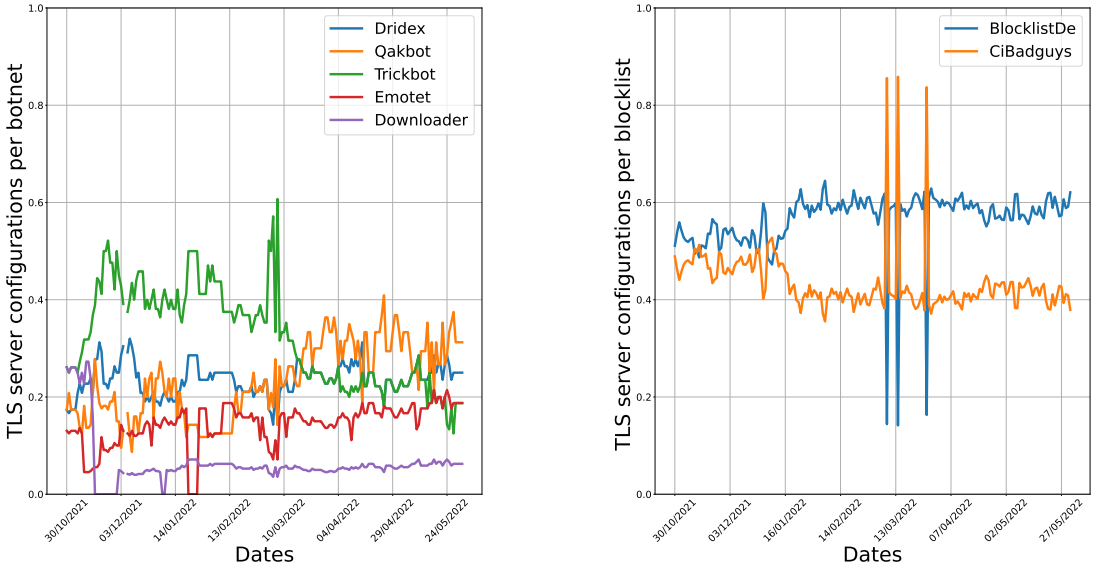

Figure 3.6: The number of unique TLS server configurations from IP addresses contained in the (a) botnet command and control server lists (i.e., CC1, CC2) and (b) blocklists (i.e., BL1, BL2).

ent botnets. Fingerprint `2ad2ad0002ad2ad0002ad2ad2ad2ad` is the most common and is produced by servers that are known to participate in Dridex, TrickBot and Emotet botnets. This fingerprint is also found in both blocklists that we parse, i.e. BL1, BL2. QakBot, TrickBot and Downloader share fingerprint `20d08b20d21d20d20c42d08b20b41d`, while it also appears in the two bloclists. The total number of unique fingerprints that overlap between different botnets are only 4. Removing the 4 *most common fingerprints* that represent the most common TLS configurations among servers, significantly decreases the number of fingerprint overlaps with benign servers that are included in the Majestic Million domains (causing a 20% decrease in numbers of overlaps shown in Fig. 3.5). This could mean that the malicious servers with the specific fingerprint imitate a popular TLS configuration profile used by normal and benign TLS servers (a technique that censorship circumvention tools also follow [43]).

Altogether, we can find many fingerprints that are calculated from the IP addresses that exist in the botnets' lists into the lists of fingerprints from IP addresses found in the two blocklists (i.e., BL1, BL2). The aggregated numbers of those fingerprints are shown

Table 3.2: Overlapping fingerprints (length of 30Bytes) between different botnets.

Botnet name	Botnet name	Fingerprint
Dridex	TrickBot	`2ad2ad0002ad2ad0002ad2ad2ad2ad`
Dridex	Emotet	`2ad2ad0002ad2ad0002ad2ad2ad2ad`
QakBot	TrickBot	`20d08b20d21d20d20c42d08b20b41d`
QakBot	TrickBot	`2ad2ad16d2ad2ad22c2ad2ad2ad2ad`
QakBot	Downloader	`04d02d00004d04d04c04d02d04d04d`
QakBot	Downloader	`20d08b20d21d20d20c42d08b20b41d`
TrickBot	Emotet	`2ad2ad0002ad2ad0002ad2ad2ad2ad`

Table 3.3: Botnet fingerprints found in blocklists CI-Badguys (BL1) and Blocklist.de (BL2).

Botnet	Fingerprints	In CI-Badguys	In Blocklist.de
Dridex	9	9	8
QakBot	36	22	25
TrickBot	47	14	14
Emotet	7	7	7
Downloader	7	6	7

in Table 3.3. This makes us confident that it is possible to distinguish servers of different botnets in a vast list of IP addresses without any prior knowledge, based only on the fingerprints calculated by JARM.

Indeed, the presence of the fingerprints that are calculated by the servers that participate in each botnet exists in the list of fingerprints that are produced by servers in the two blocklists. Figure 3.7 illustrates the ratio (values: 0 – 1) of fingerprints extracted from IP addresses that are contained in the botnet command and control server lists that were found between the fingerprints extracted from IP addresses contained in the blocklists. In Figure 3.7, we search the botnet fingerprints of full length against the fingerprints of the blocklists. For instance, on the 11th of November 2021 we extracted 372 fingerprints from servers found in Blocklist.de and 348 fingerprints from servers found in CI-Badguys list. Out of the total Blocklist.de fingerprints, 3 of them matched those of Dridex, 2 of them matched those of QakBot, 7 of them matched those of TrickBot, 1 of them matched those of Emotet and 3 of them matched those of Downloader. Out of the total CI-Badguys fingerprints, 4 of them matched those of Dridex, 2 of them matched those of QakBot, 5 of them matched those of TrickBot, 1 of them matched those of Emotet and 4 of them matched those of Downloader. The peaks observed in Fig. 3.7 (BL2) on the 9th, 14th and 27th of

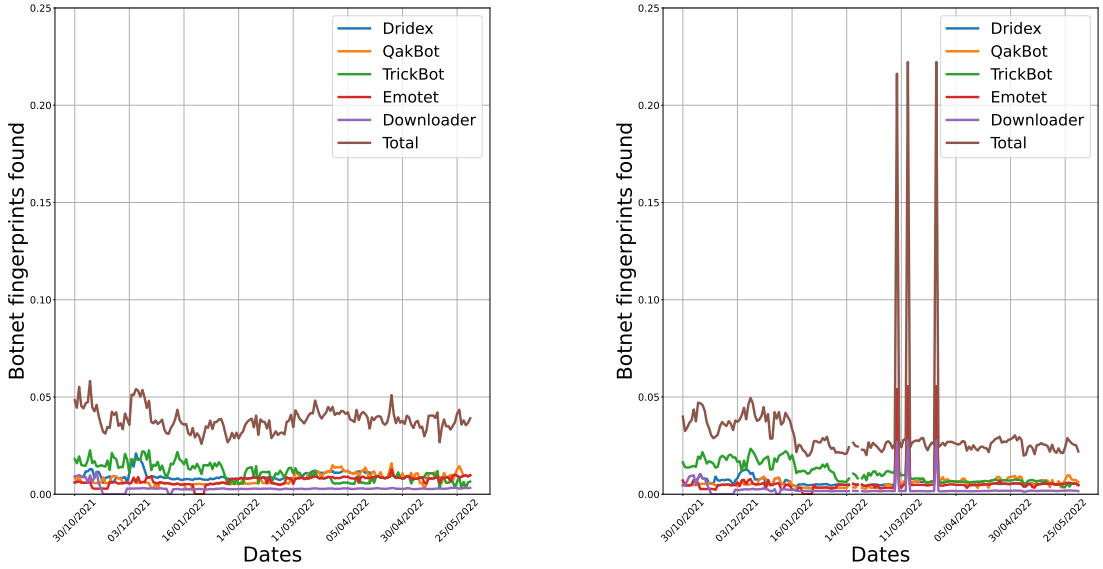

Figure 3.7: The ratio of botnet fingerprints found into the list of fingerprints calculated from IP addresses contained in the two blocklists (i.e., BL1, BL2).

March 2022, derive from a drop of the IP addresses contained in BL2. On those days, the unique IP addresses are almost 8K, while usually they are more than 23K.

In Figure 3.8, we search only the first 30 bytes of each fingerprint. As already mentioned, the first 30 bytes of each fingerprint calculated by JARM provides information about the TLS versions and ciphers that a server supports. Servers that share the same 30 first bytes in their fingerprints, also share the same TLS configuration (i.e., TLS versions and ciphers). As expected, searching for the 30 first bytes of the initial fingerprints results to higher number of botnet occurrences into the blocklists. Furthermore, extensions shared via the TLS Server Hello packet could add turbulence in fingerprints, since the application of popular fingerprinting circumvention techniques, like extension randomization, are common from servers with malicious activities (or censorship circumvention tools [43]). The existence of malicious servers that are similarly configured is highly expected. Thus, searching for these configurations could assist in their identification in encrypted communications. As shown in Figure 3.8, on the 11th of November 2021 we extracted 372 fingerprints from servers found in Blocklist.de and 348 fingerprints from

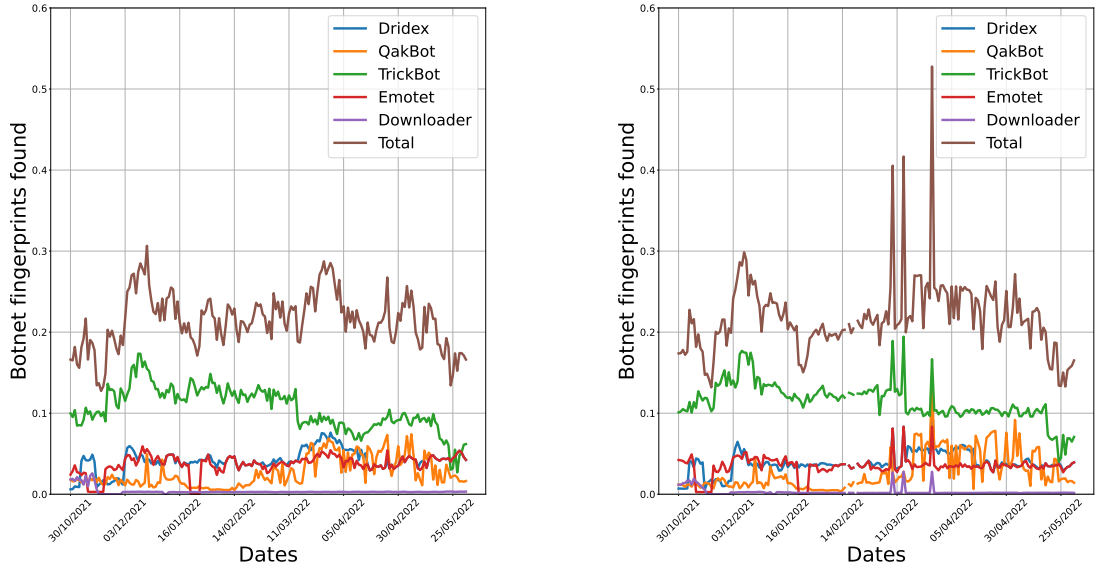

Figure 3.8: The ratio of short botnet fingerprints (30 first bytes representing the TLS version and cipher suites) found into the list of fingerprints calculated from IP addresses contained in the two blocklists (i.e., BL1, BL2).

servers found in CI-Badguys list. Out of the total Blocklist.de fingerprints, 14 of them matched those of Dridex, 3 of them matched those of QakBot, 45 of them matched those of TrickBot, 1 of them matched those of Emotet and 4 of them matched those of Downloader. Out of the total CI-Badguys fingerprints, 17 of them matched those of Dridex, 6 of them matched those of QakBot, 32 of them matched those of TrickBot, 1 of them matched those of Emotet and 9 of them matched those of Downloader. These numbers show a significant rise when we study the coverage of the examined botnets into the two blocklists, a trend that remains during the whole period of our analysis, with a peak of 30% total coverage on the 13th of December 2021 in both blocklists.

3.3.3 Randomization of Cipher Suite Vectors

As expected, malicious servers can take advantage of known TLS fingerprinting circumvention techniques like randomization of offered TLS cipher suites and extensions. Previous work though, has shown that this could be difficult in practice [54], while authors

in [8] show that it is possible to identify parts of those fingerprints. Another technique is to mimic popular TLS profiles – something that as we speculate, could explain the fingerprint overlapping that cause the false positives (as shown in Figure 3.5).

3.3.4 Configurations of Cipher Suites per Botnet

Diving into more depth, we further analyzed the data collected during the 7-month period. More specifically, for each IP address that we contact, we process the server responses to each TLS Client Hello that is sent. Table 3.4 presents the TLS versions that were selected by the botnet C&C servers across the 10 consequent TLS handshakes. While the majority of botnet families selected TLS 1.2 and 1.3, we see that Downloader also accepts communication using TLS 1.1.

Table 3.4: TLS versions used per botnet (CC1, CC2).

Botnet name	**Selected TLS versions**
Dridex	TLS 1.2, TLS 1.3
QakBot	TLS 1.2, TLS 1.3
TrickBot	TLS 1.2, TLS 1.3
Emotet	TLS 1.2, TLS 1.3
Downloader	TLS 1.1, TLS 1.2, TLS 1.3

Besides the TLS version chosen by the botnet servers, we also analyzed their preferred cipher suite. Table 3.5 presents the codes of the cipher suites that each botnet selects in most cases. In Table 3.6 we present a dictionary for these codes, along with their ranking by *ciphersuite.info* [218]. *Weak* ciphers are considered old and they should be avoided, while *insecure* ciphers can be broken with minimum effort. Dridex and QakBot select cipher suites marked as weak. In some cases, QakBot selects also an insecure cipher suite. TrickBot, Emotet and Downloader select a combination of weak and secure ciphers.

Table 3.5: Most selected cipher suites per botnet (CC1, CC2).

Botnet name	**Cipher suites (codes)**
Dridex	c013, 00c0, 0016, 0084, c012
QakBot	000a, 0005, c012, 006b
TrickBot	c030, c014
Emotet	c030, c012, cca8, 009e, 0016, 1302, c02f
Downloader	c012, c013, 0035, 1302

Table 3.6: Cipher suites dictionary and characterization by *ciphersuite.info* [218].

Code	**Cipher suite name**	Marked as
c013	TLS ECDHE RSA WITH AES 128 CBC SHA	Weak
00c0	TLS RSA WITH CAMELLIA 256 CBC SHA256	Weak
0016	TLS DHE RSA WITH 3DES EDE CBC SHA	Weak
0084	TLS RSA WITH CAMELLIA 256 CBC SHA	Weak
c012	TLS ECDHE RSA WITH 3DES EDE CBC SHA	Weak
000a	TLS RSA WITH 3DES EDE CBC SHA	Weak
0005	TLS RSA WITH RC4 128 MD5	Insecure
006b	TLS DHE RSA WITH AES 256 CBC SHA256	Weak
c030	TLS ECDHE RSA WITH AES 256 GCM SHA384	Secure
c014	TLS ECDHE RSA WITH AES 256 CBC SHA	Weak
cca8	ECDHE RSA WITH CHACHA20 POLY1305 SHA256	Secure
009e	TLS DHE RSA WITH AES 128 GCM SHA256	Secure
1302	TLS AES 256 GCM SHA384	Secure
c02f	TLS ECDHE RSA WITH AES 128 GCM SHA256	Secure
0035	TLS RSA WITH AES 256 CBC SHA	Weak

3.3.5 Advantages of Active versus Passive TLS Fingerprinting

Working with TLS fingerprints that are collected actively on a daily basis, offers the confidence of fingerprint persistence. Indeed, if we limit the fingerprint length down to 30 bytes, which shows the server's preferred configurations of TLS version and cipher suites (leaving the offered extensions out), we see that we are able to re-identify many servers operating at the same botnet in different blocklists (BL1, BL2).

In addition, since each fingerprint produced by JARM occurs after 10 consequent TLS handshakes with a single server, the resulted fingerprint is more rich in information when compared to simpler fingerprints (e.g., JA3) that are calculated by a single TLS handshake processed passively via sniffing a packet capture file. Also, since each fingerprint produced by JARM represents a server's response to a series of 10 different ‘‘`TLS Client Hello`’’ packets with diverse configurations of TLS versions and ciphers, each fingerprint is unique and difficult to replicate (even after the application of randomization techniques).

Chapter 4

State-of-the-Art

In this chapter, we present the works that we find in the literature that are able to perform traffic processing and inspection even when the network is encrypted. We examine the use cases of these works (e.g., network analytics) and how authors achieve to implement such systems. Having no visibility over the packet payload contents introduces major challenges. Thus, goal of this literature examination is to identify the means to achieve encrypted network traffic analysis and inspection effectively. This study will help the reader of this dissertation to (i) understand the challenges of traffic inspection when the network traffic is encrypted or tunnelled, (ii) discover the uses cases and applications of encrypted traffic analysis, (iii) acquire knowledge on the methods that are used to achieve encrypted traffic analysis, (iv) deduce which techniques are appropriate respecting the objectives of a system, (v) recognize the constraints each method presents, and finally, (vi) come across with the publicly available datasets that are appropriate for use.

Figure 4.1 displays the taxonomy that we propose for the related works of this dissertation. First, works are divided based on their use case and application goal. More specifically, we divide the works into four application domains: (i) the network analytics domain, (ii) the network security domain, (iii) the user privacy domain and (iv) the domain of network functions in middleboxes. Each work can be then characterized by the technique that is used (i.e., manipulation of traffic metadata & characteristics, interception of encrypted traffic, and utilization of cryptographic functions) and its main objectives (i.e., functionality, programmability and deployment).

In Section 4.1 we discuss about the works that target the network analytics domain. We divide the works into more detailed categories and we dedicate one subsection to one

sub-category (e.g., in § 4.1 we present works that focus on application and protocol classification, in § 4.1 we discuss works that identify application usage actions, etc.). Then, in § 4.1.1 we discuss about the algorithms and techniques used, in § 4.1.1 we present the publicly available datasets used and in § 4.1.2 we examine the objectives and limitations of the works reviewed in the section. We follow the same paragraph organization format for each one of the Sections 4.2–4.4.

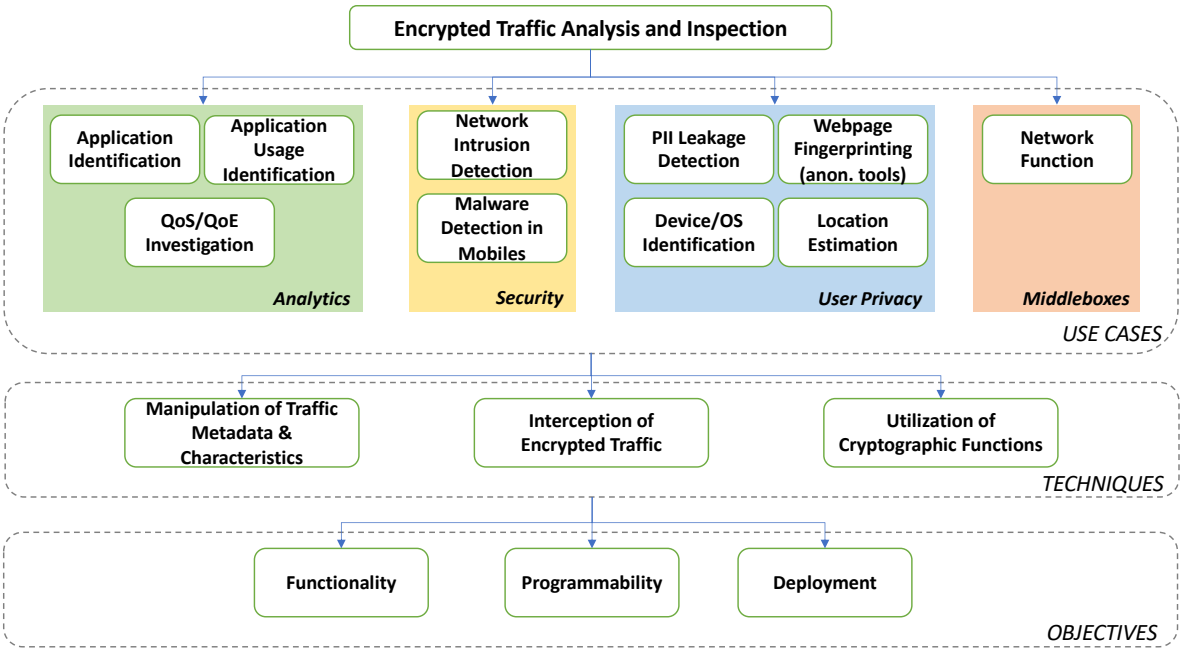

Figure 4.1: A taxonomy for encrypted network traffic inspection works categorized by use case, technique and objective.

4.1 Analytics after Network Encryption

In this section, we discuss the literature for network analytics in encrypted communications. More specifically, we present works that focus on protocol and application classification, application usage identification, as well as the investigation of quality and experience in encrypted networks. The works that are presented in this section are summarized in Table 4.1.

Table 4.1: Works in the analytics domain, sorted by category and publication year.

Traffic Analysis Domain	Work	Category	Goal	Year of Publication
Network Analytics	Karagiannis et al. [58]	Protocol/Application Identification	Network traffic classification using traffic behavioral patterns	2005
	Bernaille et al. [11]		Application detection in SSL encrypted network connections	2007
	Schatzmann et al. [109]		Webmail traffic classification	2010
	Wang et al. [134]		iOS application classification	2015
	Alan et al. [5]		Android application classification	2016
	Taylor et al. [126]		Android application classification	2016
	Lopez et al. [73]		IoT traffic classification	2017
	Taylor et al. [127]		Android application classification	2018
	Aceto et al. [1]		Mobile application classification	2018
	Aceto et al. [2]		Mobile application classification	2019
	Aiolli et al. [4]		Identification of user activities on smartphone-based Bitcoin wallet apps	2019
	Yao et al. [146]		IoT traffic classification	2019
	Ede et al. [130]		Mobile application classification	2020
	Xu et al. [143]		Mobile application classification	2022
	Coull et al. [27]	Application Usage Identification	Identification of user actions in iMessage	2014
	Conti et al. [22, 23]		Identification of user actions on Android devices	2016
	Saltaformaggio et al. [108]		Identification of user actions on mobile devices	2016
	Fu et al. [45]		Identification of user actions on mobile messaging applications	2016
	Liu et al. [72]		Identification of user actions on mobile messaging applications	2017
	Papadogiannaki et al. [90]		Identification of user actions on mobile Over-The-Top applications	2018
	Wang et al. [137]		Identification of user actions on mobile payment applications	2019
	Jiang et al. [55]		Identification of user actions in remote desktop traffic	2019
	Feng et al. [40]		Identification of user actions in online social networks	2022
	Wright et al. [140]	Stream Decoding	VoIP conversation decoding	2008
	Schuster et al. [110]		Video stream decoding	2017
	Björklund et al. [13]		Video stream decoding	2023
	Dimopoulos et al. [31]	QoS/QoE Investigation	Detection of QoE degradation	2016
	Orsolic et al. [86]		Estimation of QoE in YouTube	2016
	Mazhar et al. [77]		Investigation of video QoS in HTTPS and QUIC protocols	2018
	Khokhar et al. [59]		Estimation of QoE in YouTube	2019
	Xu et al. [144]		Investigation of mobile ABR video adaptation behavior under HTTPS and QUIC	2020
	Wu et al. [142]		Resolution identification of encrypted video streaming under HTTP/2	2023

Protocol/Application Identification There is a large number of works that examine the feasibility of traffic classification even when the communications are encrypted. The majority of such works focus on classifying the traffic's nature (e.g. video streaming, p2p traffic) and automating the classification procedure.

BLINC [58] was one of the pioneer papers that aimed to classify the network traffic

in the dark, having no access to packet payloads, no knowledge of port numbers; only information that flow collectors provide, solely based on the host behavioural patterns. Bernaille et al. [11] propose a method to detect different applications even in encrypted communication channels, by observing the first packets of an SSL connection and their sizes. This enables them to recognise the application soon enough, achieving an 85% accuracy and early classification. Schatzmann et al. [109] perform flow-based classification in order to identify webmail traffic. Authors focus on the inspection of network-level data and leverage correlations across protocols and time for HTTPS webmail classification. Wang et al. [134], taking advantage of the fact that mobile applications produce more identifiable traffic patterns, perform a packet-level analysis to determine what applications a single individual is using, exploiting side-channel information (e.g. traffic bursts) that is exposed inside the network traffic that is generated by mobile devices. To perform mobile app classification, the authors use random forests for 13 selected iOS applications achieving a classification accuracy of more than 68.59% with selected features and more than 87.23% with complete features. Alan et al. [5] investigate whether Android applications can be identified through their network traffic launch-time using only the contents of TCP/IP headers. The experiments were conducted using 1595 applications on 4 distinct Android devices. The authors made use of supervised learning methods to identify the apps that generated the traffic. Their approach is based on packet sizes, observed within the launch time traffic, since they are expected to yield good feature sets for application identification. AppScanner [126] automatically fingerprints Android applications even in encrypted traffic. For the generation of the fingerprints, authors collect network traffic traces on the mobile device while running the corresponding applications. The application classification is conducted using a supervised learning algorithm that is fed with features that are exported through the collection of network traces. This scalable framework implementation is able to identify the profiled applications (110 most popular applications in Google Play Store) with more than 99% accuracy. Stringoid [100] is a static analysis tool that estimates constructed URL strings using string concatenations in Android applications. The purpose of this work is to analyse web requests that originate from Android mobile applications. The authors use a dataset of 20 randomly selected Android applications, and with the Stringoid tool, they extract URLs from 30,000 Android applications. Lopez-Martin et al. [73] propose using a recurrent neural network (RNN)

combined with a convolutional neural network (CNN) for IoT traffic classification. The advantage of this work is that it outperforms alternative algorithms for traffic classification, while it does not require any feature engineering when applying new models. In a succeeding work, Taylor et al. [127] show that a passive eavesdropper is able to identify mobile applications by fingerprinting the network traffic that they send, despite encryption. Again, using AppScanner and machine learning techniques they exploit the information that lays in network traffic, such as packet size and direction. In addition they investigate how application fingerprints change over parameters like time, diversity of devices and versions. Aceto et al. [1] aim to improve the classification performance of mobile applications by proposing a Multi-Classification system, which combines specific decisions from base classifiers explicitly devised for mobile and encrypted traffic classification. The dataset that the authors used for testing, was collected by a mobile solutions provider. In a succeeding work, Aceto et al. [2] perform mobile traffic classification in encrypted network flow using deep learning techniques. Shen et al. [114] perform encrypted traffic classification of decentralized applications on Ethereum using a feature selection of packet lengths, bursts and time series. Aiolli et al. [4] identify user activities on Bitcoin wallet applications in mobile devices and are commonly used for sending, receiving, and trading Bitcoin. Yao et al. [146] perform IoT traffic classification for smart cities using a method that relies on a deep learning aided capsule network for efficient classification. Their proposed work eliminates the process of manually selecting traffic features. FLOWPRINT [130] offers mobile application identification by analysing the network traffic. It introduces an approach for application fingerprinting by combining destination-based clustering, browser isolation and pattern recognition (in a semi-supervised manner). It is able to construct mobile application fingerprints for not known applications. Authors evaluate FLOWPRINT and they find that it is able to perform an accuracy of 89.2%. Even after application updates or newly encountered applications, FLOWPRINT has a precision of 93.5%. As ground-truth, authors use publicly available datasets. As features, authors extract all header values controlled by the communicating app as well as the sizes and inter-arrival times of packets. In addition, for the size and time related features authors compute statistical properties, such as the maximum, standard deviation, mean absolute deviation values.

Application Usage Identification The works presented in this section offer fine-grained application event identification over encrypted traffic, often with the use of machine learning techniques.

Coull et al. [27] propose a method for traffic analysis of encrypted messaging services. More specifically, authors aim to show that an eavesdropper would be able to retrieve fine-grained information by the communication channel, such as specific user actions, the size of messages that are exchanged, or even the language that is being used for the communication. Their results demonstrate the feasibility of gaining information by observing packet lengths, but their analysis is limited to Apple's iMessage application and is an offline study. Conti et al. [22, 23] propose a system to analyse encrypted network traffic to identify user actions on Android devices, such as email exchange, interactions over social network, etc. Their framework uses TCP/IP packet fields, like IP addresses and ports, among with other features, like packet size, direction and timing. They analyse numerous Android applications with diverse functionalities, such as Gmail, Facebook, Twitter, Tumblr and Dropbox. Using machine learning, they achieve high accuracy and precision for the identification of different user actions in each tested Android application (e.g., mail exchange, posting a photo online or publishing a tweet). NetScope [108] is a work that performs robust inference of users' activities, for both Android and iOS devices, based on inspecting IP headers. This work demonstrates how a passive eavesdropper is capable of identifying fine-grained user activities within a network (even over encrypted communication channels) generated by the applications used. Based on the intuition that the implementation of each individual mobile application leaves a fingerprint on its traffic behaviour, such as transfer rates and packet exchanges, NetScope learns the subtle traffic behavioural differences between user activities becoming able to distinguish them. Fu et al. [45] propose an approach to classify usage in mobile messaging applications. Their system, namely CUMMA, classifies the usage in mobile messaging applications by taking into account user behavioural patterns, network traffic characteristics and temporal dependencies. More specifically, they show that the observation of packet lengths and time delays, can allow the classification of WhatsApp and WeChat traffic and identify the corresponding usage types (e.g. photo sharing). With this framework, the authors achieve 96% and 97% accuracy in WeChat and WhatsApp, respectively. Liu et al. [72] develop an analyser to classify encrypted mobile traffic to application usage activity. Using similarity

measurements, authors select discriminative features from traffic packet sequences. For their online analyzer, authors represent a traffic flow with a series of time windows. For their experiments, they analyse WeChat, WhatsApp and Facebook applications. Wang et al. [137] identify financial transactions at the trading stage via analyzing the encrypted network traffic, by identifying the mobile payment app from traffic data, classifying specific actions on the mobile payment app, and finally, detecting the detailed steps within the action. Jiang et al. [55] investigate if remote desktop traffic, even if encrypted, can reveal usage information. Indeed, their results show the feasibility of this, taking advantage of side-channel information leakage.

VoIP Conversation and Video Stream Decoding There are also works that use traffic analysis to extract voice information from encrypted VoIP conversations or identify encrypted video streams.

For example, Wright et al. [140] show that when the transmitted audio is encoded using variable bit rate codecs, the length of VoIP packets can be used to recognize words or phrases within a standard speech corpus. This means that a passive observer can identify phrases even in encrypted calls with an average accuracy of 50%. Schuster et al. [110] explain the root causes of burst patterns in encrypted video streams, show how to exploit these patterns for video identification, develop and evaluate a noise-tolerant identification methodology based on deep learning and, finally, they demonstrate how an attacker without direct observations of the network can identify videos being streamed. The features that authors use are flow attributes, such as down/up/all bytes per second, down/up/all packet per second, and down/up/all average packet length. The applications that authors examine are Netflix, Youtube, Amazon and Vimeo. Björklund et al. [13] identify SVT Play streams against HTTPS traffic. Authors build a k-d tree over sliding windows of packet statistics, while they use nearest-neighbour (NN) searches combined with Pearson's correlation coefficient to identify the videos.

Quality of Service/Experience investigation Streaming video content on mobile devices is a trend that is continually growing among users. This causes a tremendous demand for higher bandwidth and better provisioning throughout the network infrastructure. End-to-end encryption, though, leaves providers with limited indicators for identifying QoE

issues. Thus, the works presented in this section aim to measure QoS and QoE from the perspective of a telecommunication service provider that has only visibility on the network traffic that is often encrypted.

Dimopoulos et al. [31] propose models able to detect different levels of QoE degradation that is caused by stalling, average video quality and quality variations. The predictive models that authors develop are evaluated on the production network of a large scale mobile operator, where authors show that their system is able to accurately detect QoE problems with up to 92% accuracy. The significant features that authors extract are RTT-related, bytes transmitted, packet loss percentage and other network-related features. Orsolic et al. [86] use machine learning for the estimation of YouTube Quality of Experience. To test their approach, authors collect more than 1k different YouTube video traces under different bandwidth scenarios. Mazhar et al. [77] investigate the Quality of Service of video in HTTPS and QUIC protocols. The set of features that expose usable information is based on (i) network and transport layer header information for TCP flows, and (ii) network layer features (based on inter-arrival time, packet sizes, packet/byte counts, throughput) for QUIC flows. Khokhar et al. [59] put YouTube under experimentation and perform network traffic measurements for QoE estimation using network related features, as well. CSI [144] infers mobile ABR video adaptation behavior under HTTPS and QUIC using packet size and timing information. H2CI [142] monitors the resolution of encrypted video traffic under HTTP/2, based on fingerprints constructed by the sizes of mixed audio-video chunks.

4.1.1 Techniques

The majority of the works discussed in this section, employ machine learning techniques to investigate the feasibility of traffic analysis and usage classification even when the communications are encrypted. In this section, we present the techniques and algorithms that are more popular among the works of this category. A detailed overview of the techniques used for classification can be found in Table 4.2. As illustrated in Table 4.2, machine and deep learning techniques are very popular in the domain of network analytics. More specifically, it seems that supervised or semi-supervised algorithms are mostly used for traffic classification and network analytics. These algorithms perform well with labelled and big datasets, while they require training. For instance, for mobile application

Table 4.2: Techniques, algorithms and evaluation metrics used in the analytics domain.

Work	**Category**	**Algorithm/Technique**	**Performance Evaluation Metrics**
Karagiannis et al. [58]	Protocol and Application Identification	Graphs, Statistics	Completeness, Accuracy
Bernaille et al. [11]		Clustering with Gaussian Mixture Model	True/False Positive Rates
Schatzmann et al. [109]		Classification with Support Vector Machine (SVM)	Accuracy
Wang et al. [134]		Classification with Random Forests (RF)	Estimated accuracy, Overall accuracy, True Positive Rate
Alan et al. [5]		Classification with Jaccard's coefficient, Gaussian Naive Bayes and Multinomial Naive Bayes	Accuracy
Taylor et al. [126]		Classification with Multi-class Support Vector Machine (SVM), Multi-class RF, Binary SVM and Binary RF	Speed of training, Size of classifier, Confidence per classification, True Negatives, Robustness
Lopez et al. [73]		Classification with Recurrent Neural Network combined with a Convolutional Neural Network (CNN)	Accuracy, F1-score, Precision, Recall
Taylor et al. [127]		Classification with Multi-class Support Vector Machine (SVM), Multi-class RF, Binary SVM and Binary RF	Precision, Recall, F1-score, Accuracy
Aceto et al. [1]		Classification with Naive Bayes, Multinomial Naive Bayes, Random Forests, Support Vector, Decision Trees	Accuracy, Precision, Recall, F-measure
Aceto et al. [2]		Classification with Convolutional Neural Network	Accuracy, F-measure, Run Time Per-Epoch (RTPE)
Aiolli et al. [4]		Classification with Random Forests and Support Vector Machine	Precision, Recall, F1-score
Yao et al. [146]		CNN, Convolutional Capsule Network, Fully-connected Capsule Network, Long Short-Term Memory (LSTM)	Accuracy, F1-score, precision, recall
Ede et al. [130]		Semi-supervised fingerprinting with Clustering and cluster correlation	F1-score, Precision, Recall, Accuracy, Robustness
Xu et al. [143]		Classification using Graph Convolutional Neural network (GCN)	Accuracy, precision, recall, F1-score
Coull et al. [27]	Application Usage Identification	Classification with Binomial Naive Bayes	Accuracy
Conti et al. [22, 23]		Classification with Random Forests (RF)	F-measure, Accuracy, Precission, Recall
Saltaformaggio et al. [108]		Classification with SVM and Clustering with K-means	Detection time and True Positives, Misclassifications, False Negatives, False Positives, Precision, Recall
Fu et al. [45]		Classification with Gradient Boosted Trees, Support Vector Machine, Naive Bayes, KNeighbors	Accuracy, Precision, Recall, F-measure
Liu et al. [72]		Classification with Random Forests and Clustering with recursive Constrained KMeans	Accuracy, Precision, Recall, F-measure, Processing Throughput (pps)
Papadogiannaki et al. [90]		Frequent Pattern Mining for Signature Generation	True Positives, False Positives and Performance Throughput
Wang et al. [137]		Classification with Random Forests, Ada Boost and Gradient Boosting Decision Tree (GBDT)	Accuracy, Recall, Precision, F1-score
Jiang et al. [55]		Classification with Logistic Regression (LR), SVM, GBDT and RF	TPR, FPR and F1-score
Feng et al. [40]		Classification of traffic bursts using a LSTM model	FPR, recall, precision, processing delay
Wright et al. [140]	Stream Decoding	Hidden Markov Models (HMM)	Recall, Precision
Schuster et al. [110]		Gaussian distribution for Maximum Likelihood Estimator and Convolutional Neural Network (CNN)	Precision, Recall, Delay
Björklund et al. [13]		Classification with k-d tree over statistics and NN with Pearson's correlation	Accuracy, Tree building time, Training time, Identification time
Dimopoulos et al. [31]	QoS/QoE Investigation	Classification with Random Forests	True Positives, False Positives, Precision, Recall
Orsolic et al. [86]		Classification with Random Forest, Naive Bayes, SVM, Decision Trees	Accuracy
Mazhar et al. [77]		Classification with Decision Trees	Precision, Recall
Khokhar et al. [59]		Classification with RF, Linear Regression and RF Regression	Precision, Recall, F1-score, Accuracy, Root Mean Square Error (RMSE)
Xu et al. [144]		Fingerprint construction from chunk sizes	Accuracy
Wu et al. [142]		Fingerprint construction from sizes of mixed audio/video chunks	Accuracy, Processing time

classification, authors choose Multinomial Naive Bayes (e.g., [1]), Support Vector Machine (e.g., [126]) and Hidden Markov Models (e.g., [140]) algorithms as well as other classifiers, such as Random Forest, Decision Trees, Gaussian Naive Bayes [5] and the k-Nearest Neigh-

bors algorithm for pattern recognition [33]. For a more detailed classification (i.e, the identification of user actions and events inside mobile applications), authors choose hierarchical clustering techniques [23, 45].

Besides machine learning, it seems that recently, researchers have turned to neural networks since they perform better than single machine learning algorithms. A neural network combines different machine learning algorithms for modelling data using graphs of neurons, while it is able to make accurate decisions and learn from it own errors. This makes a neural network work independently without requiring any human intervention.

Being encrypted, network packet payloads do not offer significant information. Thus, most techniques discussed in this section take advantage of data that are available in packet headers. The majority of these techniques use information like network packet sizes, directions and time-related data, which are treated as features to train the corresponding machine learning models. More specifically, many works use as features the packet size and the packet direction [5, 16, 23, 27, 51]. Herrman et al. also use the IP packet length distribution [51]. Selecting a subset of packets in a single network flow is also common. For instance, Lu et al. do not consider incoming MTU packets [75], while Bernaille et al. keep only the first packets [12].

In Table 4.2, we also display the metrics used for the evaluation of each technique. Since the majority of these works are based on machine learning, they mostly measure their technique's (i) accuracy, (ii) precision, (iii) recall, (iv) true and (v) false positive rates. However, even though the majority of the works discussed in this section use similar evaluation metrics, we would not be able to properly compare their effectiveness since they use different datasets to train their systems. Finally, we notice that only few works present processing evaluation metrics, such as memory consumption, training time or throughput. In the next section (§ 4.1.1), we discuss about the datasets used for the works of traffic classification for network analytics.

Datasets Table 4.3 presents the public datasets that were used in the domain of network analytics. In the category of application identification and classification, Yao et al. [146] used the USTC-TFC2016 dataset [161] that among others contains network traffic from BitTorrent, Facetime, Gmail and Skype. Ede et al. [130] used the Recon dataset [164], which consists of labeled network traces of 512 Android apps from the Google Play Store, includ-

ing multiple versions for over a period of eight years.

We notice that the vast majority of the works in this section do not use public datasets to train their models. The proprietary datasets used come either by real or emulated usage representing network usage with applications of interest, either in mobile of fixed networks. The absence of public datasets that contain network traffic that is both encrypted and labeled is apparent. Thus, authors are producing their own datasets.

Table 4.3: Datasets used in the network analytics domain.

Work	**Category**	**Dataset Availability**	**Dataset details**
Yao et al. [146]	Protocol/Application Identification	Public	USTC-TFC2016 Dataset [161]
Ede et al. [130]		Public	Recon Dataset [164]
Xu et al. [143]		Available upon request	MAppGraph [166]
Feng et al. [40]	Application Usage Identification	Public	USTC-TFC2016 Dataset [161]

4.1.2 Objectives and Limitations

The majority of the works in the category of network analytics focus on the functionality of their approaches. More specifically, authors focus on the thorough examination of the traffic analysis feasibility when the network traffic is encrypted. Indeed, they show that it is possible to detect the nature of the traffic in a fine-grained manner. For instance, Conti et al. [22] are able to identify different actions (e.g., post a tweet or send a message) in mobile applications (such as Twitter and Facebook) accurately even when the network traffic is encrypted. Yet, there are works that except for functionality, they aim for programmability and deployment as well [72, 90]. OTTer achieves a detailed characterization of usage (i.e., video call, voice call, chat) in different Over-The-Top applications like Skype and WhatsApp [90]. It is also integrated into a DPI engine that is deployed in a live traffic test-bed with an average of 109 Gbps.

All the works that use machine learning algorithms for the classification of network traffic need to be retrained in order to remain robust across new and diverse data. In addition, the majority of such works choose a subset of applications or protocols in order to examine the feasibility of classification. This makes such solutions mostly effective only for applications and protocols used for training, something that introduces potential scalability and adaptability issues. Even while scoring high performance (with metrics like accuracy, precision and recall) in close-world scenarios (i.e., under a specific ground-truth

dataset), these systems will certainly produce high rates of false positives when tested against real-world datasets of traffic traces. Furthermore, numerous countermeasures exist that can perform against traffic analysis techniques (such countermeasures are discussed in Section 5), making many of the works discussed in this section unable to bypass them.

4.1.3 Relation to this Dissertation

Most relevant to our work in the domain of network analytics (§ 2.1) is the literature on fine-grained application event identification over encrypted traffic, reviewed in this section (§4.1). While our work is based on the same grounds (i.e., the feasibility of user activity identification over encrypted network traffic based on packet trains), we advance the state-of-the-art by (i) proposing a novel expressive pattern language specification, (ii) building a scalable and optimized implementation, which was integrated to our proprietary DPI engine and tested and evaluated on real-world traffic volumes, (iii) showing that the rule extraction is amenable to data mining techniques.

4.2 Security after Network Encryption

In this section, we present the state-of-the-art on encrypted network traffic analysis for network security. The works that are presented in this section are summarized in Table 4.4.

Intrusion Detection Some techniques focus on identifying malicious behavior in the network, examining the characteristics of the underlying traffic, using exclusively machine learning approaches.

Taleb et al. [38, 123] propose an approach that identifies misuses in encrypted protocols with network packet inspection that focuses on processing of packet header information. Amoli et al. present a real-time unsupervised NIDS, able to detect new and complex attacks within encrypted and plaintext communications [6]. Anderson et al. [7] compare the properties of six different machine learning algorithms for encrypted malware traffic classification. Shone et al. [116] propose a system that combines deep learning techniques for network intrusion detection. For the evaluation of their system, authors use the KDD Cup '99 and NSL-KDD datasets with high accuracy (almost 90%). Kitsune [78] is a NIDS,

Table 4.4: Works in the security domain, sorted by category and publication year.

Traffic Analysis Domain	Work	Category	Goal	Year of publication
Network Security	Amoli et al. [6]	Network Intrusion Detection	Real-time network intrusion detection within encrypted communications	2016
	Anderson et al. [7]		Encrypted malware traffic classification	2017
	Shone et al. [116]		Network intrusion detection using a combination of DL techniques	2018
	Mirsky et al. [78]		Neural-network based network intrusion detection system	2018
	Papadogiannaki et al. [89, 91, 95]		Intrusion detection with signatures from packet metadata sequences	2020–2023
	Fu et al. [44]		Malicious traffic detection via flow interaction graph analysis	2023
	Tang et al. [125]	Network Intrusion Detection (SDN)	Flow-based anomaly detection	2016
	Niyaz et al. [85]		DDoS attack identification	2016
	Shabtai et al. [111]	Malware Detection on Mobile Devices	Malware detection on Android mobile devices	2012
	Shabtai et al. [112]		Identification of malicious attacks or masquerading/injected mobile applications	2014
	Wang et al. [135]		Detection of mobile malware behavior using network traffic	2016
	Lashkari et al. [65]		Detection of malicious or masquerading mobile applications	2017
	Razaghpanah et al. [101]	Server Characterisation	TLS usage examination in Android devices	2017
	Anderson et al. [8]		TLS fingerprinting for malware identification	2019
	Kotzias et al. [61]		TLS deployment examination	2018
	Paracha et al. [96]		TLS usage examination in consumer IoT devices	2021
	Li et al. [69]		TLS fingerprinting for bot classification	2021
	Sosnowski et al. [121, 122]		TLS fingerprinting for CnC server classification	2022 – 2023

based on neural networks, and designed for the detection of abnormal patterns in network traffic. It monitors the statistical patterns of recent network traffic and detects anomalous patterns. Tang et al. [125] present a deep learning approach for flow-based anomaly detection in SDN environments. Authors build a Deep Neural Network (DNN) model for an intrusion detection system and train it with the NSL-KDD dataset, using six basic features of the NSL-KDD dataset. Niyaz et al. [85] utilize deep learning in order to detect DDoS attacks in SDN environments. The proposed system identifies individual DDoS attacks with an accuracy of almost 96% and classifies the traffic into benign or attack traffic, with an accuracy of 99.82% with low false-positives. In [44], authors propose HyperVision, a system that is able to perform anomaly detection against encrypted network traffic using an unsupervised graph learning method. HyperVision achieves 0.92 AUC and 0.86 F1, with 80.6 Gb/s detection throughput and average detection latency of 0.83s.

Intrusion and Malware Detection in Mobile Devices While the ever increasing adoption of traffic encryption has significantly improved the user privacy and security, traditional

intrusion detection systems based on inspecting unencrypted traffic, are becoming obsolete. Thus, there is a number of works that aim to detect malicious bahavior on mobile devices, mainly relying on machine learning approaches.

Andromaly [111] is a framework for malware detection on Android mobile devices. The host-based malware detection system monitors features and events that are retrieved from mobile devices and applies anomaly detection for the classification of the collected data. Shabtai et al. [112] propose a system that identifies (i) attacks or masquerading applications installed on a mobile device and (ii) injected applications with malicious code. In TrafficAV [135], the mobile network traffic is mirrored from the wireless access point to the server for data analysis. The data analysis and malware detection are performed on the server side. TrafficAV performs network traffic analysis in multiple levels. The proposed method combines network traffic analysis with a machine learning algorithm (i.e., C4.5 decision tree) and is able to identify malware in Android devices with good accuracy results. In an evaluation with 8,312 benign apps and 5,560 malware samples, the TCP flow detection model and the HTTP detection model achieve detection rates of up to 98% and 99.65%, respectively. Lashkari et al. [65] detect malicious and masquerading applications on mobile devices. Their proposed method shows a good average accuracy (91.41%) and precision (91.24%) with a low false positive rate. Authors use five different classifiers: Random Forest (RF), K-Nearest Neighbor (KNN), Decision Tree (DT), Random Tree (RT) and Regression (R). Authors have published a labeled dataset of mobile malware traffic that contains benign Android applications or injected applications (e.g., with adware or other types of malware).

Characterization of Identity In [8], authors publish a knowledge database consisting of TLS fingerprints (passively constructed) from enterprise deployments and malware traffic, together with an analysis of trends concerning the utilization of TLS by applications and malware. TLS fingerprinting is a common technique that assists in the extraction of meaningful observations from TLS handshake packets, since limited content can be revealed from encrypted data packets. TLS fingerprinting has been used for studying the TLS deployment [61] and the TLS usage in consumer IoT devices [96] and Android applications [101]. TLS fingerprinting has been also used in identifying known censorship circumvention tools, like Tor [43]. JA3 is a method that enables TLS fingerprinting and TLS client

profiling, while JA3S enables server side TLS fingerprinting [183]. Another library that enables TLS handshake fingerprinting is fingeprinTLS [181]. In a nutshell, the produced fingerprints are a combination of TLS version, accepted ciphers, extensions, elliptic curves and elliptic curve formats. Since JA3 and JA3S hashes (or JA3/JA3S fingerprints) can be easily distributed, they promote the easy cyberthreat intelligence exchange. In fact, support to JA3S has been added in platforms like MISP [209] and intrusion detection systems like Suricata [202]. In [8], for instance, authors produce fingerprints for the identification of malicious servers. Li et al. produce fingeprinTLS to distinguish malicious bots from benign [69]. Besides these two passive techniques for TLS fingerprinting (i.e., fingeprinTLS and JA3S), there is JARM [194]. JARM enables *active* TLS server fingerprinting. In an article posted online on October 2020 [192], JARM creators explain how it works and how it can be used to identify malicious servers. In the article, creators made public 1 fingerprint per botnet. The botnets that were examined are TrickBot, AsyncRAT, Metasploit, Cobalt Strike and Merlin C2 and as authors state, they contact IP addresses on port 443. Recently, Sosnowski et al. proposed a new TLS fingerprinting technique [122]. Authors use a binary classifier to investigate if a server is a command and control server, with better precision and recall results when compared to JARM. In addition, authors study *weekly* snapshots in a period of 7 months. In DissecTLS [121], authors follow a more exhaustive fingerprinting approach to calculate more effective fingerprints, which outperform five popular TLS scanners (including JARM). They perform a measurement study using the same public datasets that we use, and they repeat the measurements nine times in a period of nine weeks.

4.2.1 Techniques

In this section, we present the techniques and algorithms that are more popular among the works of this category. Table 4.5 displays works that perform intrusion detection even in encrypted networks. The majority of these works utilize machine learning algorithms for intrusion detection and classification.

Besides machine learning, researchers make use of neural networks and deep learning techniques. For intrusion detection, Shone et al. [116] propose a deep learning classification model constructed using stacked non-symmetric deep autoencoders (NDAEs).

Table 4.5: Techniques, algorithms and evaluation metrics used in the security domain.

Work	Category	Algorithm/Technique	Performance Evaluation Metrics
Amoli et al. [6]	Network Intrusion Detection	DBSCAN-based outlier detection, K-means-based outlier detection	FPR, TPR, Accuracy, Precision, Recall
Anderson et al. [7]		Classification with Linear Regression, Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, Multi-layer Perceptron	Accuracy, Classification Time
Shone et al. [116]		Auto-encoder Deep Neural Network	Accuracy, Precision, Recall, F-measure, False alarm, Training/testing Time
Mirsky et al. [78]		Isolation Forests (IF) and Gaussian Mixture Models and Deep Neural Network Auto-encoders with Ensemble Layer and Output Layer	TPR, FNR & Processing Throughput
Papadogiannaki et al. [89, 91, 95]		Signature generation for Intrusion Detection using packet metadata sequences	True Positives, False Positives and Performance Throughput
Fu et al. [44]		Graph learning module based on DBSCAN and K-means for clustering and Z3 SMT Solver to identify critical vertices	AUC, F1-score, Precision, Recall, F2, ACC, FPR, EER & Throughput, Latency, Resource Consumption
Tang et al. [125]	Network Intrusion Detection (SDN)	Deep Neural Network with an input layer, three hidden layers and an output layer	Accuracy, Precision, Recall and F-measure
Niyaz et al. [85]		Stacked Autoencoder Deep Neural Network	Precision, Recall, F-measure
Shabtai et al. [111, 112]	Malware Detection on Mobile Devices	Linear Regression, Decision Table, Support Vector Machine for Regression, Gaussian Processes for Regression, Isotonic Regression, Decision/Regression tree (REPTree)	TPR, Detection time, FPR, False Alerts, Memory & CPU consumption
Wang et al. [135]		Classification with C4.5 Decision Tree	TPR, FPR
Lashkari et al. [65]		Classification with Random Forest, K-Nearest Neighbor, Decision Tree, Random Tree, Regression.	Accuracy, Precision, FPR
Razaghpanah et al. [101]	Server Characterisation	Fingerprinting using passive analysis	N/A (measurement study)
Kotzias et al. [61]		Fingerprinting using passive analysis	N/A (measurement study)
Anderson et al. [8]		Fingerprinting using passive analysis	N/A (measurement study)
Paracha et al. [96]		Fingerprinting using passive analysis	N/A (measurement study)
Li et al. [69]		Fingerprinting using passive analysis	N/A (measurement study)
Sosnowski et al. [121, 122]		Active probing for TLS fingerprinting, Multi-label classifier for CDN server classification, Binary classifier for CnC server detection	Precision, Recall

Tang et al. [125] present a deep learning approach for flow-based anomaly detection in SDN environments. Authors build a Deep Neural Network (DNN) model for intrusion detection and train it with the NSL-KDD dataset, using basic features found in the dataset. Niyaz et al. [85] utilize deep learning in order to detect DDoS attacks in SDN environments. Kitsune [78] is a NIDS, based on neural networks, and designed for the detection of abnormal patterns in network traffic. It monitors the statistical patterns of recent network traffic and detects anomalous patterns. Again, the majority of these techniques take advantage of data that are available in packet headers like network packet sizes, directions and time-related data. More sophisticated techniques make use of additional, more advanced features, like the TLS handshake information [7, 43]. HyperVision [44] does not require a labeled dataset of know attacks, since it detects abnormal interaction patterns by analyzing connectivity, sparsity and statistical features of the graph.

In Table 4.5, we also display the metrics used for the evaluation of each technique. Similarly to Section 4.1, the most common evaluation metrics are (i) accuracy, (ii) precision, (iii) recall, (iv) true and (v) false positive rates, while few of the works report classification time and processing throughput.

Datasets Table 4.6 presents the public datasets that were used for intrusion detection. The DARPA dataset [151] is one of the most popular datasets to train and evaluate intrusion detection systems. The ISCX-IDS-2012 intrusion detection dataset [172] consists of 7 days of benign and malicious network activity. The KDD Cup '99 dataset [152] includes a wide variety of intrusions simulated in a military network environment. The NSL-KDD dataset [156] is suggested to solve some of the inherent problems of the KDD'99 dataset. Finally, Moustafa et al. [80] build and make publicly available a handy dataset for network intrusion detection systems, namely UNSW-NB15 [150]. This dataset is used for evaluation in [89,91]. The IoT-23 [171] is a labeled dataset with malicious and benign IoT network traffic and it is used in [95]. The dataset used in [44] is the MAWI Internet traffic dataset. To avoid any bias, authors also use the Kitsune dataset [78], as well as the CIC-DDoS2019 and CIC-IDS2017 datasets [162, 165].

As Anderson et al. [7] correctly points out, finding the most proper features for classification with high accuracy, recall and precision, is not a trivial procedure, while it is highly dependable on the ground-truth dataset collection that is available each time. However, user privacy related regulations make data sharing a very challenging practice for organizations and data holders. In Table 4.6, we can notice that the NSL-KDD dataset [156] is the most popular dataset that is publicly available. Yet, we observe that the majority of the datasets that are used for intrusion detection are not very recent. For instance, Amoli et al. [6] and Shone et al. [116] use the DARPA [151] and KDD Cup '99 [152] datasets respectively, both of which were more than 15 years old at the time the papers were published.

Table 4.6: Datasets used in the network security domain.

Work	**Category**	**Dataset Availability**	**Dataset details**
Amoli et al. [6]	Network Intrusion Detection	Public	DARPA [151], ISCX-IDS-2012 [172]
Shone et al. [116]		Public	KDD Cup '99 [152], NSL-KDD [156]
Papadogiannaki et al. [89, 91]		Public	UNSW-NB15 [150]
Papadogiannaki et al. [95]		Public	IoT-23 [171]
Tang et al. [125]		Public	NSL-KDD [156]
Fu et al. [44]		Public	MAWI Internet traffic dataset [169], CIC-DDoS2019 [165], CIC-IDS2017 [162]
Kotzias et al. [61]	Server Characterisation	Public	ICSI SSL Notary [157], fingerprinTLS [159]
Anderson et al. [8]		Public	Mercury [167]
Li et al. [69]		Public	Upon request

4.2.2 Objectives and Limitations

As most works that perform encrypted network traffic inspection for intrusion detection use machine learning techniques to examine the feasibility of attack classification, the primary objective is functionality. They train their models offline providing poor support for online intrusion detection when it comes to encrypted communications. The works that also aim for programmability, except for functionality, are [78, 89, 91, 111, 112]. In Section 4.4, we discuss about middleboxes that can be deployed for online traffic inspection. Some of the network functions of these middleboxes can be used for intrusion detection and firewall applications.

The majority of the works discussed in this section can be bypassed by traffic analysis resistant techniques. Anderson et al. [7] examine and present mistakes and limitations in the network traffic analysis literature, such as the utilization of old and unreliable ground-truth datasets. Indeed, if we examine the datasets that are used as ground-truth for training most of the machine learning models in this section, are not very recent.

4.2.3 Relation to this Dissertation

Most relevant to our work in the domain of network security (§ 2.2) is the literature on intrusion detection over encrypted traffic, surveyed in this section (§4.2). The majority of works that inspect encrypted network traffic turn to machine learning algorithms since it is possible to examine the *feasibility* of the identification of the traffic nature and the underlying activities for numerous use-cases (e.g., network security). Other works perform anomaly detection to identify malicious network traffic with abnormal traffic patterns, without the requirement of labeled ground-truth datasets. Such works, though, do not indicate the specific malicious activity in a fine-grained manner, just the abnormality. Our work builds on the results of those works (i.e., intrusion detection over encrypted traffic), but our main concern is to establish a procedure to effectively generate intrusion detection signatures in an automated manner and integrate them in a real-world traffic monitoring system. In our work (§ 2.2), we propose a signature mining method for intrusion detection in encrypted network traffic. Also, we develop a network intrusion detection system that achieves high processing throughput. Specifically, we aim to advance the state-of-the-art offering an intrusion detection implementation that combines the fol-

lowing: (i) we generate signatures from packet metadata, found exclusively in network packet headers, (ii) we implement a signature-based intrusion detection engine using an extended version of the Aho-Corasick algorithm to support integers, (iii) we enhance our system's performance using accelerators.

Besides our approach proposed for intrusion detection (employing passive network traffic analysis), we also engage in the characterization of malicious servers using active probing. The literature that examines the fingerprintability of malicious entities/servers (reviewed in § 4.2) is the most related to our work presented in Chapter 3. The goal of our work is to study the evolution of known command and control servers obtained by public blocklists with TLS fingerprinting. When compared to other measurement studies that perform TLS fingerprinting, our work produces a significantly larger dataset (fingerprints are constructed on a daily basis for a period of 7 months).

4.3 User Privacy after Network Encryption

Besides network analytics and security, traffic analysis has been also used to monitor and profile the characteristics of mobile applications. This kind of tools empower the user to gain (i) insight regarding the applications used and (ii) control over personally identifiable information (PII) handling. Furthermore, in this section we discuss about works that are able to fingerprint websites and applications even when anonymity tools like TOR [205] are used to hide the activity's nature, the user's identity or the user's location. Finally, we present works that enable OS and device identification even in encrypted networks. The works that are presented in this section are summarized in Table 4.7.

Endpoint Device Tools and PII Leakage Detection ProfileDroid [138] is a monitoring and profiling system that can characterise the behaviour of Android applications at the static, user, OS and network layers. The authors evaluate multiple Android applications, which present privacy and security, operational and performance issues. Authors evaluate the proposed tool using free and paid Android applications, observing (i) discrepancies between the app specification and app execution, (ii) higher costs resulted from free versions of apps, due to an order of magnitude increase in traffic, (iii) a great amount in network traffic that is not encrypted, (iv) excess communication with more than expected sources,

Table 4.7: Works in the user privacy domain, sorted by category and publication year.

Traffic Analysis Domain	Work	Category	Goal	Year of publication
User Privacy	Herrmann et al. [51]	Webpage Identification	Webpage identification through OpenSSH, OpenVPN, CiscoVPN, Stunnel, TOR, JonDonym	2009
	Lu et al. [75]		Webpage identification through SSH and SSL tunnels	2010
	Panchenko et al. [87,88]		Webpage identification through TOR	2011
	Cai et al. [16]		Webpage identification through TOR and HTTPOS	2012
	Kwon et al. [62]		Webpage identification through TOR hidden services	2015
	Draper-Gil et al. [33]		Detection and characterization of VPN traffic	2016
	Cruz et al. [28]		Identification of BitTorrent traffic in SSH tunnels	2017
	Lotfollahi et al. [74]		Application identification in VPN traffic	2017
	Shahbar et al. [113]		Identification of anonymity networks	2017
	Montieri et al. [79]		Traffic classification of anonymity tools	2019
	Jorgensen et al. [56]		Application identification in VPN traffic	2023
	Chen et al. [21]	Device/OS Identification	OS identification, NAT and tethering detection	2014
	Sivanathan et al. [117]		IoT device identification using traffic features	2018
	Skowron et al. [118]		Investigation of fingerprinting attacks targeting IoT devices	2020
	Lastovicka et al. [66]		OS identification using TLS fingerprints	2020
	Ateniese et al. [10]	Location Estimation	Position extrapolation through location-based encrypted traffic	2015
	Razaghpanah et al. [102]	PII Leakage Detection	Detection of personal information leakage in Android applications	2015
	Song et al. [120]		Detection of information leakage in Android applications	2015
	Le et al. [67]		Traffic collection and analysis to enable users have control over their data	2015
	Ren et al. [103]		Cross-platform PII leaks identification giving users control over them	2016
	Continella et al. [24]		PII leakage detection, resilient to obfuscation techniques	2017
	Rosner et al. [106]		Information leaks in TLS-encrypted network traffic	2019

and finally (v) the communication of the majority of the examined apps with Google. TaintDroid [37] identifies privacy leaks of Android applications with dynamic information-flow tracking. More specifically, the authors monitored the behavior of popular third-party Android applications and discovered potential misuse cases of user private information across applications. TaintDroid provides users with information regarding third-party applications. Haystack [102] is a mobile application distributed via popular app stores that can correlate contextual information (such as application identifiers and radio state) with specific traffic flows (encrypted or not) destined to remote services. To handle encrypted

traffic, haystack employs a transparent man-in-the-middle (MITM) proxy for TLS traffic. PrivacyGuard [120] is an open-source platform that intercepts the network traffic that is generated by mobile applications (through VPN) in order to detect sensitive information leakage. PrivacyGuard can is able to effectively detect information leakage in the majority of the applications under examination and it is shown that it outperforms TaintDroid. AntMonitor [67] is a system that passively monitors and collects packet-level measurements from Android devices in order to provide a fine-grained analysis. AntMonitor provides users with control over their personal data and supports client-side traffic collection and analysis. Authors examine PII-related features, such as (i) the IMEI, which uniquely identifies a device within a mobile network, and Android Device ID, which is an identification code associated with a device, and (ii) the phone number, email address and location, which can be uniquely associated with users. ReCon [103] is a cross-platform system that reveals personally identifiable information leakages and gives users the control over leaked information without requiring any special privileges or custom OSes. ReCon uses machine learning to identify and reveal possible PII leakage by inspecting the network traffic, while it provides a visualization tool that empowers users to control how their information is being handled by blocking or substituting it. Continella et al. [24] propose a system that detects leakages of PII and is resilient to obfuscation methods and techniques (e.g., encoding, formatting), encryption, or any other kind of transformation performed on private information before it is leaked. Moreover, Rosner et al. [106] present a black-box approach for detecting and quantifying side-channel information leaks in TLS-encrypted network traffic.

Fingerprinting In this section, we present works in the field of webpage and application fingerprinting, even while covered by privacy enhancing technologies. In addition, we study the state-of-the-art in the device/OS identification and location estimation from network traffic inspection. The works that are presented in this section are also summarized in Table 4.7.

Traffic analysis has been used to identify fine-grained information, like webpages and websites, even while transferred over encrypted tunnels established by technologies such as OpenVPN [200] or TOR [205]. Herrmann et al. [51] present a classifier that identifies up to 97% of web requests from packet traces. Lu et al. [75] identify dynamic websites that

are transferred over SSH and SSL tunnels. Panchenko et al. [87, 88] show how website fingerprinting in onion routing based anonymization networks, such as TOR, is still possible using information like packet sizes, total transmitted bytes, percentage of incoming packets and others. Cai et al. [16] present a webpage fingerprinting attack that is resilient to recent traffic analysis countermeasures' methods, such as application-level defenses like HTTPOS [76] and randomized pipelining over TOR [97]. Kwon et al. [62] perform a passive attack against hidden services and their users called circuit fingerprinting attack. Using the attack, an attacker can identify the presence of hidden service activity in the network with high accuracy. Draper-Gil et al. [33] study flow-based and time-related features that can be analyzed to detect VPN traffic and to classify encrypted traffic according to the type of traffic (e.g., browsing or streaming) using two machine learning techniques to test the accuracy of the features. They show that time-related features, when properly handled can reveal enough information for encrypted traffic characterization. Cruz et al. [28] present a deep learning method that takes advantage of a feature set that is based on the statistical behavior of TCP tunnels proxying BitTorrent traffic. The next steps is the transformation of this feature set into multiple timestep sequences and the training of a recurrent neural network. The results of this work show that it is possible to identify the existence of BitTorrent traffic in SSH tunnels. Lotfollahi et al. [74] present a system that is able to handle both traffic characterization and application identification by analysing encrypted traffic with deep learning. The proposed scheme can characterize between VPN and non-VPN traffic, protocols (e.g., FTP or P2P) and end-user applications (e.g., Skype or BitTorrent). Shahbar et al. [113] identify multilayer-encryption anonymity networks and the obfuscations techniques they use with a small number of features and a small number of packets. Montieri et al. [79] perform hierarchical traffic classification of anonymity tools, like TOR.

Device/OS Identification There are many works that focus on extracting TCP or IP packet metadata, in order to investigate if the behavior of specific packet contents can be correlated with OSes, device types and other characteristics.

Chen et al. [21] are able to perform OS identification, NAT and tethering detection by examining multiple features in the TCP/IP packet headers. The authors use real network traffic traces to evaluate the accuracy of fingerprinting and show that several techniques that can successfully fingerprint desktop OSes are not similarly effective for fingerprint-

ing mobile devices. For OS fingerprinting, authors use the following packet header values: the IP TTL value, the IP ID monotonicity, the TCP timestamp option, the TCP window size scale option and the clock frequency. Features like the TCP timestamp monotonicity, clock frequency and boot time can be used for tethering detection. Ruffing et al. [107] aim for OS identification of mobile devices even in encrypted traffic. Authors propose a traffic content agnostic algorithm that implements spectral analysis of the encrypted traffic and they show that even a network traffic input of 30 seconds can be enough for high accuracy results. Sivanathan et al. [117] classify IoT devices using network traffic features. More specifically, authors instrument an ecosystem of different IoT devices (e.g., cameras, plugs and motion sensors) and examine traffic characteristics, such as port numbers, signalling patterns, and cipher suites that are used. Lastovicka et al. [66] examine traffic patterns of the TLS protocol and train a machine learning model using features from the TLS handshake in order to identify the operating system of a device. They focus their research on mobile devices connected in a wireless network.

Location Estimation The position of a mobile device can be calculated and estimated by collecting and monitoring the network traffic that is produced by applications that contain location-based services, even when the communication channels are encrypted. For example, Ateniese et al. [10] show that an adversary could estimate the position of a mobile device by analysing the timing and sizes of encrypted network packets that are exchanged between the user's mobile device and any location-based service provider that communicates with the device.

4.3.1 Techniques

In this section, we present the techniques and algorithms that are more popular among the works of this category. Table 4.8 briefs the techniques and evaluation metrics used.

The majority of works in the category of website or device/OS fingerprinting domain use machine learning techniques, while it is also common to build fingerprints for a webpage and then compare its similarity for classification. Works that detect PII leakages either work offline (e.g., [67, 106]) or online (e.g., [24, 102]). Offline works use similar machine learning techniques as previously discussed works, while online tools intercept the encrypted network traffic before processing it.

For instance, between works that examine tunnelled network traffic (e.g., over VPN or SSH protocols) for website classification and fingerprinting, the most popular algorithms are Multinomial Naive Bayes [51], Support Vector Machine [88] and Hidden Markov Models [16]. In addition, Levenshtein distance and the Jaccard classifier are used in a number of works to examine similarities between website fingerprints and properly classify them into categories [16, 71, 75].

In addition, more recent works like [28, 74] use neural networks. Lotfollahi et al. [74] present a system that is able to handle both traffic characterization and application identification by analysing encrypted traffic with deep learning, embedding stacked autoencoder and convolution neural network (CNN) to classify network traffic. Cruz et al. [28] identify tunnelled BitTorrent traffic with a deep learning implementation. Their approach examines features that are related to the statistical behaviour of TCP tunnels proxying BitTorrent traffic. Then, authors transform the features into multiple time sequences and train a recurrent neural network. Profit combines techniques like network trace alignment, phase detection, feature selection, feature probability distribution estimation and entropy computation to quantify the amount of information leakage that is revealed via the network traffic. Rosner et al. [106] present in "Profit" a dynamic technique to detect information leakages in applications that support encryption and communicate via TLS. Profit receives a "user-supplied profiling-input suite" where application data is annotated as secret or sensitive. Profit runs the application over the user-supplied input and captures a set of variable-length network packet traces. The traces include information like packet sizes and timestamps along with their aggregations (e.g., total time and median size). Again, authors agree that finding the features that leak the most information is challenging. To another end, device and OS identification techniques use network packet header contents, such as IP TTL value, the IP ID monotonicity, the TCP timestamp option, the TCP window size scale option, and the clock frequency. For the detection of tethering, similar approaches take into consideration the monotonicity of the TCP timestamp, the timestamp clock frequency and the booting time [21].

Works that investigate PII leakage from mobile applications are also presented in Tables 4.7 and 4.8. In this category, works tend to perform network traffic interception to be able to process the encrypted traffic and follow the information flow that is exposed by mobile applications. After the traffic interception, authors are able to extract information

by inspecting plaintext packet contents.

Table 4.8: Techniques, algorithms and evaluation metrics used in the privacy domain.

Work	**Category**	**Algorithm/Technique**	**Performance Evaluation Metrics**
Herrmann et al. [51]	Webpage Identification	Classification with Multinomial Naive Bayes (MNB)	Accuracy, Training/testing Time
Lu et al. [75]		Fingerprint similarity with Levenstein distance	Accuracy, Effect of time on accuracy
Panchenko et al. [87, 88]		Classification with SVM, Fingerprint similarity	Accuracy, FPR, TPR, Runtime, Recall, Precision
Cai et al. [16]		Classification with SVM and Fingerprint similarity with Damerau-Levenshtein distance	Success rate, Likelihood, TPR
Kwon et al. [62]		Classification with C4.5 Decision Trees and Fingerprint similarity with Edit Distance	Accuracy, TPR, FPR
Draper-Gil et al. [33]		Classification with C4.5 and KNN	Precision, Recall, Accuracy
Cruz et al. [28]		LSTM and BLSTM Deep Neural Networks	Precision, Recall, Accuracy, F1-score
Lotfollahi et al. [74]		Deep Neural Network Stacked Autoencoder (SAE) and Convolution Neural network (CNN)	F1-score, TPR, FPR
Shahbar et al. [113]		Classification with C4.5 Decision Trees, Random Forests, Naive Bayes, Bayesian Network (BN)	Accuracy, Time
Montieri et al. [79]		Classification with C4.5, RF, NB, BN	Accuracy, F-measure, G-mean
Jorgensen et al. [56]		Classification with a NN and out-of-distribution (ODD) scores	Accuracy, F1-score
Chen et al. [21]	Device/OS Identification	Classification with Naive Bayes	Accuracy, Precision, Recall, F-measure
Sivanathan et al. [117]		Classification with Naive Bayes Multinomial classifier, Random Forest,	Accuracy, Root Relative Squared Error (RRSE)
Skowron et al. [118]		Classification with k-NN, Decision Trees and Random Forests	Accuracy, F1-score, Precision, Recall
Lastovicka et al. [66]		Classification with Decision trees	Accuracy, Precision, Recall, F-measure
Ateniese et al. [10]	Location Estimation	Interception of encrypted traffic and payload inspection	Accuracy and Granularity of the monitor area
Razaghpanah et al. [102]	PII Leakage Detection	Interception of encrypted traffic and payload inspection	CPU and Power Overhead, Latency, Throughput, TLS overhead
Song et al. [120]		Interception of encrypted traffic and payload inspection	Throughput, Delay, Battery Consumption
Le et al. [67]		Interception of encrypted traffic and payload inspection, Classification with SVM	Precision, F1-score, CPU and Battery cost
Ren et al. [103]		Interception of encrypted traffic and payload inspection	Accuracy, Classification time and Runtime
Continella et al. [24]		Interception of encrypted traffic and payload inspection	False positives, Execution time
Rosner et al. [106]		Trace Alignment, Phase Detection, Leakage Quantification with Shannon entropy	Information Leakage measure

Datasets Table 4.9 contains the public datasets that were used by each work. Lastovicka et al. [66] produced a dataset that was made public after the publication of their work. The dataset contains TLS fingerprints collected. The VPN-nonVPN dataset (ISCXVPN2016)

dataset [160] contains traffic from user sessions in applications of browsing, email, chat, streaming and others. The traffic is either regular or transferred over VPN. The LBNL/ICSI dataset [153] contains packet traces that span more than 100 hours of activity from a total of several thousand internal hosts. The Anon17 dataset [163]contains network traffic traces from TOR, JonDonym and I2P anonymity tools. The CRAWDAD SIGCOMM'08 dataset [155] contains traces of wireless network activity from the SIGCOMM 2008 conference. Similarly, the CRAWDAD OSDI'06 dataset [154] contains network activity from the OSDI 2006 conference. Finally, the goal of the DARPA Space/Time Analysis for Cybersecurity program [158], among others, is to enable researchers to identify vulnerabilities related to the space and time resource usage behavior of algorithms.

Table 4.9: Datasets used in the user privacy domain.

Work	Category	Dataset Availability	Dataset details
Lotfollahi et al. [74]	Webpage Identification	Public	UNB ISCX VPN-nonVPN [160]
Shahbar et al. [113]		Public	LBNL/ICSI [153]
Montieri et al. [79]		Public	Anon17 [163]
Jorgensen et al. [56]		Public	VPN/NONVPN Network Application Traffic Dataset (VNAT) [170]
Chen et al. [21]	Device/OS Identification	Public	CRAWDAD SIGCOMM'08 [155], CRAWDAD OSDI'06 [154]
Lastovicka et al. [66]		Proprietary dataset that was made public	TLS fingerprints for OS identification [168]
Rosner et al. [106]	PII Leakage Detection	Public	DARPA Space/Time Analysis for Cybersecurity program [158]

4.3.2 Objectives and Limitations

In the categories of website, location and device/OS identification, the major goal is to produce an effective methodology for accurate fingerprinting. Thus, all the works that we study in these categories have one primary target; functionality. On the other hand, works in the domain of PII leakage Detection focus on the programmability and deployability. Hence, we encounter performance-driven solutions.

The limitation of high false positive rates is major in any domain of traffic analysis. Unfortunately, it appears to be very difficult to produce a universal solution that will be able to cover a whole domain, due to the vast diversity and heterogeneity that has introduced during the recent years in every single aspect of Internet. In addition, as Juarez et al. [57] argue regarding fingerprinting, accuracy scores reduce over time.

In addition, in the category of PII leakage detection, it is common to use proxy servers or VPN in order to redirect the traffic to a controlled environment for interception and decryption of encrypted traffic. Even if we neglect the latency that is added, this technique could eventually raise privacy-related concerns if users are not properly informed about the procedure of the processing and manipulation (e.g., storage) of their traffic and personal data. In Section 4.4, we discuss about middleboxes that address this issue by processing sensitive information using hardware-assisted technologies for trusted execution.

4.3.3 Relation to this Dissertation

Although the works presented in this section (§ 4.3) are not rigidly related to the work realized during this dissertation, we include them for completeness, since they lie into the domain of encrypted traffic analysis, as well.

4.4 Network Functions in Middleboxes after Network Encryption

Quoting from RFC 3234, "a middlebox is defined as any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host" [18]. The typical use of a middlebox is to offer security (e.g., firewall, intrusion detection) or performance (e.g., caching, protocol accelerator), while other common uses are network address translation and protocol conversion. One challenge that occurred after the rapid growth of network encryption is that in order to process and operate on TLS traffic the middlebox must perform a man-in-the-middle in a connection. Of course, this raises major concerns on user privacy preservation. Network middleboxes that aim to inspect encrypted traffic operate by acting as proxies. They terminate and decrypt the client-initiated TLS session, they analyze the HTTP plaintext content, and then they initiate a brand new TLS connection to the destination. TLS makes interception difficult by encrypting data and defending against attacks (like man-in-the-middle) via the certificate validation. During the validation process, the client is responsible to authenticate the identity of the destination server, rejecting impostors. To circumvent this validation process, a self-signed CA certificate is injected into the client browser's root store at the time of installation. For network middleboxes, administrators deploy the

middlebox certificate to the corresponding devices (e.g., of the organization) in a similar manner. Then, when the proxy intercepts a connection, it will dynamically generate a certificate for a website's domain name that is signed with its CA certificate. The proxy will deliver this certificate chain to the browser [35]. Some works that allow TLS interception to inspect encrypted network traffic have already been presented in Section 4.3. In this section, we discuss about the works that enable secure processing of encrypted traffic by middleboxes. Tables 4.10–4.11 present details about these works.

Table 4.10: Works that implement network functions in midlleboxes, sorted by publication year.

Traffic Analysis Domain	Work	Goal	Year of publication
Network Functions in Middleboxes	Sherry et al. [115]	DPI on the encrypted traffic using encrypted rules	2015
	Naylor et al. [82]	Extend the TLS protocol to support middleboxes	2015
	Asghar et al. [9]	Trusted execution for Network Functions (NF) in the cloud	2016
	Canard et al. [17]	DPI on the encrypted traffic using encrypted rules	2017
	Lan et al. [64]	Trusted execution for Network Functions (NF) in the cloud	2016
	Yuan et al. [148]	DPI on the encrypted traffic using encrypted rules	2016
	Fan et al. [39]	DPI on the encrypted traffic using encrypted rules	2017
	Naylor et al. [81]	Secure outsourcing of middlebox NF in untrusted infrastructures	2017
	Han et al. [50]	Secure outsourcing of middlebox NF in untrusted infrastructures	2017
	Coughlin et al. [26]	Secure outsourcing of middlebox NFV in untrusted infrastructures	2017
	Poddar et al. [98]	Secure outsourcing of middlebox NFV in untrusted infrastructures	2018
	Trach et al. [128]	Secure outsourcing of middlebox NF in untrusted infrastructures	2018
	Goltzsche et al. [46]	Secure virtual private network (VPN) with middlebox NF	2018
	Duan et al. [34]	Secure outsourcing of middlebox NF in untrusted infrastructures	2019
	Ning et al. [83]	DPI on the encrypted traffic using encrypted rules	2019
	Guo et al. [48]	Privacy preserving packet header processing for middleboxes in the cloud	2020

Network Functions in Middleboxes BlindBox [115] performs deep-packet inspection directly on the encrypted traffic, utilizing a new protocol and new encryption schemes. Specifically, the functionality of BlindBox is provided through (i) a searchable encryption scheme [119] that enables the inspection of encrypted traffic for certain keywords, (ii) a technique to allow the middlebox to obtain encrypted rules (i.e., based on the rules from the middlebox and the private key of the endpoints), and (iii) a mechanism to allow flow decryption when a suspicious keyword is observed in the flow. mcTLS extends the traditional TLS protocol to support middleboxes by allowing endpoints and content providers to explicitly introduce middleboxes in secure end-to-end sessions while controlling which parts of the data they can read or write [82]. Asghar et al. [9] propose SplitBox, in which a cloud service provider privately computes network functions on behalf of the client. More specifically, SplitBox provides security guarantees in the honest-but-curious model and works based on cryptographic secret sharing. As proof-of-concept, authors implemented a firewall and measured the bandwidth and latency achieved. Embark [64] enables a cloud provider to support middlebox outsourcing respecting user privacy. Embark encrypts the traffic that is transmitted to the cloud and enables the cloud to process the encrypted traffic without having to decrypt it. Yuan et al. [148] propose a system architecture for outsourced middleboxes to perform DPI over encrypted traffic, without revealing either packet payloads or inspection rules. SPABox [39] is a middlebox-based system that supports keyword-based and data analysis-based DPI functions over encrypted traffic without having to decrypt it. Canard et al. [17] present BlindIDS, which is able to perform deep packet inspection directly on encrypted network packets for intrusion detection. BlindIDS does not assume knowledge over the traffic content or the patterns of detection signatures. Authors evaluate the performance of BlindIDS by presenting the overhead on sender and receiver sides (i.e., connection setup time, data encryption time) and the overhead on the service provider (i.e., detection time, memory usage). Ning et al. [83] propose PrivDPI, a tool that addresses the performance limitations of BlindBox [115]. Guo et al. [48] propose a privacy preserving packet header processing approach for middleboxes that are outsourced to cloud infrastructures. Authors perform a security analysis and identify information leakages, while they evaluate the performance of the prototype (i.e., initialization time, memory cost, latency, throughput and overhead). To overcome the limitation of privacy violation when the network traffic is intercepted for further processing, there

are works that propose hardware-assisted solutions that enable trusted execution. These works enable the secure processing of sensitive information, such as the network traffic, inside encrypted memory regions provided by Trusted Execution Environments (TEEs). One example of TEE is the Intel SGX technology, that is supported by Intel Skylake processors (and successors). TEEs offer secure processing when the execution environment could not be trusted (e.g., a cloud infrastructure). Naylor et al. [81] propose mbTLS, a protocol that enables secure outsourcing middlebox functionality to untrusted infrastructure using the Intel SGX technology [179]. Han et al. [50] present SGX-Box, a secure middlebox system implementation that offers visibility in encrypted network traffic, taking advantage of the Intel's SGX technology [179]. SGX-Box ensures that the sensitive information, such as decrypted payloads and session keys, is securely protected within the protected memory enclave. Coughlin et al. [26] propose the Intel SGX technology to overcome security issues of Network Function Virtualization (NFV) applications in cloud environments. Poddar et al. [98] present SafeBricks that also proposes the utilization of Intel SGX to shield the execution of NFV functions in untrusted environments like the cloud infrastructure. Similarly, Trach et al. [128] present ShieldBox, a middlebox framework for deploying network functions over untrusted commodity servers. ShieldBox takes advantage of the Intel SGX technology and authors deploy two use cases: (i) a multiport IP router and (ii) an intrusion detection system. Goltzsche et al. [46] propose EndBox. EndBox executes middlebox functions on client machines at a network edge and combines a virtual private network (VPN) with middlebox functions that are protected in Intel SGX hardware enclaves. Duan et al. [34] present LightBox, which offers efficient and protected middlebox functionality using the Intel SGX technology.

4.4.1 Techniques

Works in this category either process the encrypted network traffic using software based cryptographic techniques, such as searchable encryption, or process the network traffic inside encrypted memory regions in order to ensure the preservation of privacy. With searchable encryption, tools are able to search directly on encrypted data without decrypting it, and thus, without leaking information in plaitext (e.g., for privacy preserving malware detection [29]). This is achieved by encrypting the rules to be searched against the already

Table 4.11: Techniques, algorithms and evaluation metrics used by network functions in midlleboxes.

Work	**Algorithm/Technique**	**Performance Evaluation Metrics**
Sherry et al. [115]	DPI with Searchable Encryption	Overhead in Load Time & Bandwidth
Naylor et al. [82]	DPI with Searchable Encryption	Handshake, File Transfer, Page Load Times, Data Volume, CPU and Deployment Overheads
Asghar et al. [9]	DPI with Searchable Encryption	Throughput and Delay
Canard et al. [17]	DPI with Searchable Encryption	Connection Setup, Data Encryption and Detection Times, Memory Usage
Lan et al. [64]	DPI with Searchable Encryption	Performance Overhead in Throughput, Page Load Time, Time per-request
Yuan et al. [148]	DPI with Searchable Encryption	TPR and Initialization Time, Inspection Throughput, Token Overhead, Latency
Fan et al. [39]	DPI with Searchable Encryption	End-to-end Delay, Throughput, CPU Utilization, Connection Setup Overhead & Malware Detection Accuracy
Naylor et al. [81]	Secure hardware-assisted DPI (TEE: Intel SGX)	CPU Overhead, Handshake Latency, SGX I/O Throughput Overhead
Han et al. [50]	Secure hardware-assisted DPI (TEE: Intel SGX)	Performance Overhead in Throughput
Coughlin et al. [26]	Secure hardware-assisted DPI (TEE: Intel SGX)	Processing Throughput
Poddar et al. [98]	Secure hardware-assisted NFV (TEE: Intel SGX)	I/O Throughput and Memory Usage Overhead
Trach et al. [128]	Secure hardware-assisted DPI (TEE: Intel SGX)	Throughput, Latency, Scalability
Goltzsche et al. [46]	Secure hardware-assisted DPI (TEE: Intel SGX)	Round Trip Time, Throughput, Latency, CPU usage Overheads
Duan et al. [34]	Secure hardware-assisted DPI (TEE: Intel SGX)	Throughput, CPU usage, Packet Delay
Ning et al. [83]	DPI with Searchable Encryption	Latency, Bandwidth, Token Encryption Time, Round Trip Total Time
Guo et al. [48]	DPI with Searchable Encryption	Initialization Time, Memory Cost, Processing Latency, Throughput, Token Overhead

encrypted traffic (e.g., [17]). Trusted Execution Environments (TEEs) enable the secure code execution and information processing commonly using hardware-assisted features like memory enclaves. Intel SGX [179] is one of the most popular hardware-assisted TEEs and is frequently used to provide confidentiality and integrity guarantees to applications in environments where the OS could eventually become untrusted, like a cloud infrastructure (e.g., trusted antivirus in the cloud [30]).

In [9, 17, 39, 48, 64, 82, 83, 115, 148], authors perform network middlebox applications with core functionality the deep packet inspection using searchable encryption. In [26, 34, 46, 50, 81, 98, 128] authors shield the network traffic processing using hardware enclaves that the Intel SGX technology provides. Table 4.11 displays the most common techniques used for network functions in middleboxes. Also, we present the metrics that each work

used for evaluation.

Datasets The datasets used to evaluate the works of this section are not other than the ones used by traditional works in network traffic processing inside middleboxes. Depending on the application goal (e.g., intrusion detection or firewall), authors utilize the relevant rules (e.g., Snort rules [203] in [17, 39, 83, 148]) and traffic traces (e.g., DARPA [151] in [148]). In this section, we do not include a table, since the datasets used by works in this category do not provide labeled and encrypted network traffic to be used as ground-truth data for encrypted network traffic analysis.

4.4.2 Objectives and Limitations

While the main objective of the majority of the works presented in this survey is to ensure the functionality of their systems by providing knowledge on how to properly analyze encrypted network traffic in order to extract information about its nature, the works in this category focus also on the programmability and deployment of their systems. Thus, authors provide evaluation results not only for the effectiveness of their solution but also for the processing performance and the overhead that is introduced using either a software-centric solution like the searchable encryption or a hardware-assisted technology like TEEs.

Although cryptographic tools like the searchable encryption are very effective and significantly preserve user privacy when it comes to network functions in middleboxes, the individual functions that are performed (e.g., rule encryption and connection setup) add an essential overhead to the end-to-end performance. Similarly, TEEs also increase the processing overhead. For instance, with Intel SGX, substantial overhead can be presented when the computations are I/O bound [19]. Even though both techniques are effective, they introduce the trade-off of user privacy versus performance.

4.4.3 Relation to this Dissertation

Although the works presented in this section (§ 4.4) are not closely related to the work presented in this dissertation, we include them for completeness, since they also advocate the need for encrypted traffic processing.

Chapter 5
Discussion

5.1 Encrypted Traffic Analysis Countermeasures

Even though encrypted traffic analysis techniques can be used by different actors (such as ISPs/CSPs, etc.) to *benignly* extract information about network usage, encrypted traffic analysis techniques can be also used by malicious actors (such as governments that censor websites or prohibit Internet usage) in order to harm the privacy that network encryption offers to Internet users. Some of the most popular solutions propose randomizing network packet sizes, padding bytes to packets for a fixed size and time tuning for inter-packet transmission. More sophisticated solutions are explicitly discussed in the following paragraphs. In this chapter, we discuss about the (i) techniques that can be used against encrypted network traffic analysis and (ii) systems that exist and aim to defend against it.

Anonymity Tools and Tunnelling Protocols Onion Routing serves as an overlay network designed to anonymize communications and applications (e.g., web browsing and instant messaging). TOR is one good example of onion routing. Among others, it uses fixed-size cells that are the unit of communication in TOR [205]. As we have already discussed in Section 4.3 though, anonymity tools are not enough to prevent website and hidden services identification [104], since the existence of anonymity tools can be identified using numerous encrypted traffic analysis techniques (§ 4.3.1). Also, in [145] authors show that the utilization of OpenVPN is also detectable.

Traffic Shaping As already discussed, features and characteristics of network traffic that present patterns after encryption (e.g., packet sizes and timing), can reveal surprising in-

formation about the traffic's nature and contents. Even though encrypted traffic analysis can be legitimate, these techniques raise important concerns about privacy related issues. An approach that typically mitigates such threats is the padding of packets sizes or the transmission of packets at fixed timing intervals; obfuscating the behaviour of a communication mean. However, this method can become inefficient because it results to time overheads. Wright et al. propose a method for hindering statistical traffic analysis algorithms. Their approach proposes the modification of a certain network traffic "class" to look like another. Authors show how to modify packets' characteristics in real-time with low overhead in order to reduce the accuracy measurements of traffic classifiers. The morphed data is then sent to the network stack encrypted and then sent to the destination [141]. AnonRep [149] builds on top of anonymity and privacy guarantees for the case of reputation and voting systems. TARN [147] randomises IP addresses, while TARANET [20] employs packet mixing and splitting to achieve constant-rate transmission, providing anonymity at the network layer. Luo et al. [76] design the HTTPOS fingerprinting defense at the application layer. HTTPOS acts as a proxy that receives HTTP requests and obfuscates them before allowing transmission. Specifically, it modifies network-related features, such as the total packet size, the packet timestamp and the payload size. In addition, it uses HTTP pipelining to obfuscate the number of the transmitted packets. Authors show that HTTPOS was successful in defending against a number of traffic classifiers. Dyer et al. [36] create a defense, namely BuFLO that combines previously proposed countermeasures, such as fixed packet sizes and constant rate traffic. Authors improve other related defenses at the expense of a high bandwidth overhead. Cai et al. [15] make modifications to the BuFLO defense proposing the rate adaptation technique. Yet, this adds a bandwidth overhead. Nithyanand et al. [84] propose Glove, that groups website traffic into clusters. This provides privacy guarantees and reduces the bandwidth overhead by grouping web traffic into similar sets. Panchenko et al. [88] propose "website camouflage", which is actually an obfuscation technique that randomly requests a second website, simultaneously with the actually requested one. Frolov et al. [43] propose uTLS that enables tool maintainers to automatically mimic other popular TLS implementations to prevent censorship. Walkie-Talkie is a website fingerprinting defense approach that modifies the browser to communicate in half-duplex mode, since it produces burst sequences that leak less information to the adversary. This makes sensitive and non-sensitive pages look the

same [136].

Traffic Analysis Resistant IoT Devices Hafeez et al. [49] demonstrate that an adversary, with access to the network traffic of a "smart" home network, can lead to the identification of the device types and some user interactions with IoT devices. In order to defend against traffic analysis attacks, authors propose a "traffic morphing" technique that shapes network traffic in ways that make it more difficult to achieve an attack that identifies devices and activities. In order to mask the background traffic, authors send traffic on an upstream link at a constant rate, irrespective of real background traffic rate of an IoT device. Meanwhile, when an IoT device is inactive, authors send dummy traffic representing device activity to upstream link, so that an adversary can not identify real activity of the IoT device. While this approach is not very sophisticated, it points to the direction of defending against traffic analysis on IoT devices. In [133], authors propose IoTReguard, a system that aims to explore network traffic features that reveal the most relevant ones and hide them to protect users' privacy.

Traffic Analysis Resistant Messaging Applications There have been efforts to create messaging protocols that provide anonymity and privacy guarantees in the face of traffic analysis. Dissent [139] and Riposte [25] are systems that provide strong user privacy guarantees. They protect packet metadata, but they suffer from scalability issues. Herd [68] is another system that tackles the case of anonymity for VoIP calls, by addressing, like the former proposals, some of the limitations of the more general-purpose Tor anonymity network [32]. Vuvuzela [129] and Atom [63] are more scalable systems (thousands of messages for millions of users) that employ differential privacy to inject noise into observable metadata.

5.2 Quality and Quantity of Data

Approaches like our work presented in Sections 2.1 and 2.2 depend exclusively on the ground truth. Therefore, data must not be characterized by low quality or insufficient quantity. Unfortunately, only a small amount of recent public datasets, properly labelled (presented in § 4.1.1, § 4.2.1, § 4.3.1 and § 4.4.1) exist. In our work presented in Chapter 2, to evaluate our methodology we use a combination of public datasets (i.e., [150, 171, 208])

and ground-truth data that we collect in our own environments. In this way, we have the flexibility to deploy different OS/application versions and examine the diversity in the network traffic. This enables us to ensure that our methodology is resilient to different network stack implementations. Overall, poor ground truth information can undermine the effectiveness of passive network traffic analysis, highlighting the need for accurate and reliable data to support network monitoring, management and security.

For the characterization of malicious servers, we collect IP addresses posted on public lists. For instance, the Feodo Tracker Botnet C2 IP Blocklist mentions in its description that it "only contains active botnet C&C servers or such that have been active in the past hours". Probing IP addresses retrieved from lists that are getting updated frequently keeps the possibility of false positives low.

5.3 Validation and False Positives

When network traffic is encrypted, achieving result validation in a real-world deployment, at the ISP/CSP level, is challenging. Encryption is used to protect sensitive information from unauthorized access or interception, which can make it difficult to monitor or analyze network traffic. In use cases that are related to network analytics (discussed in Section 2.1), result validation requires a combination of encryption and decryption techniques, network tunneling solutions, behavioural analytics and machine learning techniques. When the use case is related to network security (discussed in Section 2.2), result validation requires a combination of endpoint detection, network-based analysis and security event management strategies.

5.4 Passive Monitoring versus Active Scanning

In Chapter 2 we follow a passive network monitoring and analysis approach . Passive network traffic monitoring has several advantages, including the ability to detect anomalies and security threats without disrupting the normal flow of network traffic. Passive network traffic analysis can be performed on a wide range of network types and protocols, making it a versatile tool for network administrators and security professionals. However, passive network traffic analysis also has some limitations, since it requires specialized tools and

expertise to capture and analyze network traffic, which can be time-consuming. Also, it might not provide a complete picture of network activity, as we capture packets that are transmitted over a specific and dedicated network setup. In addition, passive network traffic analysis can be subject to privacy violations, as it may capture sensitive data that is transmitted over the network, and thus sharing data can be challenging.

Active network scanning is the method of monitoring network traffic by actively sending packets into the network and analyzing the responses received. As discussed in Chapter 3, active network traffic enables us a granular and targeted analysis of network activity, focused on botnet configurations. Of course, active network traffic scanning must be carefully controlled and monitored to ensure that it does not inadvertently (i) create security vulnerabilities, (ii) provide attackers with useful information about the network [99], or (iii) cause network congestion affecting the overall network performance.

5.5 TLS 1.3 and Beyond

Expecting the vast adoption of TLS 1.3, we do not perform TLS certificate fingerprinting, like other relevant solutions [189]. The TLS 1.3 handshake is quite different from earlier versions of TLS, with a large portion of it getting encrypted [61]. The TLS 1.3 protocol ensures the privacy and security of the certificate exchange through the use of digital signatures and cryptographic mechanisms. Thus, the introduction of TLS 1.3 encourages our proposed methodology, in which we propose the inspection of sequences of packet metadata, like the packet payload size.

Our methodology for the identification of events on encrypted network traffic (§ 2.1, § 2.2) is not affected by the TLS 1.3 ClientHello Padding Extension [180] that enables padding in the TLS ClientHello messages to a desired size, since we do not process such packets. Likewise, the characterization of malicious servers technique is not affected by the TLS 1.3, since the certificate data that is present in the “TLS Server Hello” packet is ignored and not used for the fingerprint construction.

Chapter 6
Conclusion

6.1 Synopsis of Contributions

In this work, we discussed the fine-grained identification of events over encrypted network traffic focusing on scalability and maintainability. For the application event detection use case, we demonstrated that (i) a simple regex-inspired language is expressive enough to achieve a minimum hit rate of 84%, (ii) the proprietary DPI engine processed an average of 109 Gbps with no packet loss, and (iii) the rule extraction is amenable to data mining techniques. Similarly, for the use case of malicious activity detection, we demonstrate a methodology to mine signatures for intrusion detection in encrypted networks. We show that a simple signature language can be expressive and effective enough also for intrusion detection in encrypted networks, while it can achieve a processing throughput of up to 85Gbps (the implementation details of this proof-of-concept DPI engine is presented in § 2.2.3). The work focuses on a real-world implementation because we believe that just like substring pattern matching is a requirement in a state-of-the-art network monitoring system, so is packet metadata sequence matching, even if techniques such as encryption and traffic analysis resistance exist to evade them.

We also explore how botnets evolve in time and the fingerprint overlapping with legitimate servers. By actively contacting IP addresses of known command and control servers, we create a database of TLS server fingerprints grouped by botnet. We show that an out-

dated list of fingerprints can cause false positives. We narrow down the size of the fingerprint to avoid analysis circumvention techniques applied by malicious servers. Investigating the existence of those fingerprints into blocklists of maliciously acting IP addresses, we are able to (re-)identify the same TLS server configurations that could indicate specific botnet activity, based on our knowledge base.

The key takeaways of this dissertation are the following:

- Encrypted traffic inspection is feasible using patterns of packet metadata sequences
- Packet metadata sequences can be described using a simple yet expressive language that also enables automated mining
- With the integration of the pattern language into two different DPI engines (one proprietary and one proof-of-concept), we demonstrate that it can achieve high processing throughput for real-time processing
- Although botnets operate using encrypted traffic nowadays, we can still determine a server's activity from a TLS fingerprint
- As time passes, we observe higher fingerprint overlapping between malicious and legitimate servers that causes accuracy degradation; this implies the need to update the fingerprint database frequently

6.2 Directions for Future Work and Research

There are several aspects that are worth further work and research. Specifically, we plan to generate signatures and evaluate our methodology using more traffic captures from other popular malware families. Also, we aim to examine (i) how we can approach traffic inspection for encrypted protocols that multiplex network flows (e.g., VPN) and (ii) how sensitive our patterns are to more network stack implementations of endpoints. Of course, we plan

to enhance and fine-tune our proof-of-concept DPI engine (described in Section 2.2.3) to achieve the optimal end-to-end processing performance.

As future work for the characterization of malicious servers on the Internet, we aim to enrich our TLS fingerprints database with more and different botnets, explore approaches that could help us recognize the randomization of cipher suite vectors and measure how common this randomization is and in which botnet families. We will perform a more in depth analysis of those server TLS responses specifically to uncommon "TLS Client Hello" configurations. Finally, we plan to further verify our findings by monitoring and analyzing each malware by installing it in a virtual environment.

Bibliography

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Multi-classification approaches for classifying mobile app traffic. *Journal of Network and Computer Applications*, 103:131–145, 2018.

[2] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer Networks*, 165:106944, 2019.

[3] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[4] Fabio Aiolli, Mauro Conti, Ankit Gangwal, and Mirko Polato. Mind your wallet's privacy: identifying bitcoin wallet apps and user's actions through network traffic analysis. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1484–1491. ACM, 2019.

[5] Hasan Faik Alan and Jasleen Kaur. Can Android applications be identified using only TCP/IP headers of their launch time traffic? In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 61–66. ACM, 2016.

[6] Payam Vahdani Amoli, Timo Hamalainen, Gil David, Mikhail Zolotukhin, and Mahsa Mirzamohammad. Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets. *JDCTA (International Journal of Digital Content Technology and its Applications*, 10(2):1–13, 2016.

[7] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1723–1732. ACM, 2017.

[8] Blake Anderson and David McGrew. Tls beyond the browser: Combining end host and network data to understand application behavior. In *Proceedings of the Internet Measurement Conference*, pages 379–392, 2019.

[9] Hassan Jameel Asghar, Luca Melis, Cyril Soldani, Emiliano De Cristofaro, Mohamed Ali Kaafar, and Laurent Mathy. Splitbox: Toward efficient private network function virtualization. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 7–13, 2016.

[10] Giuseppe Ateniese, Briland Hitaj, Luigi Vincenzo Mancini, Nino Vincenzo Verde, and Antonio Villani. No place to hide that bytes won't reveal: Sniffing location-based encrypted traffic to track a user's position. In *International Conference on Network and System Security*, pages 46–59. Springer, 2015.

[11] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. In *International Conference on Passive and Active Network Measurement*, pages 165–175. Springer, 2007.

[12] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[13] Martin Björklund, Marcus Julin, Philip Antonsson, Andreas Stenwreth, Malte Åkvist, Tobias Hjalmarsson, and Romaric Duvignau. I see what you're watching on your streaming service: Fast identification of dash encrypted network traces. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pages 1116–1122. IEEE, 2023.

[14] Jonas Bushart and Christian Rossow. Padding ain't enough: Assessing the privacy guarantees of encrypted {DNS}. In *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.

[15] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130. ACM, 2014.

[16] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

[17] Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine, and Mohamed Sabt. Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 561–574, 2017.

[18] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, RFC 3234, February, 2002.

[19] Sang Kil Cha, Iulian Moraru, Jiyong Jang, John Truelove, David Brumley, and David G Andersen. Splitscreen: Enabling efficient, distributed malware detection. *Journal of Communications and Networks*, 13(2):187–200, 2011.

[20] Chen Chen, Daniele E Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. Taranet: Traffic-analysis resistant anonymity at the network layer. *arXiv preprint arXiv:1802.08415*, 2018.

[21] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. Os fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 173–180. ACM, 2014.

[22] Mauro Conti, Luigi V Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 297–304. ACM, 2015.

[23] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, 2016.

[24] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, pages 1–16, 2017.

[25] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 321–338. IEEE, 2015.

[26] Michael Coughlin, Eric Keller, and Eric Wustrow. Trusted click: Overcoming security issues of nfv in the cloud. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 31–36, 2017.

[27] Scott E Coull and Kevin P Dyer. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review*, 44(5):5–11, 2014.

[28] Michelangelo Cruz, Roel Ocampo, Isabel Montes, and Rowel Atienza. Fingerprinting bittorrent traffic in encrypted tunnels using recurrent deep learning. In *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pages 434–438. IEEE, 2017.

[29] Helei Cui, Yajin Zhou, Cong Wang, Qi Li, and Kui Ren. Towards privacy-preserving malware detection systems for android. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 545–552. IEEE, 2018.

[30] Dimitris Deyannis, Eva Papadogiannaki, Giorgos Kalivianakis, Giorgos Vasiliadis, and Sotiris Ioannidis. Trustav: Practical and privacy preserving malware analysis in the cloud. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 39–48, 2020.

[31] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. Measuring video qoe from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference*, pages 513–526, 2016.

[32] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[33] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[34] Huayi Duan, Cong Wang, Xingliang Yuan, Yajin Zhou, Qian Wang, and Kui Ren. Lightbox: Full-stack protected stateful middlebox at lightning speed. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2351–2367, 2019.

[35] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. The security impact of https interception. In *NDSS*, 2017.

[36] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[37] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[38] Zubair Md Fadlullah, Tarik Taleb, Nirwan Ansari, Kazuo Hashimoto, Yutaka Miyake, Yoshiaki Nemoto, and Nei Kato. Combating against attacks on encrypted protocols. In *2007 IEEE International Conference on Communications*, pages 1211–1216. IEEE, 2007.

[39] Jingyuan Fan, Chaowen Guan, Kui Ren, Yong Cui, and Chunming Qiao. Spabox: Safeguarding privacy during deep packet inspection at a middlebox. *IEEE/ACM Transactions on Networking (TON)*, 25(6):3753–3766, 2017.

[40] Yebo Feng, Jianzhen Luo, Chengyan Ma, Teng Li, and Liang Hui. I can still observe you: Flow-level behavior fingerprinting for online social network. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 6427–6432. IEEE, 2022.

[41] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 40–52. Springer, 2014.

[42] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S Tseng. Vmsp: Efficient vertical mining of maximal sequential patterns. In *Canadian conference on artificial intelligence*, pages 83–94. Springer, 2014.

[43] Sergey Frolov and Eric Wustrow. The use of tls in censorship circumvention. In *NDSS*, 2019.

[44] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. *arXiv preprint arXiv:2301.13686*, 2023.

[45] Yanjie Fu, Hui Xiong, Xinjiang Lu, Jin Yang, and Can Chen. Service usage classification with encrypted internet traffic in mobile messaging apps. *IEEE Transactions on Mobile Computing*, 15(11):2851–2864, 2016.

[46] David Goltzsche, Signe Rüsch, Manuel Nieke, Sébastien Vaucher, Nico Weichbrodt, Valerio Schiavoni, Pierre-Louis Aublin, Paolo Cosa, Christof Fetzer, Pascal Felber, et al. Endbox: Scalable middlebox functions using client-side trusted execution. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 386–397. IEEE, 2018.

[47] Antonio Gomariz, Manuel Campos, Roque Marin, and Bart Goethals. Clasp: An efficient algorithm for mining frequent closed sequences. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 50–61, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[48] Yu Guo, Mingyue Wang, Cong Wang, Xingliang Yuan, and Xiaohua Jia. Privacy-preserving packet header checking over in-the-cloud middleboxes. *IEEE Internet of Things Journal*, 2020.

[49] Ibbad Hafeez, Markku Antikainen, and Sasu Tarkoma. Protecting iot-environments against traffic analysis attacks with traffic morphing. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 196–201. IEEE, 2019.

[50] Juhyeng Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. Sgx-box: Enabling visibility on encrypted traffic using a secure middlebox module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 99–105. ACM, 2017.

[51] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[52] Nguyen Phong Hoang, Michalis Polychronakis, and Phillipa Gill. Measuring the accessibility of domain name encryption and its impact on internet filtering. In *International Conference on Passive and Active Network Measurement*, pages 518–536. Springer, 2022.

[53] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3366–3383, 2021.

[54] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *2013 IEEE Symposium on Security and Privacy*, pages 65–79. IEEE, 2013.

[55] Minghao Jiang, Gaopeng Gou, Junzheng Shi, and Gang Xiong. I know what you are doing with remote desktop. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pages 1–7. IEEE, 2019.

[56] Steven Jorgensen, John Holodnak, Jensen Dempsey, Karla de Souza, Ananditha Raghunath, Vernon Rivet, Noah DeMoes, Andrés Alejos, and Allan Wollaber. Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification. *IEEE Transactions on Artificial Intelligence*, 2023.

[57] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

[58] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[59] Muhammad Jawad Khokhar, Thibaut Ehlinger, and Chadi Barakat. From network traffic measurements to qoe for internet video. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2019.

[60] Platon Kotzias, Leyla Bilge, Pierre-Antoine Vervier, and Juan Caballero. Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises. In *NDSS*, 2019.

[61] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*, pages 415–428, 2018.

[62] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 287–302, 2015.

[63] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 406–422. ACM, 2017.

[64] Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 255–273, 2016.

[65] Arash Habibi Lashkari, Andi Fitriah A Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A Ghorbani. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 233–23309. IEEE, 2017.

[66] Martin Lavstovivcka, Stanislav Spavcek, Petr Velan, and Pavel Celeda. Dataset: Using tls fingerprints for os identification in encrypted traffic. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6. IEEE, 2020.

[67] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. Antmonitor: A system for monitoring from mobile devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*, pages 15–20. ACM, 2015.

[68] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. Herd: A scalable, traffic analysis resistant anonymity network for voip systems. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 639–652. ACM, 2015.

[69] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. Good bot, bad bot: Characterizing automated browsing activity. In *2021 IEEE symposium on security and privacy (sp)*, page 17, 2021.

[70] Junjie Liang, Wenbo Guo, Tongbo Luo, Honavar Vasant, Gang Wang, and Xinyu Xing. Fare: Enabling fine-grained attack categorization under low-quality labeled data. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2021.

[71] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[72] Junming Liu, Yanjie Fu, Jingci Ming, Yong Ren, Leilei Sun, and Hui Xiong. Effective and real-time in-app activity analysis in encrypted internet traffic streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 335–344. ACM, 2017.

[73] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[74] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, pages 1–14, 2017.

[75] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security*, pages 199–214. Springer, 2010.

[76] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, volume 11. Citeseer, 2011.

[77] M Hammad Mazhar and Zubair Shafiq. Real-time video quality of experience monitoring for https and quic. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1331–1339. IEEE, 2018.

[78] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[79] Antonio Montieri, Domenico Ciuonzo, Giampaolo Bovenzi, Valerio Persico, and Antonio Pescapé. A dive into the dark web: Hierarchical traffic classification of anonymity tools. *IEEE Transactions on Network Science and Engineering*, 2019.

[80] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[81] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 88–100. ACM, 2017.

[82] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. *ACM SIGCOMM Computer Communication Review*, 45(4):199–212, 2015.

[83] Jianting Ning, Geong Sen Poh, Jia-Ch'ng Loh, Jason Chia, and Ee-Chien Chang. Privdpi: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1657–1670, 2019.

[84] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 131–134. ACM, 2014.

[85] Quamar Niyaz, Weiqing Sun, and Ahmad Y Javaid. A deep learning based ddos detection system in software-defined networking (sdn). *arXiv preprint arXiv:1611.07400*, 2016.

[86] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. Youtube qoe estimation based on the analysis of encrypted network traffic using machine learning. In *2016 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2016.

[87] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[88] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[89] Eva Papadogiannaki, Dimitris Deyannis, and Sotiris Ioannidis. Head (er) hunter: Fast intrusion detection using packet metadata signatures. In *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2020.

[90] Eva Papadogiannaki, Constantinos Halevidis, Periklis Akritidis, and Lazaros Koromilas. Otter: A scalable high-resolution encrypted traffic identification engine. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 315–334. Springer, 2018.

[91] Eva Papadogiannaki and Sotiris Ioannidis. Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware. *Sensors*, 21(4):1140, 2021.

[92] Eva Papadogiannaki and Sotiris Ioannidis. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[93] Eva Papadogiannaki and Sotiris Ioannidis. Pump up the jarm: Studying the evolution of botnets using active tls fingerprinting. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 764–770. IEEE, 2023.

[94] Eva Papadogiannaki, Lazaros Koromilas, Giorgos Vasiliadis, and Sotiris Ioannidis. Efficient software packet processing on heterogeneous and asymmetric hardware architectures. *IEEE/ACM Transactions on Networking*, 25(3):1593–1606, 2017.

[95] Eva Papadogiannaki, Giorgos Tsirantonakis, and Sotiris Ioannidis. Network intrusion detection in encrypted traffic. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2022.

[96] Muhammad Talha Paracha, Daniel J Dubois, Narseo Vallina-Rodriguez, and David Choffnes. Iotls: understanding tls usage in consumer iot devices. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 165–178, 2021.

[97] Mike Perry. Experimental defense for website traffic fingerprinting. *Tor project blog*, 2011.

[98] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Safebricks: Shielding network functions in the cloud. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 201–216, 2018.

[99] Elias Raftopoulos, Eduard Glatz, Xenofontas Dimitropoulos, and Alberto Dainotti. How dangerous is internet scanning? a measurement study of the aftermath of

an internet-wide scan. In *Traffic Monitoring and Analysis: 7th International Workshop, TMA 2015, Barcelona, Spain, April 21-24, 2015. Proceedings 7*, pages 158–172. Springer, 2015.

[100] Marianna Rapoport, Philippe Suter, Erik Wittern, Ondrej Lhóták, and Julian Dolby. Who you gonna call? analyzing web requests in android applications. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 80–90. IEEE, 2017.

[101] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. Studying tls usage in android apps. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 350–362, 2017.

[102] Abbas Razaghpanah, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Phillipa Gill, Mark Allman, and Vern Paxson. Haystack: In situ mobile traffic analysis in user space. *ArXiv e-prints*, 2015.

[103] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.

[104] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[105] Luigi Rizzo, Marta Carbone, and Gaetano Catalli. Transparent acceleration of software packet forwarding using netmap. In *INFOCOM, 2012 Proceedings IEEE*, pages 2471–2479. IEEE, 2012.

[106] Nicolás Rosner, Ismet Burak Kadron, Lucas Bang, and Tevfik Bultan. Profit: Detecting and quantifying side channels in networked applications. In *NDSS*, 2019.

[107] Nicholas Ruffing, Ye Zhu, Rudy Libertini, Yong Guan, and Riccardo Bettati. Smartphone reconnaissance: operating system identification. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pages 1086–1091. IEEE, 2016.

[108] Brendan Saltaformaggio, Hongjun Choi, Kristen Johnson, Yonghwi Kwon, Qi Zhang, Xiangyu Zhang, Dongyan Xu, and John Qian. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *WOOT*, 2016.

[109] Dominik Schatzmann, Wolfgang Mühlbauer, Thrasyvoulos Spyropoulos, and Xenofontas Dimitropoulos. Digging into https: flow-based classification of webmail traffic. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 322–327, 2010.

[110] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1357–1374, 2017.

[111] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[112] Asaf Shabtai, Lena Tenenboim-Chekina, Dudu Mimran, Lior Rokach, Bracha Shapira, and Yuval Elovici. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43:1–18, 2014.

[113] Khalid Shahbar and A Nur Zincir-Heywood. Packet momentum for identification of anonymity networks. *Journal of Cyber Security and Mobility*, 6(1):27–56, 2017.

[114] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, Xiaojiang Du, and Yiting Liu. Encrypted traffic classification of decentralized applications on ethereum using feature fusion. In *Proceedings of the International Symposium on Quality of Service*, page 18. ACM, 2019.

[115] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *ACM SIGCOMM Computer communication review*, 45(4):213–226, 2015.

[116] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, 2018.

[117] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2018.

[118] Monika Skowron, Artur Janicki, and Wojciech Mazurczyk. Traffic fingerprinting attacks on internet of things using machine learning. *IEEE Access*, 8:20386–20400, 2020.

[119] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55. IEEE, 2000.

[120] Yihang Song and Urs Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 15–26. ACM, 2015.

[121] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle. Dissectls: A scalable active scanner for tls server configurations, capabilities, and tls fingerprinting. In *Passive and Active Measurement: 24th International Conference, PAM 2023, Virtual Event, March 21–23, 2023, Proceedings*, pages 110–126. Springer, 2023.

[122] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. Active tls stack fingerprinting: Characterizing tls server deployments at scale. *arXiv preprint arXiv:2206.13230*, 2022.

[123] Tarik Taleb, Zubair Md Fadlullah, Kazuo Hashimoto, Yoshiaki Nemoto, and Nei Kato. Tracing back attacks against encrypted protocols. In *Proceedings of the 2007 international conference on Wireless communications and mobile computing*, pages 121–126. ACM, 2007.

[124] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. Copperdroid: automatic reconstruction of android malware behaviors. In *Ndss*, 2015.

[125] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep learning approach for network intrusion detection in software defined networking. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 258–263. IEEE, 2016.

[126] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 439–454. IEEE, 2016.

[127] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2018.

[128] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. Shieldbox: Secure middleboxes using shielded execution. In *Proceedings of the Symposium on SDN Research*, pages 1–14, 2018.

[129] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[130] Thijs Van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium (NDSS)*, volume 27, 2020.

[131] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, 2008.

[132] Giorgos Vasiliadis, Michalis Polychronakis, and Sotiris Ioannidis. MIDeA: A Multi-Parallel Intrusion Detection Architecture. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.

[133] Andressa Vergütz, Bruna V dos Santos, Burak Kantarci, and Michele Nogueira. Data instrumentation from iot network traffic as support for security management. *IEEE Transactions on Network and Service Management*, 2023.

[134] Qinglong Wang, Amir Yahyavi, Bettina Kemme, and Wenbo He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 433–441. IEEE, 2015.

[135] Shanshan Wang, Zhenxiang Chen, Lei Zhang, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*, pages 1–6. IEEE, 2016.

[136] Tao Wang and Ian Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1375–1390, 2017.

[137] Yaru Wang, Ning Zheng, Ming Xu, Tong Qiao, Qiang Zhang, Feipeng Yan, and Jian Xu. Hierarchical identifier: Application to user privacy eavesdropping on mobile payment app. *Sensors*, 19(14):3052, 2019.

[138] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. ProfileDroid: multi-layer profiling of android applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 137–148. ACM, 2012.

[139] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–182, 2012.

[140] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 35–49. IEEE, 2008.

[141] Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, volume 9. Citeseer, 2009.

[142] Hua Wu, Xin Li, Gang Wang, Guang Cheng, and Xiaoyan Hu. Resolution identification of encrypted video streaming based on http/2 features. *ACM Transactions on Multimedia Computing, Communications and Applications*, 19(2):1–23, 2023.

[143] Hongbo Xu, Shuhao Li, Zhenyu Cheng, Rui Qin, Jiang Xie, and Peishuai Sun. Trafficgcn: Mobile application encrypted traffic classification based on gcn. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 891–896. IEEE, 2022.

[144] Shichang Xu, Subhabrata Sen, and Z Morley Mao. Csi: inferring mobile abr video adaptation behavior under https and quic. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[145] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. {OpenVPN} is open to {VPN} fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 483–500, 2022.

[146] Haipeng Yao, Pengcheng Gao, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Zhu Han. Capsule network assisted iot traffic classification mechanism for smart cities. *IEEE Internet of Things Journal*, 6(5):7515–7525, 2019.

[147] Lu Yu, Qing Wang, Geddings Barrineau, Jon Oakley, Richard R Brooks, and Kuang-Ching Wang. TARN: A SDN-based traffic analysis resistant network architecture. *arXiv preprint arXiv:1709.00782*, 2017.

[148] Xingliang Yuan, Xinyu Wang, Jianxiong Lin, and Cong Wang. Privacy-preserving deep packet inspection in outsourced middleboxes. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[149] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. AnonRep: Towards tracking-resistant anonymous reputation. In *NSDI*, pages 583–596, 2016.

[150] Dataset: The UNSW-NB15 Dataset. `https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/`. Accessed: 2020-05-05.

[151] Dataset: The DARPA'99 Dataset. `https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset`, 1999. Accessed: 2021-02-01.

[152] Dataset: The KDD Cup 1999 Dataset. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999. Accessed: 2021-02-01.

[153] Dataset: The LBNL/ICSI Dataset. `https://www.icir.org/enterprise-tracing/download.html`, 2005. Accessed: 2021-02-01.

[154] Dataset: A detailed traceset of network activity at OSDI 2006. `https://ieee-dataport.org/open-access/crawdad-microsoftosdi2006`, 2007. Accessed: 2021-02-01.

[155] Dataset: Wireless network measurement in the SIGCOMM 2008 conference. `https://ieee-dataport.org/open-access/crawdad-umdsigcomm2008`, 2008. Accessed: 2023-02-01.

[156] Dataset: The NSL-KDD Dataset. `https://www.unb.ca/cic/datasets/nsl.html`, 2009. Accessed: 2021-02-01.

[157] Dataset: Near Real Time SSL Notary Service. https://www.icsi.berkeley.edu/icsi/node/5065, 2012. Accessed: 2023-05-01.

[158] Dataset: Space/Time Analysis for Cybersecurity (STAC) . https://www.darpa.mil/program/space-time-analysis-for-cybersecurity, 2015. Accessed: 2021-02-01.

[159] Dataset: FingerprinTLS. https://github.com/synackpse/tls-fingerprinting, 2016. Accessed: 2023-05-01.

[160] Dataset: The UNB ISCX VPN-nonVPN Dataset. https://www.unb.ca/cic/datasets/vpn.html, 2016. Accessed: 2021-02-01.

[161] Dataset: The USTC-TFC2016 Dataset. https://github.com/yungshenglu/USTC-TFC2016, 2016. Accessed: 2021-02-01.

[162] Dataset: Intrusion Detection Evaluation Datasets (CIC-IDS2017). https://www.unb.ca/cic/datasets/ids-2017.html, 2017. Accessed: 2023-5-13.

[163] Dataset: The Anon17 Dataset. https://projects.cs.dal.ca/projectx/Download.html, 2017. Accessed: 2023-02-01.

[164] Dataset: The Recon Dataset. https://recon.meddle.mobi/appversions/, 2018. Accessed: 2021-02-01.

[165] Dataset: DDoS Evaluation (CIC-DDoS2019). https://www.unb.ca/cic/datasets/ddos-2019.html, 2019. Accessed: 2023-5-13.

[166] Dataset: Encrypted Network Traffic Classification using Deep Learning. https://soeai.github.io/MAppGraph/, 2019. Accessed: 2023-04-01.

[167] Dataset: Mercury network metadata capture and analysis. https://github.com/cisco/mercury, 2019. Accessed: 2023-05-01.

[168] Dataset: TLS Fingerprints for OS Identification in Encrypted Traffic. https://zenodo.org/record/3461771, 2019. Accessed: 2021-02-01.

[169] Dataset: MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/, 2020. Accessed: 2023-05-01.

[170] Dataset: VPN/NONVPN Network Application Traffic Dataset (VNAT). https://www.ll.mit.edu/r-d/datasets/vpnnonvpn-network-application-traffic-dataset-vnat, 2020. Accessed: 2023-04-01.

[171] Dataset: IoT-23 dataset. A labeled dataset with malicious and benign IoT network traffic. https://www.stratosphereips.org/datasets-iot23, 2021. Accessed: 2021-7-13.

[172] Dataset: The ISCX Dataset. https://www.unb.ca/cic/datasets/ids.html, 2021. Accessed: 2021-02-01.

[173] Webpage: Android tcpdump. https://www.androidtcpdump.com. Accessed: 2018-03-09.

[174] Webpage: Application Layer Packet Classifier for Linux. Available: http://l7-filter.sourceforge.net/. Accessed: 2020-05-05.

[175] Webpage: Busybox (android application). https://play.google.com/store/apps/details?id=stericson.busybox&hl=en. Accessed: 2018-03-09.

[176] Webpage: netstat(8) - linux man page. https://linux.die.net/man/8/netstat. Accessed: 2020-05-05.

[177] Webpage: OpenCL. Available: http://www.khronos.org/opencl/. Accessed: 2020-05-05.

[178] Webpage: OpenCL Vector Data Types. https://www.khronos.org/registry/OpenCL/sdk/1.0/docs/man/xhtml/vectorDataTypes.html. Accessed: 2019-07-11.

[179] Webpage: Intel Software Guard Extensions. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html, 2014. Accessed: 2020-07-05.

[180] Webpage: A Transport Layer Security (TLS) ClientHello Padding Extension. https://datatracker.ietf.org/doc/html/rfc7685, 2015. Accessed: 2020-7-17.

[181] Webpage: FingerPrinTLS. https://github.com/LeeBrotherston/tls-fingerprinting/tree/master/fingerprintls, 2016. Accessed: 2022-1-23.

[182] Webpage: Joy: A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring. http://www.dvwa.co.uk, 2017. Accessed: 2020-10-11.

[183] Webpage: JA3: A method for profiling SSL/TLS Clients. https://github.com/salesforce/ja3, 2018. Accessed: 2022-1-23.

[184] Webpage: The Transport Layer Security (TLS) Protocol Version 1.3. https://tools.ietf.org/html/rfc8446, 2018. Accessed: 2020-10-29.

[185] Webpage: A nonprofit Certificate Authority providing TLS certificates to 225 million websites.: 2019 Annual Report. https://abetterinternet.org/documents/2019-ISRG-Annual-Report-Desktop.pdf, 2019. Accessed: 2020-10-29.

[186] Webpage: Encrypted Traffic Analysis. https://www.enisa.europa.eu/publications/encrypted-traffic-analysis, 2019. Accessed: 2023-05-05.

[187] Webpage: TLS 1.3: One Year Later. https://www.ietf.org/blog/tls13-adoption/, 2019. Accessed: 2020-10-29.

[188] Webpage: Automatic SQL injection and database takeover tool. http://sqlmap.org, 2020. Accessed: 2020-25-11.

[189] Webpage: Cisco Encrypted Traffic Analytics. https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html, 2020. Accessed: 2020-05-05.

[190] Webpage: Damn Vulnerable Web Application (DVWA). https://github.com/cisco/joy, 2020. Accessed: 2020-10-11.

[191] Webpage: Dirbuster. https://tools.kali.org/web-applications/dirbuster, 2020. Accessed: 2020-25-11.

[192] Webpage: Easily Identify Malicious Servers on the Internet with JARM. https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a/, 2020. Accessed: 2022-7-10.

[193] Webpage: Hydra Package Description. https://tools.kali.org/password-attacks/hydra, 2020. Accessed: 2020-25-11.

[194] Webpage: JARM: An active Transport Layer Security (TLS) server fingerprinting tool. https://github.com/salesforce/jarm, 2020. Accessed: 2022-1-23.

[195] Webpage: Let's Encrypt. https://letsencrypt.org, 2020. Accessed: 2020-10-29.

[196] Webpage: Metasploit: The world's most used penetration testing framework. https://www.metasploit.com, 2020. Accessed: 2020-25-11.

[197] Webpage: Nikto. https://tools.kali.org/information-gathering/nikto, 2020. Accessed: 2020-25-11.

[198] Webpage: Nmap Security Scanner. https://nmap.org, 2020. Accessed: 2020-25-11.

[199] Webpage: OpenCL runtimes: Obtain runtimes to execute or develop OpenCL applications on Intel Processors. https://www.intel.com/content/www/us/en/developer/articles/tool/opencl-drivers.html#proc-graph-section, 2020. Accessed: 2023-03-03.

[200] Webpage: Openvpn. https://openvpn.net, 2020. Accessed: 2020-05-05.

[201] Webpage: Snort: README.ssl. Available: https://www.snort.org/faq/readme-ssl, 2020. Accessed on Oct. 28, 2020.

[202] Webpage: Suricata Open Source IDS / IPS / NSM engine . Available: https://www.suricata-ids.org/, 2020. Accessed: 2020-05-05.

[203] Webpage: The Snort IDS/IPS. Available: https://www.snort.org/, 2020. Accessed: 2020-05-05.

[204] Webpage: The Zeek Network Security Monitor. Available: https://www.zeek.org/, 2020. Accessed: 2020-05-05.

[205] Webpage: Tor project. https://www.torproject.org, 2020. Accessed: 2020-05-05.

[206] Webpage: blocklist.de. https://lists.blocklist.de/lists/, 2021. Accessed: 2022-1-03.

[207] Webpage: Encrypted Network Traffic Analysis, Transformation and Normalization Techniques. https://www.cybersane-project.eu/files/2023/01/CyberSANE_Deliverable_3.2_Encrypted_Network_Traffic_Analysis_Transformation_and_Normalization_Techniques_v2.pdf, 2021. Accessed: 2023-05-05.

[208] Webpage: Malware Capture Facility Project. Normal Datasets. https://www.stratosphereips.org/datasets-normal, 2021. Accessed: 2021-2-25.

[209] Webpage: MISP - Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing. https://www.misp-project.org, 2021. Accessed: 2022-1-03.

[210] Webpage: Snort Community rules 3100. https://www.snort.org/downloads/community/snort3-community-rules.tar.gz, 2021. Accessed: 2021-2-25.

[211] Webpage: Spamhaus Botnet Threat Update: Q4-2021. https://www.spamhaus.org/news/article/817/spamhaus-botnet-threat-update-q4-2021, 2021. Accessed: 2022-07-15.

[212] Webpage: The 2021 TLS Telemetry Report. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report, 2021. Accessed: 2022-07-15.

[213] Webpage: The CINS Score CI-Badguys list . https://cinsscore.com/list/ci-badguys.txt, 2021. Accessed: 2022-1-03.

[214] Webpage: The Feodo Tracker Botnet C2 IP Blocklist. https://feodotracker.abuse.ch/downloads/ipblocklist.csv, 2021. Accessed: 2022-1-03.

[215] Webpage: The Majestic Million. https://downloads.majestic.com/majestic_million.csv, 2021. Accessed: 2022-1-03.

[216] Webpage: The MalSilo IPv4 feed. https://malsilo.gitlab.io/feeds/dumps/ip_list.txt, 2021. Accessed: 2022-1-03.

[217] Webpage: ThreatLabz State of Encrypted Attacks 2022 Report. https://www.zscaler.com/blogs/security-research/2022-encrypted-attacks-report, 2022. Accessed: 2023-06-04.

[218] Webpage: TLS Ciphersuite Search. https://ciphersuite.info, 2022. Accessed: 2022-10-31.

[219] Webpage: Search Engine for the Internet of Everything. https://www.shodan.io, 2023. Accessed: 2023-02-13.

[220] Webpage: The Censys Platform. https://censys.io, 2023. Accessed: 2023-06-04.

Appendix A
Publications

A considerable part of this work (Section 2.2 and Chapter 4) was implemented in the context of the CyberSANE project funded by the European Commission under Horizon 2020[1]. More specifically, FORTH participated in and led Task 3.3 "Encrypted Network Traffic Analysis" and contributed to the deliverable D3.2 "Encrypted Network Traffic Analysis, Transformation and Normalization Techniques" [207], where I had the opportunity to describe the motivation for my work, develop a system, test and demonstrate it[2] via one use case defined in the energy pilot (Lightsource Labs[3]). The pilot demonstrated, tested and validated the CyberSANE System, by showcasing a variety of potential cyber-attack scenarios within the Solar Energy management platform, used for a number of digital services such as helping secure the electrical grid and reducing the cost of electricity. Our work in the context of CyberSANE produced four publications [89, 91, 92, 95]. In addition, our work in Task 3.3 was selected by the European Commission's Innovation Radar in the domain of secure networks and computing. Section 2.1 describes the work conducted during my 6-month internship in Niometrics[4], back in 2018. There, I had the opportunity to identify the need to build a system able to inspect encrypted network traffic in real-time, solely by monitoring network packet metadata. After the end of the internship, we published a paper describing our work in mobile application analytics against encrypted network traffic [90]. In addition, I was honoured to participate in a report published by ENISA in 2019

[1] https://www.cybersane-project.eu
[2] https://www.cybersane-project.eu/cybersane-pilot-case-study-2/
[3] `https://www.lightsourcelabs.com/`
[4] `http://niometrics.com`

demonstrating the state-of-the-art in the domain of encrypted traffic analysis [186], while our extensive literature review was published in [92]. Finally, part of Chapter 3 is presented in [93].

Publications (peer-reviewed)

The research activity related to this thesis has so far produced the following publications (ordered by publication date):

1. *Pump Up the JARM: Studying the Evolution of Botnets Using Active TLS Fingerprinting.* Eva Papadogiannaki, Sotiris Ioannidis. 2023 IEEE Symposium on Computers and Communications (ISCC). [93]

2. *Network Intrusion Detection in Encrypted Traffic.* Eva Papadogiannaki, Giorgos Tsirantonakis, Sotiris Ioannidis. 2022 IEEE Conference on Dependable and Secure Computing (DSC). [95]

3. *A survey on encrypted network traffic analysis applications, techniques, and countermeasures.* Eva Papadogiannaki, Sotiris Ioannidis. 2021 ACM Computing Surveys (CSUR). [92]

4. *Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware.* Eva Papadogiannaki, Sotiris Ioannidis. 2021 Sensors. [91]

5. *Head (er) Hunter: fast intrusion detection using packet metadata signatures.* Eva Papadogiannaki, Dimitris Deyannis, Sotiris Ioannidis. 2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). [89]

6. *Otter: A scalable high-resolution encrypted traffic identification engine.* Eva Papadogiannaki, Constantinos Halevidis, Periklis Akritidis, Lazaros Koromilas. 2018 International Symposium on Research in Attacks, Intrusions, and Defenses (RAID). [90]

Publications (deliverables/reports)

1. *Encrypted Network Traffic Analysis, Transformation and Normalization Techniques.* Sergio Zamarripa Lopez, Spyros Papastergiou, Nicola Tamburini, Andrea Pruccoli, Eva Papadogiannaki, Manos Athanatos, Konstantinos Kontakis, Sofia Spanoudaki, Georgia Koutsouri, Marianna Manou Kaklamani, Saoulidis Harris. Deliverable 3.2 CyberSANE. August 2021. [207]

2. *Encrypted Traffic Analysis: Use Cases & Security Challenges.* Paraskevi Dimou, Jan Fajfer, Nicolas Muller, Eva Papadogiannaki, Evangelos Rekleitis, Frantisek Strasak. European Union Agency for Cybersecurity (ENISA). November 2019. [186]

Posters (peer-reviewed)

1. *GPU-accelerated encrypted network traffic inspection.* Eva Papadogiannaki, Sotiris Ioannidis. 6th ACM-W Europe Celebration of Women in Computing (womENcourage, 2019).

2. *High performance encrypted network traffic inspection using hardware accelerators.* Eva Papadogiannaki, Giorgos Vasiliadis, Sotiris Ioannidis. Presented in the 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT, December 2018) and in the 1st Summit on Gender Equality in Computing (GEC, 2019).

Highlights

1. *Detection signatures for encrypted traffic.* Innovation selected by the European Commission's Innovation Radar in the context of the CyberSANE project funded by the European Commission under Horizon 2020 (www.innoradar.eu/innovation/40424).