THE UNIVERSITY *of* EDINBURGH

Safeguarding User Security and Privacy: Seeing through Communications and Obfuscating Network Activities

Haoyu Liu

Doctor of Philosophy
Institute of Computing Systems Architecture
School of Informatics
University of Edinburgh
2023

Abstract

The prevalence of the Internet creates a virtual world that connects billions of devices based on a stack of network protocols, enabling users who are physically distant to communicate seamlessly. While the accessibility of the Internet catalyzes a range of innovations, including web applications, mobile phones, peer-to-peer systems, and industrial/domestic Internet of Things (IoT), this heterogeneity also introduces multi-faceted threats to daily users. From a security perspective, network attacks originating from the vast IP space are challenging to differentiate from normal connections, causing financial losses worth trillions of dollars each year. The privacy threats are even stealthier as end users are often exposed to highly encapsulated interfaces, and remain unaware of the data transmission occurring behind the scenes, which opens up the opportunity for data over-collection, interception and Internet censorship.

This thesis addresses critical threats in the domain of network security and privacy by leveraging a combination of advanced data-driven techniques and reverse-engineering. In terms of user privacy, we examine the network connections generated by different distributions of the Android mobile operating system (OS) running on popular smartphone, including Samsung, Huawei, Xiaomi, Oppo and Oneplus. For this, we employ an extensive repertoire of techniques such as Man-in-the-Middle (MITM) interception, code reverse-engineering and dynamic hooking. The privacy analysis conducted reveals extensive private data collection in the background without user consent across all brands, which constitute breaches of the GDPR. Moreover, the firmware released for different regions, e.g., the Chinese versions, follows less stringent privacy standards and leaks considerably more private information at the OS level, including geo-location, personal identifiers and social-relationships. These behaviors do not change when the devices move outside China, posing threats to business travelers, students studying aboard and also tourists purchasing phones from China. Secondly, we introduce Amoeba, a novel black-box adversarial approach targeting ML-supported Internet censorship. The proposed attack strategy frames the task of finding adversarial network flows as a sequence generation problem, and employs reinforcement learning to interact with censoring classifiers and to learn a policy that generates 'harmless' network flows. This is the first black-box attack specifically designed against traffic analysis models, achieving $\sim$94% Attack Success Rate (ASR) on average, which is comparable to white-box algorithms. The trained version of Amoeba can be also deployed as a transport layer extension. Thirdly, we analyze a range of large-scale network attacks and identify key temporal interrelations between illicit traffic in the wild,

a factor that previous studies largely overlooked. Based on the analysis, we design an original Deep Learning (DL)-based Network Intrusion Detection System (NIDS), incorporating a feature extractor, a temporal aggregator, a data augmentor and a novel architecture - Bidirectional Asymmetric LSTM (Bi-ALSTM). The proposed system outperforms existing solutions by at least 33% in terms of F1 score under a rigorous testing environment. Finally, the thesis discusses potential future research directions that are inspired by the results obtained throughout this PhD project.

Lay summary

The rapid evolution of network infrastructure has resulted in billions of devices being interconnected. As a result, network traffic and the information exchanged have grown and diversified significantly. This, coupled with the complexity of network architectures, protocols, and application scenarios, has led to various security and privacy concerns. Often, solutions aimed at solving one issue inadvertently create other problems that are difficult to anticipate and resolve. Security and privacy challenges vary across different parts of a network architecture, necessitating individual examination and solutions. This study adopts an abstract perspective of network communications, encompassing three key parties: senders, intermediate links, and recipients.

At the senders side, with the rising prevalence of smartphones, mobile operating system (OS) privacy has become a critical topic. Despite improvements made in multiple version updates, Android continues to grapple with privacy concerns. Many studies focus on whether mobile apps disclose sensitive information, but the OS itself could potentially leak private data to back-end servers. In this thesis, we perform an in-depth measurement study of the data collected by popular proprietary variants of the Android OS, developed by Samsung, Xiaomi, Huawei, Realme and OnePlus. Our analysis reveals that all the variants transmit a worrying volume of private data to OS developers and third parties, including device-identifiers, configuration data and telemetry. Cross-regional differences exist in terms of privacy violations. The customized Android OS distributions released in China transmit a concerning volume of Personally Identifiable Information (PII) not only to device vendors, but also to Chinese mobile network operators and over-the-top service providers, irrespective of the SIM card usage or location. The data includes persistent device identifiers, location identifiers, user profiles, and social connections. These data collection behaviors remain unchanged when devices are used outside China, potentially subjecting users to continued tracking and privacy violations, even in jurisdictions with stronger data protection laws.

Delivering messages from senders to recipients can be fraught with challenges. Man-in-the-middle (MITM) attacks, involving either passive observation of traffic or active tampering with the original payload, occur in the midst of connections. Network censorship poses a catastrophic risk to free Internet access. By employing various techniques, including blacklisting, deep packet inspection (DPI), active probing, and machine learning (ML)-based traffic analysis models, censors can infer the contents or the underlying protocols used during network communications, effectively blocking undesired traffic. To tackle the ever evolving threat of Internet censorship fueled by

Machine Learning (ML) classifiers, we introduce Amoeba, a novel black-box attack utilizing reinforcement learning, which crafts adversarial flows based solely on the classification results of censoring classifiers, requiring no additional knowledge. The generated adversarial flows can effectively spoof censoring engines, with around 94% attack success rates. Amoeba is demonstrated to be stable and robust under different network environments with noisy feedback from the censoring classifiers. The adversarial flows are transferable across different models, and thus capable of subverting unknown classifiers.

From the perspective of recipients, receiving all traffic indiscriminately can pose substantial risks as not all communications are well-intentioned, and distinguishing these intentions can be challenging. It is crucial to implement precautions to protect servers from overload or exploitation. Network Intrusion Detection Systems (NIDS) serve as a viable solution to examine traffic at the network layer, enabling early detection of intrusions. We design NetSentry, a robust deep learning-based NIDS that is designed for detecting the early phases of cyberthreats. NetSentry incorporates a series of effective techniques, including feature augmentation and a novel deep learning block - Bidirectional Asymmetric LSTM (Bi-ALSTM), and is evaluated on large cybersecurity datasets. The results obtained confirm that NetSentry outperforms a collection of existing tools.

We conclude this thesis by identifying potential future research directions. Overall, this thesis addresses the security and privacy issues of different parts involved in network communications and makes efforts in tailoring deep learning to network activity analysis and obfuscation.

Acknowledgements

I still feel surreal when writing this part of the thesis, as my PhD journey is coming to the end. First of all, I would like to express my sincere gratitude towards my principal supervisor, Dr Paul Patras, who consistently offered me support and guidance throughout this PhD project. The biggest influence Paul had on me was not achieved directly through language and communication. Instead, I perceived optimism, persistence, and kindness in Paul. These qualities, like a stream flowing gently, have been continually influencing me for the past four years. They have taught me not only how to do academic work better, but also how to be a better person. Without Paul's constant help, I could not have completed my PhD project. However, more than completing the PhD, what I am truly grateful for are the qualities I've felt and learned, which will benefit me for a lifetime.

Secondly, I would like to thank my second supervisor Hannes Tschofenig, and Kami Vaniea, who provided me valuable comments and insights on my annual reviews. Their advice, coming from different academic fields, has aided me in viewing my research problems from a variety of perspectives. Also, many thanks to the anonymous reviewers of my submitted and published papers. The comments that recognized my contributions made me realize that efforts might pay off, while the dissenting comments made me understand that reconciliation with oneself is even more important. I want to thank all the current and former members of the Mobile Intelligence Lab. Rui Li and Chaoyun Zhang kindly helped me adapt to the new life at the beginning of my PhD. It was my honor to work with Alec F. Diallo, Yini Fang, Luyang Xu, Weihe Li and Rupen Mitra, and to have discussions with them, during which we inspired each other and better shaped our ideas.

I am thankful to Scotland's Innovation Centre for sensing, imaging and Internet of Things (IoT) technologies (CENSIS), Arm Ltd and the School of Informatics, which provided me with funding in support of this PhD project.

Pursuing a PhD is challenging and requires a great deal of patience and a good rhythm of life. In the past 4 years, I met many people and made a lot of friends. Various activities have enriched our lives and brought a great deal of joy. I am grateful to those I played badminton with: Hao Yu, Samuel Knight, Danny, Hao Chen, April, Panda, Leslie and many more from ECBC. I thank those I played board games regularly with: Guanhao Su, Nancy Wang, Wei Chen, Zirong Zeng, Sa Yin and many more. Many thanks to all my friends who came and already left Edinburgh, as their presence has left me with countless memorable moments.

Special thanks to my girlfriend, Jiexuan Liu, who has always supported me emotionally, expressing her joy and sorrow authentically. Her presence is undoubtedly a stabilizing factor in my life. Finally, I want to express my gratitude to my parents and grandparents who raised and selflessly nurtured me. Without them, I would not have had the opportunity to step out of Xi'An to Shenzhen, and then study abroad in Edinburgh, experiencing a broad and diverse world.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Parts of this work have been published in or submitted to academic conferences and journals, including:

1. **Liu, H.**, Diallo, A. and Patras, P., *"Amoeba: Circumventing ML-supported Network Censorship via Adversarial Reinforcement Learning"*, (Technical report, Under review) [98].

2. **Liu, H.**, Leith, D. and Patras, P., *"Android OS Privacy Under the Loupe – A Tale from the East"*, in 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec), May, 2023 [99].

3. **Liu, H.**, Patras, P., and Leith, D., *"On The Data Privacy Practices Of Android OEMs"*, in PloS ONE 18.1 (2023): e0279942 [101].

4. **Liu, H.**, and Patras, P., *"NetSentry: A Deep Learning Approach to Detecting Incipient Large-scale Network Attack"*, Computer Communications 191 (2022): 119-132 [100].

5. Maroto, J., **Liu, H.**, and Patras, P., *"On the Struggle Bus: A Detailed Security Analysis of the m-tickets App"*, in 23rd International Conference on Information Security (ISC 2020), Dec. 2020 [138].

(*Haoyu Liu*)

Table of Contents

Acronyms

AnoEAN Encoding Adversarial Network. 27

AOSP Android Open Source Project. 7, 63

ARIMA Autoregressive Integrated Moving Average. 11

ASR Attack Success Rate. 93–100, 133

AUC Area Under Curve. 123, 125

BAP Blind Adversarial Perturbation. 23, 90, 91, 93, 94, 97

Bi-ALSTM Bidirectional Asymmetric LSTM. vi, xiii, 117–119, 122, 124, 125, 127–129, 132

Bi-LSTM Bidirectional LSTM. 116, 117, 121, 122, 124, 125, 128, 129, 131

BuFLO Buffered Fixed-Length Obfuscation. 21

CIC Canadian Institute for Cybersecurity. 120

CNN Convolutional Neural Networks. 11, 26, 89, 116, 117, 121, 122

ConvLSTM Convolutional LSTM. 116–118, 120, 122, 125, 127

CS-BuFLO Congestion-Sensitive BuFLO. 21

CSRF Cross-Site Request Forgery. 107

CVAE Conditional Variational Autoencoder. 27

CW Carlini & Wagner. 74, 90, 91, 93, 94

DAGMM Deep Autoencoding Gaussian Mixture Model. 121, 122

DDoS Distributed Denial of Service. 24, 25, 103, 105, 106, 132

DF Deep Fingerprinting. 89, 90, 96, 98

DL Deep Learning. 8, 10, 11, 14, 21, 24, 102, 104, 134, 135

DMM Dirichlet Mixture Model. 27

DoS Denial of Service. 14, 25, 107, 110, 113, 125

DPI Deep Packet Inspection. 6, 20, 22, 23, 71

DT Decision Tree. 11, 75, 90, 95, 96, 98

ECDF Empirical Cumulative Distribution Function. 98, 99, 124, 128

FGSM Fast Gradient Sign Method. 23

FNR False Negative Rate. 129

FP False Positives. 122, 131

FPR False Positive Rate. 20, 123, 128

FTE Format Transforming Encryption. 22

GAN Generative Adversarial Networks. 23, 27, 74, 90, 137

GDPR General Data Protection Regulation. 29

GFW Great Firewall. 20, 136

GMM Gaussian Mixture Models. 27

GRU Gated Recurrent Unit. 26, 82, 83

HIDS Host-based Intrusion Detection Systems. 8

IAT Inter-Arrival Time. 8, 110, 135

IPS Intrusion Prevention System. 131

ISP Internet Service Providers. 71

KNN K-Nearest Neighbours. 26

LRCN Long-term Recurrent Convolutional Networks. 117

LSTM Long Short-Term Memory. 11, 26, 83, 90, 95, 98, 109, 115–119, 125

MACs Multiply-Accumulate Operation Counts. 124, 131

MITM Man-In-The-Middle. 5, 7, 10

ML Machine Learning. vi, 8, 9, 13, 15, 20–22, 24–26, 72, 78, 90, 103, 113, 129, 132, 133, 135, 136

MLP Multilayer Perceptron. 26, 84, 85, 90, 116, 121

MSE Mean Squared Error. 82

NID Network Intrusion Detection. 26, 103, 104, 107, 117, 119, 132

NIDS Network Intrusion Detection Systems. vi, xi, xiii, 6, 9, 15, 24, 25, 103, 104, 106, 107, 109, 122, 129, 131–133, 135, 136

NMAE Noromalized Mean Absolute Errors. 92

NN Neural Network. 13, 75, 90, 91, 93, 95

OC-NN One Class Neural Nets. 121, 122

OS Operating System. 106, 120

PDF Probabilistic Density Function. 27, 121

PII Personally Identifiable Information. xi, 17–19, 31, 40, 103

PPO Proximal Policy Optimization. 85, 86

RF Random Forest. 90, 95, 96, 98

RL Reinforcement Learning. 13, 15, 75, 78, 85, 87, 133, 136, 137

RNN Recurrent Neural Networks. 11, 26, 115

ROC Receiver Operating Characteristic. 123, 125

SDAE Stacked Denoising Autoencoder. 90, 98

SMB Server Message Block. 106, 107

SVM Support Vector Machine. 11, 75, 90, 95

TCBs Transmission Control Blocks. 72

TLS Transport Layer Security. 5, 8

TN True Negatives. 122

TP True Positives. 122

TPR False Positive Rate. 123

TRPO Trust Region Policy Optimization. 85

VoIP Voice over Internet Protocol. 134

WF Website Fingerprinting. 8, 21, 73, 89, 90, 135

XSS Cross-Site Scripting. 25, 106, 128, 132

Chapter 1

Introduction

As network infrastructure continues to evolve, billions of devices ranging from laptops and mobile phones carried by individuals to dedicated cloud and data centers hosting a variety of services, are interconnected via the Internet. As a result, the volume of traffic flowing over networks has increased dramatically over recent years and the information users exchange has diversified considerably. At the same time, the multitude of network architectures, protocols, and application scenarios has given rise to security and privacy issues that threaten individual and business users alike.

Technical solutions introduced to tackle one specific problem often spawn multiple others that may have been hard to foresee and for which finding solutions becomes non-trivial. For instance, the current network stack is the product of decades of evolution, in which the introduction of Transport Layer Security (TLS), positioned between the transport and application layers, followed after security researchers realized that sending payload in plaintext presented the opportunity for interception and Man-In-The-Middle (MITM) attacks. TLS and the associated Certificate Authorities makes it hard for attackers to control intermediate links, but may inadvertently hide potential privacy issues. For example, modern mobile phones house various applications and services, many of which communicate with back-end servers periodically for synchronization/updates, as well as for data requests and storage. In this process, the TLS-encrypted connections deter potential network attacks, but also make the data transmissions opaque to everyone except for the service providers. In other words, determining whether any sensitive private information violation occurs becomes challenging for both users and researchers. Likewise, in the case of network attacks targeting web servers, such as SQL injection, if the use of TLS is enforced then SQL injection payloads are not visible to network managers and therefore can easily by-

pass network firewalls. Encryption inhibits the use of Deep Packet Inspection (DPI) and content matching to design NIDS. On the other hand, encryption also discourages network censors from discovering whether 'sensitive' content is accessed by users.

Security and privacy issues are unique to different parts of a network architecture, and therefore a case-by-case scrutiny and resolution are required. Rather than enumerating all the possible issues by service, product or protocol, in this work we start from an abstract view of network communications, as shown in Figure 1.1. This abstraction describes the nature of any network activity, consisting of three major parties, namely senders, intermediate links and recipients. We can thus categorize network security and privacy issues by the components in this abstraction. Before diving into research questions, we consider it is crucial to clarify the differences between security and privacy. Although these terms often appear together, they essentially refer to distinct concepts.

1. *Network Security* concerns the protection against cyberthreats or security hazards. Security is closely linked with technical specifications, encompassing network protocols and systems engaged in network activities. An insecure design could lead to aberrant behavior in communication protocols or system malfunctions, which may become susceptible to exploitation by malicious actors.

2. *Network Privacy* pertains to the right to control what information is transmitted and the visibility of one's information in the network. While a security vulnerability often leads to a breach of user privacy, it is not a necessary condition. A system may be secure against external attackers, but could exfiltrate user information intrinsically, which is related to our work in Chapter 3. Similarly, network protocols may be fortified with end-to-end encryption, but observers can still infer the coarse intent of the communications. This aspect is explored in Chapter 4.

In what follows we describe the issues specific to each component, along with key research questions that this thesis addresses.

1.1 Key Research Questions

Senders - Mobile OS Privacy: Mobile OS privacy has become an increasingly important topic especially in the last two decades, given the growing prevalence of smartphones. It is estimated that around 6.92 billion people will be using smartphones in 2023, accounting for 86.11% of the world population [39]. Android and iOS are

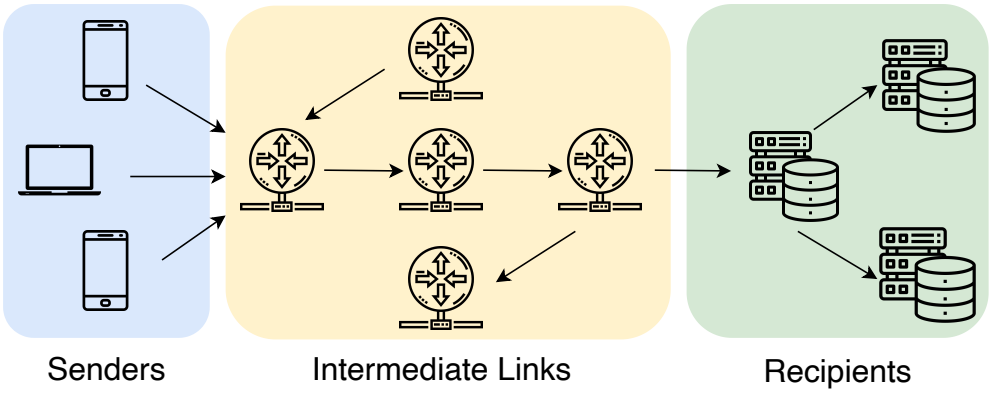


Figure 1.1: Abstraction of network communications

the two most dominant mobile operating systems worldwide, each characterized by unique features. Android offers a highly customizable user experience and a flexible ecosystem, albeit with certain security issues, such as malware and broken permission management. Through multiple version updates, Android has made significant improvements to fix known security issues, but a number of privacy concerns persist. For example, the analysis of whether mobile apps disclose sensitive information has been the focus of much research [77, 129, 126, 128, 173]. However, not only mobile apps but also the mobile OS itself may leak private information to the back-end servers. This risk can be more challenging to detect since smartphone vendors are able to customize the Android OS firmware based on their needs, owing to the flexibility of Android Open Source Project (AOSP). Apps installed by users may have limited access to sensitive data due to the users' privacy awareness or system-imposed restrictions, while the OS has unlimited access and may pose greater threats to individual users if misconfigured or deliberately configured in an invasive way. This motivated me to conduct research that seeks to answer the following questions:

- Is private data leaked from customized Android OS?
- What types of private information would be transmitted and where are the destination endpoints?
- Are there differences among smartphone brands in terms of private information collection?
- Are there regional differences in terms of private information collection?

Intermediate Links - Circumventing Ever-evolving Internet Censorship: Transmitting messages from senders to recipients is not always plain sailing. MITM at-

tacks refer to those network attacks that occur in the middle of connections, which involves either passive observation of the traffic or active tampering with the original payload. The use of TLS, to a large extent, helps prevent such attacks but does not fully eliminate privacy concerns when an intermediate hop is controlled by an adversary. Specifically, Website Fingerprinting (WF) is the technique of inferring with visited websites in encrypted network traces purely based on statistical information, such as bidirectional packet distributions and packet Inter-Arrival Time (IAT). State-of-the-art WF can achieve 98% accuracy against Tor traffic in a fully controlled environment [149]. If WF only served to violate user privacy, then network censorship could have a catastrophic impact on the free access to the Internet. By leveraging a collection of techniques, including blacklisting, DPI, active probing and ML-based traffic analysis models, censors are capable of inferring the contents or the underlying protocols used during network communications and focus efforts in blocking unwanted traffic. The ever-evolving censorship approaches significantly threaten people's right to access media and their freedom of expression in the censored regions, especially when ML techniques are used for detecting sensitive traffic at machine speed. To tackle network censorship, we conducted research aiming to answer the following questions:

- How feasible is it to leverage ML/Deep Learning (DL) techniques for Internet censorship?
- How can we circumvent ML-supported censorship?
- Can we circumvent ML-supported censorship by attacking the underlying models rather than perpetually designing new network obfuscation protocols?

Recipients - Discovering Network Attacks at the Network Layer: Blindly receiving all the traffic arriving at a receiver can pose significant risks, as the intentions behind these communications are not always out of good will (requesting services versus attacks). Determining the intentions behind these requests can be challenging especially at the level of back-end servers/data centers, due to the vast volume of traffic and the anonymity of transport protocols (as IP can hardly be linked with real identities). Implementing precautions is essential to protect servers from being overloaded or exploited. Host-based Intrusion Detection Systems (HIDS) is deployed on the hosts to inspect data that originates from the host system, such as window server logs, firewalls logs, application system audits, or database logs. This approach has the merit that the payload of all the connections are visible such that decisions made are with high

confidence and few false alarms. A limitation, on the other hand, is that such a defense method requires server/system maintainers themselves to have a certain level of security awareness and experience, which is not always a given. NIDS is another option that monitors the network traffic extracted through packet capture. It is deployed at Internet Service Providers (ISP) or organizational gateways, allowing for the joint examinations of network connections to multiple destinations, and thereby enabling earlier intrusion detection. Traditional NIDS largely apply finite rules preset by human experts to detect intrusion activities, which lacks flexibility and is prone to subversion. Meanwhile, ML-based NIDS are consistently evaluated in a single, controlled environment, which questions their true generalization abilities. Developing reliable and accurate NIDS is crucial for safeguarding network security, and in this thesis we conducted research that aims to elucidate the following questions:

- Are existing ML-based NIDS robust in different network environments?
- How should we evaluate the generalization abilities of a model/system with limited access to network traffic?
- What is the most suitable ML approach to detecting generic intrusions: supervised, semi-supervised or unsupervised?
- What improvements can be made to the existing training routines of ML-based NIDS?

1.2 Research Methodology

Our research objectives span different components of the network, and therefore the research methods employed vary to match the specifics of the associated tasks. In this thesis, we mainly adopt two sets of research approaches: analysis-driven and data-driven.

Analysis-driven Approaches. The analysis of Android OS privacy is largely reliant on manual inspection, given that each customized OS comes pre-loaded with various system packages. These packages each have distinct communication procedures with relevant backend servers and specific focus on the range of private data handled. To accurately understand the privacy practices of a customized Android OS, it is critical to scrutinize every bit of transmitted information in plaintext, rather than simply inferring

speculatively what private data may be sent to the backend servers. To achieve this aim, we utilized a set of analysis approaches, including

1. Device rooting: Examining Android OS requires having unlimited access to the devices themselves, thus device rooting is a necessity, and the premise of the following steps.

2. MITM Interception: The use of TLS blocks researchers from observing network connections generated by the mobile OS. MITM interception plants a self-signed certificate into the devices, so that traffic can be decrypted before reaching the intended destinations.

3. Code Analysis and Dynamic Hooking: It is common that app developers would apply an extra layer of encoding or encryption before transmitting sensitive information. Finding how the encoding/encryption works requires tracing back to the source code. Sometimes dynamically hooking the apps is needed, especially when asymmetric encryption is applied.

4. Permission Analysis: Only knowing what information is leaked is not the end of the research but knowing why the information can be accessed and leaked is also important, which is the essential reason for diving into the permission system and inspecting the permissions assigned to each system apps.

Data-driven Approaches. As we shift our focus to the intermediate links and recipients in the network, the traffic volume aggregated from millions of connections becomes significantly larger than what individual devices can generate. The careful inspection of each flow becomes impractical, especially considering that the prevalent use of encryption largely discourages any form of manual inspection. Instead, data-driven approaches, specifically those based on DL, emerge as more effective and efficient in analyzing substantial volumes of traffic data. The following advantages of DL in network security and privacy are noteworthy.

1. Scalability: Deep learning algorithms are scalable to massive datasets, as they can extract features and learn statistical patterns from a large amount of data by utilizing stochastic optimization. This is particularly suitable for analyzing the vast volume of network traffic going through any intermediate links or servers.

2. Powerful Pattern Recognition: Unlike visual data or semantic data, network traffic is not directly interpretable by humans. Single statistical features, extracted

from network flows, can seldom serve as definitive indicators of specific network activities. Furthermore, identifying patterns in high-dimensional features often exceeds the capability of traditional methods. DL algorithms, on the other hand, excel in finding complex patterns in high-dimensional data because of the inherent nonlinearity of the deep neural networks utilized.

3. Automated Feature Extraction: The process of extracting features from network traffic traditionally relies heavily on expert knowledge, and sometimes, due to human limitations, important features may be overlooked. DL is capable of extracting features from raw packet/flow sequences via multiple intermediate layers. For instance, Convolutional Neural Networks (CNN) that are used for automated traffic feature extraction have achieved significant success [149, 113].

4. Friendly to Sequential Data: Network traffic involving timestamps by nature is a form of sequential data, at both packet level and flow level. Traditional ML algorithms, such as Support Vector Machine (SVM), Decision Tree (DT), and Autoregressive Integrated Moving Average (ARIMA), either struggle to handle time-sensitive data effectively, or are limited to low-dimensional timeseries data. In contrast, Recurrent Neural Networks (RNN) such as Long Short-Term Memory (LSTM) are specifically designed to extract the temporal correlations and patterns from sequential inputs. They generate latent representations which can be utilized for downstream tasks, such as classification.

1.3 Contributions

This thesis addresses research challenges facing network security and privacy as identified above. In particular, we leverage both analysis-driven and data-driven approaches and tailor them to individual problems, leading to the following contributions:

1.3.1 Privacy Analysis of Customized Android OS

I conduct an in depth measurement study of the data shared by a range of popular proprietary variants of the Android OS, namely those developed by Samsung, Xiaomi, Huawei, Realme and OnePlus. In addition, we report on the data shared by the LineageOS and /e/OS open-source variants of Android. Samsung currently has by far the largest share of this market, followed by Xiaomi, Huawei, Oppo (the parent com-

pany of Realme) and OnePlus [73]. LineageOS is probably the most popular open-source Android variant, currently used on around 30M handsets,[1] while /e/OS is a new privacy-focused fork of LineageOS.

We find that the Samsung, Xiaomi, Huawei and Realme Android variants all transmit a substantial volume of data to the OS developer (i.e. Samsung etc) and to third parties that have pre-installed system apps (including Google, Microsoft, Heytap, LinkedIn, Facebook). LineageOS sends similar volumes of data to Google as these proprietary Android variants, but we do not observe the LineageOS developers themselves collecting data nor pre-installed system apps other than those of Google. Notably, /e/OS sends no information to Google or other third parties and sends essentially no information to the /e/OS developers.

Moreover, we perform a cross-regional analysis and compare the preinstalled system apps on the Chinese (CN) and Global (e.g., EU) Android OS distributions from the same OS developers. We find that the number of preinstalled third-party apps on CN OS distributions is 3 to 4 times larger than for the corresponding Global OS distribution, and that these are given 8 to 10 times as many permissions as third-party apps in Global distributions, including many more permissions classed as dangerous.

We find that the Chinese smartphones studied send a worrying amount of Personally Identifiable Information (PII) not only to the device vendor but also to Chinese mobile network operators (e.g., China Mobile and China Unicom), even though they do not provide any service to the device, i.e., a SIM card has not been inserted or we use a SIM card that ensures connectivity to a different operator in China or in the UK, and to over-the-top service providers (e.g., Baidu). The data we observe being transmitted includes persistent device identifiers (IMEI, MAC address, etc.), location identifiers (GPS coordinates, mobile network cell ID, etc.), user profiles (phone number, app usage patterns, app telemetry), and social connections (call/SMS history/time, contact phone numbers, etc.). Combined, this information poses serious risks of user deanonymization and extensive tracking, particularly since in China every phone number is registered under a citizen ID. Moreover, the data collection behaviors do not change when the devices move outside China, despite potentially being under jurisdictions where users should benefit from stronger data protection, meaning that phone vendors and some third-parties are still able to track business travelers and students studying abroad, including the foreign contacts they make on their visits.

Overall, our findings paint a troubling picture of the state of user data privacy in

[1] `https://stats.lineageos.org/`, accessed 31st July 2021

the world's two largest Android markets, and highlight the urgent need for tighter privacy controls to increase the ordinary people's trust in technology companies, many of which are partially state-owned.

1.3.2 Network Traffic Obfuscation Against ML-supported Censorship

Censorship is an arms race. Recent studies revealing that ML algorithms, which learn statistical features from network flows, can effectively identify 'offending' tunneled traffic, despite not exhibiting deterministic fingerprints [29, 170]. As the second contribution in this thesis, we formulate the problem of finding adversarial flows against censoring classifiers as a packet sequence generation task. To solve it, we design Amoeba,[2] a novel black-box attack through reinforcement learning, which learns to craft adversarial flows solely based on the classification results of censoring classifiers as negative rewards and progresses with a policy that maximizes the expected future rewards (return), without any further knowledge.

Specifically, we make no assumption about the underlying model of a censoring classifier, which may or may not apply feature engineering and may not be differentiable (and hence approximating gradients impractical for generating adversarial flows), but instead treat the problem of finding adversarial flows as a process of generating sequences of packets that, when considered together as flows, will be misclassified.

Our design incorporates a StateEncoder – a dedicated Neural Network (NN) that encodes arbitrarily long network flows into fix-sized hidden representations, to help the Reinforcement Learning (RL) agent interpret the context of sequence generation at each timestep.

We evaluate Amoeba on datasets collected using two popular anti-censorship systems, Tor and TLS tunneling; our experimental results indicate that the adversarial flows generated by our Amoeba have $\sim$94% attack success rates, regardless of the type of ML classifier a censor may deploy. We further show empirically that such adversarial flows are transferable across different models, and thus capable of subverting unknown classifiers.

[2]Our censorship circumvention algorithm's name draws inspiration from the unicellular organism with the same name that is capable of altering its shape. Similarly, our solution alters the shape of network flows.

We demonstrate that the Amoeba is stable in different network environments, and robust when receiving noisy and unclear rewards during training.

Finally, we discuss practical aspects of deploying our Amoeba as a transport layer extension, making the case for its adoption for mainstream censorship circumvention.

1.3.3 Network-based Intrusion Detection via Deep Learning

The third contribution of this thesis attacks the network-based intrusion detection problem and proposes a novel DL-based system, NetSentry, focusing on the early stages of cyberthreats that are essential to the success of large-scale and high-impact attacks such as botnet and ransomware.

Firstly, we scrutinize several attack chains and identify key temporal interrelations between illicit traffic occurring in the wild; based on this analysis, we design Bidirectional Asymmetric LSTM (Bi-ALSTM), an original ensemble of sequential neural models that effectively captures the temporal dynamics of malicious traffic and classifies specific threats, including Denial of Service (DoS), Port Scanning, and Brute Forcing.

Since not every attack type can be distinguished accurately with limited information available at the network layer, we introduce a novel training technique that relies on feature augmentation and abstract labeling. The feature augmentation scheme improves the heterogeneity of cyberattacks that were collected in a controlled environment, which helps NN models learn a more robust decision boundary. Abstract labeling, on the other hand, prevents overfitting by grouping similar types of attacks into one class;

Then, we train our Bi-ALSTM on a large dataset published by the Canadian Institute for Cybersecurity, we cross-evaluate our approach with a previously unseen dataset collected in a different network topology, and we compare its performance against that of state-of-the art benchmarks. Results demonstrate Bi-ALSTM outperforms existing approaches by at least 33% in terms of F1 score;

Lastly, we discuss practical aspects of deploying NetSentry in real-life, including computational overhead and robustness to a range of evasion attacks.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 introduces the background of the research objectives as the preamble of the thesis. Chapter 3 gives a comprehensive privacy analysis of customized Android OS developed by a collection of popular brands and released in the EU and China respectively. In Chapter 4, we present a novel RL-based adversarial attack against ML-supported traffic analysis model, as a new direction to circumvent Internet censorship. Chapter 5 targets the server-side security and introduces a robust ML-based NIDS throttling cyberthreats at the early stage. Finally, we discuss the potential future research directions and conclude the thesis in Chapter 6.

Chapter 2

Background

2.1 Android OS Privacy Analysis

As the Android ecosystem continues to expand, a significant number of smartphone users remain largely oblivious to the Personally Identifiable Information (PII) being divulged by their devices and the applications they utilize [165]. This has motivated extensive privacy and security research over recent years [126, 128], and triggered data protection legislation with nearly 100 articles laying out privacy requirements [41]. In what follows, we delve into the boundary of PII collection from Android OS, methods implemented for privacy analysis concerning mobile devices, and the permission system aiming to limit access to private data.

2.1.1 PII Collection

PII is a generic term referring to information which can be used to identify or trace individual users [161]. The common identifiers include:

1. Device Identifiers: specific to a mobile device or OS distribution, including MAC address, Android ID, Google Advertiser ID, IOS IFA ID and IMEI;

2. User Identifiable Information: identifies individual users, including name, gender, email address and date of birth; Telemetry that logs the use history of specific apps would not directly reveal user identity but it may be possible to infer based on previous behaviors.

3. Geolocation Information: covers both coarse location information, such as MCC, MNC, LAC and nearby Wi-Fi MAC address, and accurate location information

such as GPS Coordinates;

4. Social Relationship: includes phone numbers and contact history.

5. Credentials: includes saved username and password.

It has been shown that a vast amount of PII is directly embedded in queries, headers or post body of HTTP requests. By using ISP traffic logs or self-collected datasets, numerous mobile advertising and tracking services were uncovered, in which persistent identifiers including IMEI, IMSI, MAC address and advertising IDs are often transmitted [164, 174, 126, 115]. Note that most of the services are hosted by third-party companies instead of phone vendors. [130] track the update of more than 500 apps across 8 years and document the evolution of PII collection. Further work examines over 500 apps in the Google Play Store and shows that 76% of them collect and transmits PII insecurely and 34% send PII to third parties [75]. Privacy analyses have also been conducted to compare over 5,000 free mobile apps with their paid versions, demonstrating that paid apps are not necessarily more privacy-aware than their free counterparts, with 34% exhibiting the same data transmission behavior [62]. PII collection is not limited to a specific mobile platform. [131] reveals that the top 100 most popular apps in IOS, Windows phones and Android extensively collects device identifiers, user identifiers and locations. Moreover, a case study on the Covid contact tracing apps was conducted recently, revealing that the Google Play Services integrated in the apps would regularly contact Google servers, making it possible to track users' location via the change of IP addresses [33]. It was reported that both Google and Apple collect a number of PII including IMEI, IMSI and telemetry. IOS devices also transmit nearby Wi-Fi MAC addresses and GPS coordinates, which users cannot control [88].

2.1.2 Mobile Privacy Analysis Techniques

Mobile apps are capable of accessing a series of PII with or without user permission. However, it can be extremely time-consuming to thoroughly examine which type of PII is accessed and uploaded by each app since encryption is always applied. Automated tools thus come into play. PiOS analyzes control flow graphs of binary files and examines the reachability of key PII for iOS applications [36]. FlowDroid proposes a static analysis method by taking the Android Application life-cycle into consideration [6]. AppIntent studies if the transmission of private data is intended by the users through symbolic execution [185]. Taintdroid [37] marks (taints) sensitive data and tracks the

data flow during execution. While the techniques of dynamic tracing create an automated analysis environment, user input generation is also crucial to improve coverage of privacy leaks. [105, 63, 7] automatically generate UI events to trigger as many activities and functionality as possible. [26] proposes a black-box method without the knowledge of app source code to detect privacy leaks through differential analysis.

2.1.3 Customized Android OS and Permission System

The existing mobile privacy analysis is limited to third-party apps in app stores. When an app is installed by users, a permission dialog would usually pop up, notifying which permissions would be accessed. The user is given choices at this step to control the potential leak of PII. However, little attention is drawn to the privacy analysis of customized Android OS and the bundled packages. It should be noted that the privacy leak at the OS level, if exists, would be more alarming since users are unaware of which packages are preinstalled and what permissions they are granted by default. Android categorizes permissions into the following types [57]:

1. *install-time permissions*: give the app limited access to data or allows the app to perform limited actions, such as network access and prevent phone from sleeping. Install-time permissions are displayed to the user when the app is installed and the system would automatically grant these permissions. Signature permissions are a special type of normal permissions, and would only be granted if the app is signed by the same certificate as the app or the OS that defines the permission.

2. *runtime permissions*: are also called dangerous permissions which give the app additional, and substantially sensitive access to the phone, including accessing photos, obtaining geolocation and accessing call log. Runtime permissions need to be granted by user during runtime by default.

3. *special permissions*: correspond to particular app operations which only the platform and OEMs can define. Special permissions are usually defined to protect access to particularly powerful actions, such as drawing over other apps or picture-or-picture.

In theory, Android OS customization allows vendors to pre-install apps, modify the canonical permission system and grant runtime permission to the preinstalled packages. Unfortunately, previous work on Android customization analysis mainly focuses

on the security side [2]. It was shown that vulnerable security configurations exist at a large scale in custom Android firmware, and these may potentially lead to a series of privacy attacks, such as stealing emails and altering system settings without proper permissions [2]. [179] found that due to vendor customization, more than 80% of pre-loaded packages are over-privileged, some of which can be exploited for permission re-delegation attack or privacy leaks. The customization of device drivers by Samsung is found to underpin various attacks, including taking a photo or screenshot without permissions [193]. These security concerns also hold from the privacy perspective, since the phone vendors or third-party developers may exploit them and leverage pre-installed packages to access private data stealthily.

2.2 Internet Censorship and Traffic Obfuscation

Internet censorship is carried out in a number of countries in the world, including China, Iran, Russia and India, to block unwanted communications/services. It is achieved by a collection of techniques including IP filtering, DNS poisoning, DPI and active probing [38, 170].

IP filtering and DNS poisoning is the most straightforward method to prevent users from establishing connections. Censors leverage this approach when there is a list of specifically prohibited targets. For example, both DNS resolution and TCP connections to Google Services fail in China, as Google is on the Great Firewall (GFW)'s blacklist [67]. DPI is used to examine network flows when their destination IPs are not forbidden per se but the connections look suspicious to the censor. Besides, DPI can inspect application-layer contents for protocol identification. Tor clients use a unique cipher suite during TLS handshakes, which allows the GFW to narrow down the suspected targets of Tor connections [48]. Active probing involves sending carefully crafted probes to suspicious servers to determine whether they support forbidden protocols, which was tested effective to a range of popular censorship circumvention systems, including Tor, Shadowsocks, Lantern and obfuscated SSH [48, 11]. In recent years, ML algorithms (Decision Tree-based, SVM, etc.) were adopted to detect sensitive network flows and demonstrated promising detection rate and relatively low False Positive Rate (FPR). [170, 8].

2.2.1 Relations with Website Fingerprinting

Censorship by ML appears similar to a sibling branch of privacy research, named WF [120, 78, 119, 133, 149, 150], which infers visited websites in network traces based on statistical features, such as bidirectional packet sizes and interarrival time. The implementation of WF also involves ML/DL models that are trained for a multi-classification task with each class corresponding to a target website. ML-supported Censorship, on the other hand, has little interest in the contents in network flows, but care more about the underlying protocols that generate the network traffic. If a protocol, such as Tor [158] and Shadowsocks [22], is deemed suspicious or capable of circumventing traditional censorship techniques, it would be considered as a target for which a ML/DL model can be trained. Since both WF and ML-supported censorship conduct network traffic classification, the models designed for the former can be easily adopted for the latter, which our experiments in Chapter 4 confirmed. However, the defense methods for two techniques are not identical given the different threat models applied.

2.2.2 Obfuscation Approaches

To confront WF, the most straightforward, but not the most efficient approach is to eliminate packet and timing features completely, known as *regularization*. Buffered Fixed-Length Obfuscation (BuFLO) [34], Congestion-Sensitive BuFLO (CS-BuFLO) [16] and Tamaraw [17] enforce a fixed packet rate with regular sequence end times to erase the statistical features exposed from individual websites. Moreover, Walkie-Talkie [171] modifies the browser to communicate in half-duplex mode rather than full-duplex mode, and molds multiple burst sequences together, which introduces both less data overhead and less time overhead. However, at the same time of erasing website fingerprints, regularization approaches introduce unique protocol fingerprints, such as fixed packet rates and half-duplex communications, which can easily be modeled by censorship techniques. Circumventing censorship requires more efforts in shaping sensitive traffic into 'innocuous'-look connections, and the following methodology has been explored over the past decade:

1. *Randomization:* A randomizing approach aims at hiding all static protocol fingerprints, by shaping traffic with a deliberately designed randomized function. A large fraction of Tor traffic contains 568-byte packets due to the use of fixed onion cells plus TLS headers, which illuminates itself among the rest of protocols in the network. ScrambleSuite [176], obfs3 [159] and obfs4 [186] utilize a

set of probabilistic distributions to determine the sizes of packets to be sent, as an attempt to hiding canonical Tor connections.

2. *Protocol Mimicry:* Morphing sensitive traffic into a benign protocol is another direction of censorship circumvention, which are effective to fool DPI. Format Transforming Encryption (FTE) [35] is a lightweight, cryptographic approach that generates the ciphertext containing keywords of common web protocols, such as `HTTP GET`, to mislead protocol identification performed by DPI. TrafficMorph [178] and SkypeMorph [109] statistically transform packet sizes such that the packet distribution in general resemble the traffic generated by a cover protocol, such as Skype.

3. *Tunneling:* However, morphing sensitive traffic into a cover protocol is fundamentally imperfect, because it is impossible to faithfully mimicking every facet of a protocol, including handshakes, error handling or even implementations bugs [69]. One way to bypass the flaws of protocol mimicry is directly tunneling covert traffic into a cover protocol, such that it is still the cover protocol that takes responsibility of building and handling the communication links, and inside the covert traffic is purely treated as contents. Meek [43] tunnels Tor traffic over HTTPS connections. Protozoa [9] hijacks the WebRTC stack in Chromium and transmits hidden messages through real-time video streaming apps. V2ray [163] supports a range of tunnels including HTTP, TLS and Shadowsocks. DeltaShaper [10] transforms covert data into images and transfers them in Skype video calls.

As introduced above, the existing countermeasures target traditional censorship techniques such as DPI or active probing. The emergence and application of ML in Internet censorship [170, 8], however, has broken the balance of the arm race, making proposed circumvention approaches detectable in the network. We urgently need a countermeasure against ML-based traffic analysis model.

2.2.3 Adversarial Attacks for Computer Vision

Traffic analysis models, in essence, are a type of ML classification model. Several adversarial attacks have been proposed to attack ML classifiers in the areas of computer vision. Finding adversarial examples, i.e., images that should be recognized as belonging to class A being instead misclassified as of class B, can be achieved by adding

adversarial perturbations such that the modified input images remain visually similar to their original versions, but produce erroneous classification results. Fast Gradient Sign Method (FGSM) [55] finds adversarial example by iteratively adding small perturbations in each pixel along the direction of the sign of gradients until the classification result becomes wrong:

$$x_{i+1} = x_i + \varepsilon * \text{sign}\left(\nabla_x J(\theta, x, y)\right)$$

[19] further improves adversarial attacks by formulating it into an optimization task

$$\text{minimize} \|\delta\|_p + c \cdot f(x+\delta) \tag{2.1}$$

$$\text{such that } x+\delta \in [0,1]^n, \tag{2.2}$$

in which δ is the added perturbation and $f(\cdot)$ represent the objective function to deceive the ML model. c is a constant and balance these two terms. [3] considers the ML classifier to be a black box and queries the classification scores of different inputs. The algorithm randomly adds perturbations to a small patch of the image, until an adversarial example is found.

2.2.4 Adversarial Attacks for Traffic Analysis

Conducting adversarial attacks in the networking domain is fundamentally different because of the existence of feature engineering. Finding adversarial perturbations in the feature space would not guarantee that a corresponding, legitimate network flow exist. [145, 93] experimented with Generative Adversarial Networks (GAN) which generate network flow features that are indistinguishable by ML classifiers, but their methods are less practical to use in the real world. On the other hand, iPET [146] and NIDSGAN [194] proposes GAN-based methods to generate perturbations on network traffic directly as an attempt for deceiving the ML classifiers that use raw traffic representation as inputs. Apart from adding perturbations to existing packets, Blind Adversarial Perturbation (BAP) [114] learns the optimal position in a flow where to insert dummy packets, disturbing directional features. The biggest limitation, however, is that these methods assume no manual feature extraction of network traffic in place. Apart from attacking ML-based models, Geneva [13] and SymTCP [175] design automated algorithms to discover the vulnerabilities of stateful DPI system implemented by censors.

2.3 Network-based Intrusion Detection System

NIDS, deployed by ISPs or in organizational networks, were originally proposed to capture network anomalies by matching pre-set firewall rules. Because of the heterogeneity and the constantly evolving characteristic of network attacks, those systems can always be circumvented, thus losing their efficacy [12]. In recent years, a tendency from designing NIDS for specific types of attack to the attempt to building a generic NIDS has transformed. In this section, we review popular systems or models designed for different types of intrusion detection.

2.3.1 Modeling Offensive Footprint Profiling

Modeling the unique malicious nature of network anomalies is effective for intrusion detection. BotHunter [60] builds infection dialogues to describe the dynamic process of Botnet infection, then employs modularized detection engines to identify the footprint of each stage in an attack. BotSniffer [61] identifies bot activity by highlighting the spatio-temporal correlations of Command and Control (C&C) traffic originating from pre-programmed behaviors. Profiling malicious code execution paths plays an important role in detecting malware [84, 112]. Likewise, stealth Distributed Denial of Service (DDoS) amplification can be fingerprinted by its unique two-stage behavior (i.e., scan and attack) [85]. These contributions demonstrate that modeling the potential links between different attack phases has merit in practice, and is served as the key methodology to build a defensive system. Conversely, these systems model the attack profiles of specific types of intrusion (Botnet, DDoS, etc.) and appear difficult to generalize to a broader spectrum, whereas data-driven approaches do not need to exploit modelling and have gained huge popularity.

2.3.2 What are we talking about when we are saying NIDS?

Before diving into ML/DL-based methods for NIDS, it is necessary to draw a clear border of the detection targets of NIDS. Ideally, a versatile NIDS should be able to detect every type of network attacks, including exploitation to zero-day vulnerabilities. However, zero-day attacks by nature are scarce and hard to spot in reality, which poses challenges to fuel a DL model. In fact, the majority of the security researchers have very limited access to real-world data, resulting in the situation that the overwhelming proportion of existing NIDS are evaluated on public-available datasets, among which

4 datasets are widely used:

1. *KDD-99* [80]: consists of around 5,000,000 connection records and 24 types of attacks. The connection records are described by 42 features including both system-level information (sudo attempt, etc.) and network-level information (duration, etc.). 24 types of attacks fall into 5 generic categories, namely, probes, denial of services, remote to user attacks and user to root attacks.

2. *NSL-KDD* [156]: Despite frequent usage of *KDD-99*, the dataset contains many deficiencies such as duplicate records, simulation artifacts and the fact that it does not reflect modern network traffic and attacks. *NSL-KDD* removes duplicate records and selects records that are difficult to classify, resulting in a condensed and cleaned version of the dataset. Technically speaking, *KDD-99* and *NSL-KDD* are not strictly designed for a pure NIDS given that parts of the features can only be acquired at the host level. The systems evaluated by these two datasets may be deployed on hosts which are capable of collecting such information, but would not be suitable to deploy at the sites such as routers or organizational gateways.

3. *IDS-2017* [144]: is collected by Canadian Institute of Cybersecurity in 2017 from a small, simulated network environment. The dataset contains around 3,000,000 network flows in raw packet capture format which is further transformed into a feature set with 81 categorical and numeric features. 12 attacks are collected, covering DoS, Cross-Site Scripting (XSS), brute-forcing, scanning and Botnet.

4. *IDS-2018* [25]: simulates an organizational network consisting of 5 subnets (departments). There are 420 client machines and 30 servers generating a range of benign traffic, including HTTPS, HTTP, SMTP, IMAP and SSH. 50 attack machines are set up to produce a collection of network attacks over 10 days. 17 types of attacks are generated in total, including DoS, DDoS, brute-forcing, scanning, XSS, SQL injection and Botnet. The size of *IDS-2018* is 5$\sim$6 times larger than *IDS-2017*.

It can be seen that the datasets above cover mostly high-profile and large-scale network attacks. Supervised ML models can possibly detect seen or similar attacks, but lacks the ability to discover zero-day attacks, which the application scenario although many supervised methods would not explicitly state.

2.3.3 Supervised Learning for IDS

From the perspective of ML, the most straightforward idea of solving network intrusion detection is to treat it as a classification problem that network traffic is either benign or malicious. A set of traditional machine learning techniques, including SVM [188], Multilayer Perceptron (MLP) [136], Random Forest [137], AdaBoost [70] and CNN [95] has been used as anomaly detectors. It is worth noting that ML-based classifiers may suffer from low detection rates to the minority classes due to imbalanced datasets [81]. Thus, the implementation of oversampling or undersampling would be necessary in that case to remedy the issue [20]. A unique method of tackling imbalanced datasets is also proposed in [18] which introduce the setting of an adversarial sampling agent which tries to sample specific types of attacks with the maximum probability of being misclassified. Otherwise, the sampling agent gets penalized during the training process.

Statistical methods are insensitive to imbalanced data, but usually suffer from the curse of dimensionality, due to the existence of high dimensional features in most of the network traffic datasets [40, 144, 66]. The Naïve Bayes classifier proposed by [110] adopts the fast-correlation based filter to reduce dimensions and K-Nearest Neighbours (KNN) classifier [96] uses KNN and K-means to preprocess data from n to 1 dimension before training. The drawback of these two-step approaches is that dimension reduction is isolated from the subsequent statistical analysis, possibly removing key information that could be used later [195].

Moreover, Recent research starts to study the effectiveness of exploiting sequential models, including RNN, LSTM and Gated Recurrent Unit (GRU), on Network Intrusion Detection (NID) [108, 32, 94, 189, 168]. Previous studies consider NID as a time-invariant task, i.e., network traffic packets or flows only independently occur in the cyber environment. This, however, is unlikely to hold given that the cyber adversaries always tend to initiate a series of attacking operations to compromise the vulnerable targets, meaning that potential time dependencies would exist. In this regard, sequential models may exhibit a higher accuracy by capturing the underlying temporal correlations in the network traffic. The problem is, aforementioned studies lack a specific threat model to highlight what exact temporal information can be learned, thus these algorithms are hardly supported by any practical knowledge. Plus, the training data are always randomly sampled from the dataset in these research, which would inevitably make the correlations in consecutive traffic flows invisible to the model.

2.3.4 Semi-supervised Learning for IDS

Detecting intrusions with semi-supervised learning has gradually gained popularity recently, since during training, this method does not require real attack traffic that is difficult to collect in the real world. Semi-supervised techniques are in line with the following methodology: Benign traffic would appear distinct from malicious attacks in feature spaces, hence modelling the statistical patterns of the former would be sufficient to tell whether a new sample is in the same group.

Reconstruction-based models learn to condense and reconstruct benign data via Autoencoders and its variants, and during testing, high reconstruction errors would indicate that the new samples are not seen before, and likely to be anomalous. N-BaIoT leverages separate deep Autoencoders to learn the features of different commercial IoT devices [106]. [102] proposes Conditional Variational Autoencoder (CVAE) to reconstruct benign traffic. Kitsune utilizes a lightweight ensemble of shallow Autoencoders to achieve online intrusion detection [107].

Besides, the underlying statistical patterns of the benign traffic can be modelled by probabilistic algorithms as well, such as Dirichlet Mixture Model (DMM) [111] and Gaussian Mixture Models (GMM) [195]. At the training stage, the model attempts to estimate the Probabilistic Density Function (PDF) of the benign samples, and then the likelihood of a new datum being sampled for the PDF would function as the main discriminative factor for inference.

Recent research also shows an interest in utilizing GAN to detect intrusions because of its capability to approximate complex high-dimensional distributions under an adversarial setting [54]. [190] adopts Bidirectional GAN to learn the distributions of benign data, and Encoding Adversarial Network (AnoEAN) is proposed specifically for anomaly detection by training an encoder (for benign traffic) under an adversarial environment [50].

In short, semi-supervised methods rely on the assumption that the network anomalies deviate far from benign data, so that a model that has captured the hidden patterns of benign samples in feature spaces, is capable of detecting malicious ones by comparing their feature discrepancies. However, it is worth noting that these semi-supervised algorithms are originally designed for the general anomaly detection issue rather than for network intrusions. The difference is that in the former, benign data are always assumed to have relatively similar characteristics, such as the same objects in images. When it comes to the network environment, benign traffic is originated from numerous

applications with distinct functionalities, including browsing, streaming, downloading, gaming, to name just a few. It is still uncertain whether these methods are capable of modelling 'benign' traffic with such a broad range. We experimented with popular semi-supervised models in Chapter 5 for network intrusion detection which in general are not on par with supervised benchmarks.

Chapter 3

Privacy Analysis of the Customized Android OS Communications

Mobile phones have become an indispensable part of modern life due to their portability and versatile functionality. There are currently more than 3 billion smartphones and tablets in use worldwide which run variants of the Android operating system (OS). While the well designed and encapsulated OS offers much convenience, the data transmission behind the scenes are almost opaque to users. The analysis of whether mobile apps disclose sensitive information to their associated back-end servers has been the focus of much research [77, 129, 126, 128, 173], especially with a view to risks such user de-anonymization, location tracking, behavior profiling, and cross-linking of data by different stakeholders in the device/software supply chain. In contrast, the disclosure of information at operating system level has received almost no attention and this aspect is not well understood. Mobile OS behavior has come to the fore only recently, with analyses of the Google-Apple Exposure Notification (GAEN) system that underpins Covid contract tracing apps [92] and following revelations of mass surveillance of journalists, politicians, and human rights activists though spyware exploiting zero-touch vulnerabilities (see the Pegasus project [45]). General Data Protection Regulation (GDPR) [41] in Europe stipulates that in the process of any data collection, (i) the data should be anonymized, i.e., cannot reasonably be linked to an individual and so is not personal data, (ii) with informed consent for a defined purpose and (iii) for the legitimate interests of the vendors. Similar regulations and laws have been implemented in various other countries. [58, 118]. The privacy analysis at the OS level, however, can be difficult since the wide use of TLS encryption along with the CA system not only prevents interception in the middle of the routes but also hinders people

other than phone vendors understanding what private information is transmitted. In this chapter, we offer the first in-depth measurement study of the data privacy practices of popular proprietary variants of Android OS in different regions of the world, and perform a cross-regional analysis between customized Android OS released in China and Europe, revealing the different attitudes towards sensitive data collection although some versions of the firmware are developed by the same mobile manufacturers.

3.1 Problem Statement

The transmission of user data from mobile handsets to back-end servers is not intrinsically a breach of privacy. For instance, it can be useful to share details of the device model/version and the locale/country of the device when checking for software updates. This poses few privacy risks if the data is common to many handsets and therefore cannot be easily linked back to a specific handset/person [155, 104].

Two major issues in handset privacy are (i) release of sensitive data, and (ii) handset deanonymization i.e., linking of the handset to a person's real world identity.

Release of sensitive data. What counts as sensitive data is a moving target, but it is becoming increasingly clear that data can be used in surprising ways and that so-called metadata can be sensitive data. One example of potentially sensitive metadata is the name, timing and duration of the app windows viewed by a user. This can be used to discover the time and duration of phone calls, when texts/messages are sent and received, when a prayer or dating app is used, and so on. More generally, such data reveals what apps a user spends most time viewing and which windows within the app they look at most. Another example is the list of apps installed on a handset. This can reveal user interests and traits [122, 143]. The list of apps can also act as a handset fingerprint, unique to only a small number of handsets, and so be used for tracking.

Data which is not sensitive in isolation can become sensitive when combined with other data, see for example [24, 30, 187]. This is not a hypothetical concern since large vendors including Google, Samsung, Huawei, and Xiaomi operate mobile payment services and supply custom web browsers with the handsets they commercialize.

The key requirement for privacy is that the data is common to many handsets. Risk factors therefore include whether data is tagged with identifiers that can be used to link different data together and to link it to a specific handset or person. Tagging data with the handset hardware serial number immediately links it to a single handset. Other long-lived device identifiers include the IMEI (the unique serial number of a SIM

slot in a handset) and the SIM IMSI (which uniquely identifies a SIM on the mobile network). To mitigate such risks, Google provides a Google Advertising ID that a user can reset to a new value. The idea is that data tagged with the new value cannot be linked to data tagged with the old value, and so resetting the identifier creates a break with the past. However, this is undermined if the new and old values can both be tied back to the same device and so linked together. It is worth noting that there already exist commercial services that given a Google Advertising ID offer to supply the name, address, email etc of the person using the handset [1].

Deanonymization. Android handsets can be directly tied to a person's identity in at least two ways, even when a user takes active steps to try to preserve their privacy. Firstly, via the SIM. When a person has a contract with a mobile operator then the SIM is tied to that contract and so to the person. In addition, several countries require presentation of photo ID to buy a SIM. Secondly, via the app store used. On Android handsets the Google Play store is the main way that people install apps. Use of the Google Play store requires login using a Google account, which links the handset to that account since Google collect device identifiers such as the hardware serial number and IMEI along with the account details [92, 89].

A handset can also become linked to a person's identity when data is collected that allows their identity to be inferred or guessed with high probability. One way that this might happen is via a handset's location time history. Many studies have shown that location data linked over time can be used to de-anonymize users, see e.g., [53, 153] and later studies. This is unsurprising since, for example, knowledge of the work and home locations of a user can be inferred from such location data (based on where the user mostly spends time during the day and evening), and when combined with other data this information can quickly become quite revealing [153]. It is worth noting that every time a handset connects with a back-end server, it necessarily reveals its IP address, which acts as a rough proxy for user location via existing geoIP services. With this in mind, the frequency with which connections are made becomes relevant, e.g., observing an IP address/proxy location once a day has much less potential to be revealing than observing one every few minutes.

Section 2.1.1 details a broad range of PII while in this work we consider the following four major categories:

1. **Device-specific:** Information that is bundled with the device upon manufactur-

[1] `https://www.vice.com/en/article/epnmvz/industry-unmasks-at-scale-maid-to-pii`, accessed 18th Aug 2021.

ing or setup, such as International Mobile Equipment Identity (IMEI), Mobile Equipment Identifier (MEID), Android ID, MAC address, hardware serial number, installed packages and hardware model.

2. **Location-specific:** Information that directly or indirectly reveals the location of the user, such as GPS coordinates, the SSID and MAC address of nearby Wi-Fi access points, Mobile Country Code (MCC), Mobile Network Code (MNC), Location Area Code (LAC), and cell ID (CID).

3. **User Profile:** Information that reveals user identity (phone number) and user traits (list of installed apps, app usage logs).

4. **Social Relationships:** Information that contains details about the personal contacts of the user, such as phone call and SMS history.

Since this work targets the privacy practices at the OS level, we confine consideration to the information transmitted by the OS and system apps that come preinstalled on a handset, leaving out any third-party apps that a user would install themselves. We note that system apps such as Settings and Messages cannot be removed by the user, and preinstalled system apps can have access privileges that user-installed apps may not gain easily, especially not without explicit user consent. We also care about the Android OS privacy practices in different regions in the worlds, among which two regions are the most representative, Europe and China. EU issued the toughest data protection law – GDPR in the world. Any mobile devices in the EU should be obligated to the law, and thereby follows the highest privacy standards. China, on the other hand, represents the biggest market in the world using Android with a relatively isolated Android OS environment (Google services are forbidden). The ecosystem of Android appears drastically different and privacy protection of the local OS remain unknown.

3.2 Methodology and Data Collection

We purchased the following handsets in Europe which are all shipped with Global/European firmware:

1. Samsung Galaxy S9 (model SM-G960F)/Android 10 (build QP1A.190711.020, One UI v2.0);

2. Xiaomi Redmi Note 9 (model M2003J15SG)/Android 10 (build QP1A.190711.020, MIUI Global 12.0.7 QJOMIXM);

3. Realme 6 Pro (model RMX2063)/Android 10 (build `RMX2063_11_A.38`, realme UI v1.0);

4. Huawei P10 Lite (model MAR-LX1B)/Android 9[2] (build 9.1.0.372, EMUI 9.1.0);

5. Google Pixel 2/Android 10 (LineageOS build 17.1-20210316, opengapps 10.0-nano-20210314);

6. Google Pixel 2/Android 10 (eos build e-0.11-q-20200917).

In this chapter, we may use global and European firmware interchangeably since phone vendors have different conventions for naming. To conduct cross-regional analysis, we select China as the second target region due to the massive number of the market share. We purchased the following handsets with Chinese firmware in mainland China:

1. Xiaomi *Redmi Note 11* with Android 11/MIUI 12.5.4.0 RGBCNXM;

2. OPPO *Realme Q3 Pro* with Android 11/realme UI v2.0 RMX2205_11_A.13 (based on ColorOS 11);

3. *OnePlus 9R* with Android 11/ColorOS 11.2 LE2100_11_A.05

Regional differences take the form of differences in the installed firmware, e.g., differences between the system apps installed in the Global/EU vs. CN Android OS distributions. In addition, even when the same version of system app is installed in the CN and Global distributions, the app may contain logic that causes it to behave differently depending on the region, e.g., by checking the current location. Regarding the latter, by reverse-engineering the main system apps, we find that mobile devices tend to use locale, firmware version, IP address and MCC (which identifies the mobile operator country) to localize their behavior. For example, Xiaomi maintains a list of URLs in a Java HashMap:

```
1 {"CN": "data.mistat.xiaomi.com",
2 "INTL": "data.mistat.intl.xiaomi.com",
3 "IN": "data.mistat.india.xiaomi.com"},
```

[2]Following US trade sanctions against Huawei, Android 9 is the latest version of Android available on a Huawei handset that we could root.

and uses a hardcoded value ("CN") and locale together, to select an endpoint in the Chinese firmware. The preinstalled Amap package on Realme and OnePlus handsets uses the MCC to determine the API endpoint:

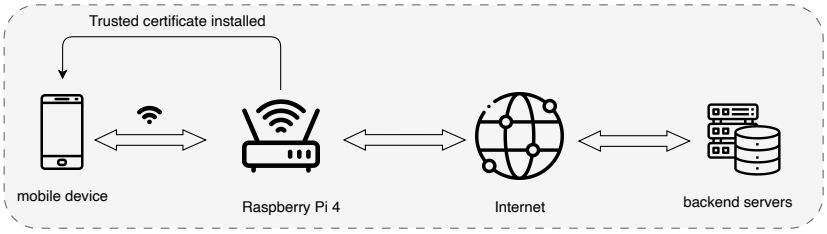
```
StringBuilder sb = new StringBuilder();
if (a2) {
    sb.append("http://aps.oversea.amap.com/APS/r?ver=5.1&q=0");
} else if (z) {
    sb.append("http://aps.testing.amap.com/APS/r?ver=5.1&q=0");
} else if (z2) {
    sb.append("https://aps.amap.com/APS/r?ver=5.1&q=0");
} else {
    sb.append("http://aps.amap.com/APS/r?ver=5.1&q=0");}
```

in which `a2` is `True` if MCC does not equal 460 (China).

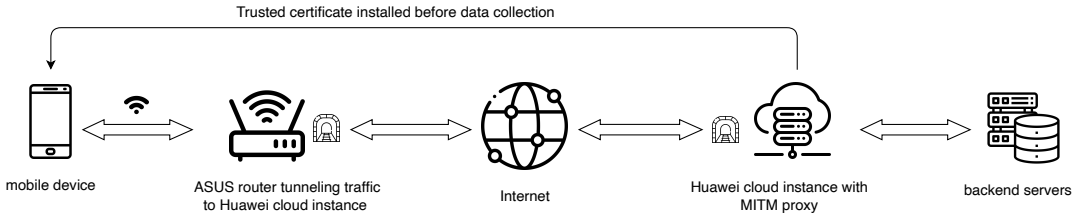
3.2.1 Environment Setup

Almost all the data we observe is sent over HTTPS connections and so encrypted using TLS/SSL (in addition to any other encryption used by the app). However, decrypting SSL connections is relatively straightforward. We route handset traffic via a Wi-Fi access point (AP) that we control, configure this AP to use `mitmdump` as a proxy [27] and adjust the firewall settings to redirect all Wi-Fi HTTP/HTTPS traffic to mitmdump so that the proxying is transparent to the handset. When a process running on the handset starts a new network connection, the mitmdump proxy pretends to be the destination server and presents a fake certificate for the target server. This allows mitmdump to decrypt the traffic. It then creates an onward connection to the actual target server and acts as an intermediary, relaying requests and their replies between the app and the target server while logging the traffic. The setup is illustrated schematically in Figure 3.1 (upper).

For Chinese handsets, we do not conduct experiments locally and recognize that the data collected outside China may not fully represent a device's behavior in that region. To overcome this issue, we set up a network tunnel between our campus and a Huawei Cloud instance in Shanghai as shown in Figure 3.1 (lower). The IP address observed by the backend server is thus that of the Huawei Cloud server located in Shanghai. Specifically, we first configured a Wi-Fi access point on an ASUS RT-AC86U router which supports third-party firmware and thus allows for the configuration of many VPN or tunneling protocols. We build a tunnel where the proxy server is based on a Huawei Cloud s6.small instance running Ubuntu 20.04 in Shanghai, and the ASUS router running Koolshare 3.0.0.4 works as a proxy client, which is configured to redirect any



(a) Measurement Setup for EU/Global mobile handsets, which are configured to access the Internet using a Wi-Fi access point hosted on a Raspberry Pi.



(b) Measurement Setup for CN mobile handsets. ASUS Wi-Fi router tunnels TCP/UDP traffic to a Huawei Cloud Instance in Shanghai.

Figure 3.1: Measurement setup. A system certificate is installed on the phone to be able to decrypt outgoing traffic. The MITMProxy pretends to any process running on a Raspberry Pi/Cloud instance to be the destination server, creates a connection to the actual target, and relays requests and their replies between handset and server while logging the traffic.

TCP/UDP packets to the endpoint on the Huawei Cloud instance (see Figure 3.1) To avoid any negative impact on the censorship circumvention community, we do not disclose the tunneling protocol used. During our traffic collection campaign, only the handsets being studied are permitted to connect to the Wi-Fi access point, in order to prevent other sources of traffic from interfering with our measurements. We set up each handset using Chinese as the language to simulate a local user. With this setup, the only app that we observed still adapting to the device's true location was the Amap app. Fortunately, through reverse-engineering (decrypting/decoding every connection in the collected traces via mitmproxy and apk decompiling and analysis, which allows us to examine all the transmitted fields in plaintext) we confirmed that the contents of the messages transmitted to Amap's different API endpoints are the same in all regions, and so leaves our traffic collection unaffected. We did not find any system services or preinstalled applications that use GPS to select an endpoint.

3.2.2 Device Setup

Efforts are also needed on the handset side to decrypt HTTPS connections. System processes typically carry out checks on the authenticity of server certificates received when starting a new connection and abort the connection when these checks fail. Installing the mitmproxy CA cert as a trusted certificate causes these checks to pass, except on the Huawei handset. Installing a trusted cert is slightly complicated in Android 10 and above, since the system disk partition, on which trusted certs are stored, is read-only and security measures prevent it being mounted as read-write. Fortunately, folders within the system disk partition can be overridden by creating a new mount point corresponding to the folder, and in this way the mitmdump CA cert can be added to the `/system/etc/security/cacerts` folder. On the Huawei handset each system app contains embedded server certificate SHA256 hashed and when starting an HTTPS connection checks that the certificate offered by the server matches one of these hashes. It is thus necessary to bypass these checks on each app individually (installing a system-wide trusted cert is not enough), for which We used Riru/edXposed.

At the start of each test we removed any SIM card and carried out a hard factory reset of the handset, i.e., we used `TWRP` to manually wipe the data partition, thereby forcibly removing all user data and settings, all user installed apps and resetting any disk encryption. Note that we observed that simply clicking on the "factory reset" option in the UI sometimes did not fully remove user data and settings.

Following this factory reset, the handset reboots to a welcome screen and the user is then typically asked to agree to terms and conditions, and presented with a number of option screens. We note that all the option toggle switches default to the opt-in choice, and so it is necessary for the user to actively select to opt-out. To mimic a privacy conscious user, we unchecked any of the options that asked to share data and only agreed to mandatory terms and conditions.

1. *Samsung*: we unchecked the Sending of Diagnostic Data, Information Linking, Receipt of Marketing Information components of the terms and conditions, skipped the Protect Your Phone screen, did not sign into a Samsung account (which raises a warning that it disables Samsung Cloud, Bixby, Galaxy Themes, Find My Mobile, Samsung Pass, Galaxy Store, Secure Folder).

2. *Xiaomi*: we unchecked the Location, Send Diagnostic Data Automatically, Automatic System Updates, Personalized Ads, User Experience Program options.

3. *Realme*: we unchecked the User Experience Program and Uploading Device Error Log Data components of the terms of service, unchecked the Wi-Fi Assistant and Auto-update Overnight options.

4. *OnePlus*: we unchecked the 'User Experience Program', 'Automatically Select the Best Wi-Fi', 'Automatically Switch to Mobile Network', 'Auto Update Overnight' and skip Learn Swipe Gestures.

5. *Huawei*: we selected No Thanks on the Enhanced Services screen, Later on the User Experience Improvement Program screen, Update Manually on the Keep Your Software Up-To-Date screen.

6. *LineageOS*: we unchecked the Help Improve LineageOS, Location Services options.

7. *e/os*: we unchecked the Location Services option, skipped Fingerprint Setup, Protect Your Phone and /e/ account setup. All the mobile OSes, apart from e/os, also displayed a Google services screen on first startup. On this we unchecked the Use Location, Allow Scanning, Send Usage and Diagnostic Data options, and we did not log in to a Google user account.

During this startup process, we left Wi-Fi disabled and since no SIM was inserted, there was also no cellular data connection. This allowed us to install the mitmproxy CA cert, and on the Huawei handset Riru/edXposed modules to disable HTTPS cert checks by individual system apps, before the handset made any network connections. Wi-Fi access was then enabled after these steps were completed.

3.2.3 Data Collection

We seek to define simple experiments that can be applied uniformly to the handsets studied (so allowing direct comparisons) and that generate reproducible behavior. Mobile OS developers commonly provide add-on services that can be used in conjunction with their handsets, e.g., Samsung offer Cloud storage, Bixby, the Samsung Store; Huawei offer Cloud storage, the AppGallery store; Xiaomi offer Xiaomi Cloud, Mi Coin and Credit. Here we try to keep these two aspects separate and to focus on the handset as a device in itself, separate from optional services such as these. We also assume a privacy-conscious but busy/non-technical user, who when asked, does not

select options that share data but otherwise leaves handset settings at their default values.

On Android the Settings app must be used to e.g., enable location and Wi-Fi. Since use of the Settings app is not optional for handset users, we include them in our tests. In addition, while on Android apps may be sideloaded over `adb`, all of the handsets except for those with Chinese firmware include the Google Play Store and for most users this is the primary way to install apps. Other than on e/os, use of the Google Play Store requires the user to sign in to a Google account and so disclose their email address and perhaps other personal details. We therefore also include opening of the handset Google Play Store app and login to a Google account in our tests.

With these considerations in mind, for each handset we carry out the following experiments:

1. Start the handset following a factory reset (mimicking a user receiving a new phone), recording the network activity.
2. Connect to the controlled Wi-Fi access point;
3. Leave the handset untouched for 24 hours and log the network activity via *mitmproxy* on Huawei Cloud;
4. Insert a SIM card, disable Mobile Data and log the network activity through Wi-Fi. The phone was connected to a UK cellular network at this stage;
5. Keep the handset idle for 24 hours with a SIM card inserted and record the network activity;
6. Turn off and on location and record the network activity;
7. Make and receive a phone call, and log the network activity;
8. Send and receive a message, and log the network activity;
9. Open Camera, Clock, Note, Photos, Recorder, Settings one-by-one, and log the network activity. If any application asks for permissions, we only grant the minimal set that ensures they remain functional. The following permissions are granted:
   (a) `android.permission.CAMERA`: is granted to the Camera app on all handsets;

(b) `android.permission.RECORD_AUDIO`: is granted to the Recorder app on all handsets;

It should be noted that permissions including `READ_CALL_LOG`, `CALL_PHONE`, `ANSWER_PHONE_CALLS`, `READ_PHONE_NUMBERS`, `READ_CONTACTS`, and `WRITE_CONTACTS` are granted to Dialer and permissions including `SEND_SMS`, `RECEIVE_SMS`, `READ_SMS`, `READ_PHONE_NUMBERS`, `READ_CONTACTS` and `WRITE_CONTACTS` are granted to Message apps without the need for user interactions.

10. (**Only for non-Chinese phones**) Open the pre-installed Google Play app and log in to a user account, recording the network activity. Then log out and close the app store app;

11. Remove the SIM card and log the network activity;

12. Factory Reset the device and log the network activity.

We highlight a number of potential limitations of the traces collected in our study.

1. For Chinese handsets, we do not alter the GPS location of the handsets and we do not know whether being inside mainland China would trigger the AMap maps app to behave differently. This is limited to the Realme and OnePlus handsets, which have the AMap app preinstalled and running in the background;

2. There is a one-year gap between the data collection for European-purchased devices and that for Chinese-purchased devices. Due to the passage of time, the Chinese devices in our investigation are shipped with Android 11, whereas the devices purchased in Europe run Android 10. However, we find that the differences that we observe between the CN and Global variants of the Android OS are not due to the Android version, but instead are mainly associated with the preinstalled applications and system services.

3. We only collect and analyze the network traffic generated by system apps and by basic applications such as the Dialer and Message apps. Nevertheless, we argue that these largely reflect the attitude of vendors towards user privacy.

4. Traffic analysis is limited to the apps which do not implement certificate pinning.

3.2.4 Key Connection Data

Key network connections that transmit private information are detailed in Appendix A and B. In the remaining sections of this chapter, we summarize the PII collection into the major categories defined in Section 3.1, while interested readers can refer to the Appendix which documents all the destination endpoints that receive private data and the specific type of data being transmitted.

3.3 Privacy Analysis

3.3.1 Code Analysis and Dynamic Hooking

The traffic collected still cannot be directly used for privacy analysis because *(i)* packet payloads are almost always encrypted – not just due to the widespread use of HTTPS to transfer data but, as we will see, also because the message data is often further encrypted by the sender using a cipher that may not be explicitly specified through meta-data, particularly when the data may be sensitive (end-to-end encryption); *(ii)* prior to message encryption, data is often encoded in a binary format for which there is little or no public documentation; and *(iii)* for proper attribution, we need to be able to link a message to the sending process/app on the handset.

Some HTTP requests would embed the name of the package in headers or queries, which allows us to pinpoint the source APK files and figure out the applied encryption algorithm via decompiling. However, it is also common to see HTTP requests with only a minimal set of headers and everything else encrypted. In cases where this type of requests occur, such as factory reset or opening an app, we log them and configure *mitmproxy* to delay these requests for 20 seconds. We then trigger the handset to generate those requests again. Once they are observed and delayed on the proxy server side, we run an exhaustive search in all the `/proc/<pid>/net/tcp6` files, which record the socket usage of each process, including source and destination IP addresses, and the UID of the connection-initiating package. Upon determining the target android packages, we retrieved the Android application packages (APKs) of the apps on the `/system` disk partition and decompiled them. While the bytecode of Android Java apps can be readily decompiled, the code is almost always deliberately obfuscated in order to deter reverse engineering. As a result, reverse engineering the encryption and binary encoding in an app can feel a little like exploring a darkened maze. Perhaps unsurprisingly, this is frequently a time-consuming process, even for experienced re-

searchers/practitioners.

After repeating the procedure on all the HTTPS requests with unreadable/encrypted contents, we uncover a few data encryption routines that the app developers tend to use in practice, as summarized below.

Symmetric Encryption. We identify a large number of packages that apply AES encryption algorithms when uploading PII, by decompiling the associated APK files. For example, `com.coloros. weather` on Realme and OnePlus encrypts GPS coordinates and PII by AES (CTR mode), and concatenates the key and the initialization vector at the end of the ciphertext. `com.nearme.romupdate` also appends AES keys at the end when querying system update information. On the other hand, `com.ted.number`, China Unicom SDK, `com.nearme.instant.platform` and `com.android. updater` (Xiaomi) hardcode AES keys in the package, which are used to encrypt POST contents of a range of HTTP requests. For these packages using symmetric encryption algorithms, as long as we acquire the AES keys, the contents can be easily decrypted.

However, an exception is `com.amap.android.location` which encapsulates encryption/decryption algorithm in a precompiled JNI library. Instead of getting the keys, we have to hook the plaintext before encryption during runtime, for which we utilize Frida and Riru/EdXposed. Frida [47] is a dynamic code instrumentation tool that has full access to the memory and injects Javascript code into the target process on Android, allowing users to hook variables and function calls during runtime. The flexibility of JS API makes it possible to hijack a running program with minimal coding and configurations. However, it is not possible to spawn an Android process that has no foreground activity, meaning that hooking the variables created right after a process is started can be difficult. An alternative is using Riru/EdXposed. Riru [132] injects code into Zygote which is a special process handling the forking of every new Android app, and [1] is an Riru module that provides universal hooking APIs. This framework is able to inject code into the app at the very beginning when it is forked from Zygote, thereby providing better control of code instrumentation than Frida. The drawback, on the other hand, is that any changes of the hooking code requires a reboot to be effective. In this work we use Frida 15.2.2, Riru 25.4.4 and EdXposed 0.5.2.2 collectively.

Asymmetric Encryption. Asymmetric encryption is involved in a small number of packages, such as `com.mobiletools.system helper` (China Unicom SDK), when uploading private information. Specifically, the package contains a hardcoded RSA public key, and each HTTP request contains the ciphertext encrypted with an AES key and also the RSA-encrypted AES key. Since the RSA private key is stored on the back-

end server, we have to utilize Frida to hook runtime plaintext before it gets encrypted. A few Xiaomi packages adopt similar routines while uploading app telemetry.

3.3.2 Traffic Analysis - Global Firmware

Table 3.1 gives an overview of the data collection observed on the handsets studied. It is helpful to consider this in light of four basic questions: (i) who is collecting data, (ii) what sort of data is being collected, (iii) can resettable identifiers be relinked to the device, (iv) what is the potential for cross-linking of data collected by different parties.

Q1: Who is Collecting Data?

Mobile OS Developers We observe that Samsung, Xiaomi, Realme and Huawei all collect data from user handsets, despite the user having opted out of data collection/telemetry/analytics and making no use of services offered by these companies. This data is tagged with long-lived identifiers that tie it to the physical device, including across factory resets.

In contrast, LineageOS and e/os were not observed to collect handset data. The latter is notable because a case might be made for the necessity of mobile OS operators collecting handset data in order to monitor software operation and catch problems early (i.e., devops). However, it is hard to justify the necessity of such data collection, i.e., that users should have no opt-out, when two mobile OSes adopt an opt-in approach. It is also worth noting that it can be hard to distinguish between diagnostics for existing software and beta testing for new or updated software/features. Traditionally, beta testing has always been opt-in.

Pre-installed Third-Party System Apps. System apps are pre-installed on the `/system` partition of the handset disk. Since this partition is read-only, these apps cannot be removed. They are also privileged in the sense that they can be assigned permissions without needing user consent, be silently started, etc. The Settings app is, for example, a system app. All the mobile OSes studied, apart from e/os, have pre-installed Google system apps. We discuss these further below, but first we consider pre-installed system apps from other companies.

The Samsung handset studied also contains pre-installed system apps from Microsoft that send handset telemetry data to `mobile.pipe.aria.microsoft.com`, `app.adjust.com` (a third-party analytics company[3]) and use Firebase push mes-

[3]Their website says “Adjust offers a number of analytics tools designed to give you the deepest

Table 3.1: Summary of data collection by each Android OS variant.

	Samsung	**Xiaomi**	**Realme**	**Huawei**	**LineageOS**	**e/os**
persistent Device Identifiers	IMEIs, hardware serial numbers	IMEIs, Secure DeviceID, MD5 hash of Wi-Fi MAC address	IMEI, deviceID, guid	hardware serial number, device RSA cert	-	-
Resettable Identifiers Relinkable to Device	Samsung Consumer ID, Firebase IDs	VAID, Google Ad ID	VAID, OAID, device_id, registrationId, Google Ad ID, Firebase IDs	-	-	-
Third-Party System App Data Collectors	Google, Mobile Operator, Microsoft, LinkedIn, Hiya	Google, Mobile Operator, Facebook	Google, Heytap	Google, Daily Motion, Avast, Qihoo 360, Microsoft	Google	-
Main Telemetry Collectors (By Data Volume)	Google, Samsung, Microsoft	Google, Xiaomi	Google, Heytap	Google, Microsoft	Google	-
Loggers of App Usage Over Time	Samsung	Google, Xiaomi	-	Google, Microsoft	-	-
Loggers of Apps Installed On Handset	Google, Samsung	Google, Xiaomi	Google, Realme, Heytap	Google, Huawei	Google	-

saging. A LinkedIn (now owned by Microsoft) system app also sends telemetry to `www.linkedin.com/li/track`. This third-party data collection occurs despite no Microsoft/LinkedIn apps were ever opened on the device, and no popup or request to send data was observed.

The Samsung and Xiaomi handsets studied also contain pre-installed system apps from mobile operators (SFR/Altice in France, Deutsch Telekom in Germany), which were observed to send telemetry. Note that our handsets were bought second-hand on the Internet and a more controlled study of operator installed system apps may well be warranted. As well as sending telemetry directly, the SFR/Altice app on the Samsung handset also uses Google Analytics to log events.

The Realme handset studied contains pre-installed system apps from Heytap, a Singapore-based private company. It appears that Realme partners with Heytap, who provide account management, cloud data, an app store, etc.

Huawei also appears to partner with a number of third parties to provide handset system services. The Huawei handset studied contains a pre-installed `com.huawei.systemmanager` app which has embedded within it components from third-party scanning/antivirus services Avast (based in the Czech Republic) and Qihoo 360 (based in China). App data is sent to avast.com when an app is installed on the handset. Periodic connections are also observed to `360safe.com` (associated with Qihoo 360) that send device data. The `com.huawei.himovie.overseas` system app sends handset data to servers associated with Dailymotion, even though no video app was ever opened on the handset (perhaps these connections prefetch news/topical videos). The Microsoft Swiftkey keyboard app `com.touchtype.swiftkey` is pre-installed on the Huawei handset and sends crash data to `in.appcenter.ms/logs` and telemetry data to `telemetry.api.swiftkey.com`.

In addition to mobile operator system app sharing data on the Xiaomi handset, a pre-installed Facebook app collects data.

Apart from Google's GApps, no third-party system apps on the LineageOS handset were observed to perform data collection. On e/os, we observed no data collection by system apps.

Google System Apps (GApps). The Samsung, Xiaomi, Realme and Huawei handsets studied all have pre-installed Google system apps, the so-called GApps package. These include Google Play Services,[4] Google Play Store, YouTube, Gmail, Maps,

insight into your user interaction, your marketing channels, and your campaign performance".

[4]Google Play Services provides the API for Google Firebase services such as Google Analytics and

Drive, Wallet, Chrome. On LineageOS it is necessary to install GApps to use the Google Play Store, but this is not necessary with e/os (which uses the open-source MicroG re-implementation of Google Play Services and the Google Play app). It is known that Google Play Services and the Google Play Store send large volumes of handset data to Google and collect long-lived device identifiers, although until recently there has been a notable lack of measurement studies (see [92, 89]). Other Google apps such as YouTube and Gmail also send handset data and telemetry to Google.

It is worth noting that the volume of data uploaded by Google is considerably larger than the volume of data uploaded to other parties. For example, Figure 3.2 shows the average rate at which data is uploaded from each handset when lying idle, broken down by data source. The volume of data sent to Google is broken out into a separate plot to make it easier to see the volumes sent to other companies.

It can be seen that no data is uploaded to the LineageOS or e/os developers. On the Realme handset Heytap uploads around 3-4× more data than Samsung, Xiaomi and Huawei. Realme themselves collect far fewer data than Heytap, about half of that collected by Samsung, Xiaomi and Huawei. On the Samsung handset the Microsoft system app uploads a similar volume of data as Samsung.

The volume of data uploaded by Google varies across the handsets. It is zero for e/os, since it uses the MicroG open source re-implementation of Google GApps. LineageOS and Samsung send similar volumes of data, Xiaomi and Huawei about twice as much and Realme about three times as much. These differences are likely related to different configurations of Google GApps e.g. on LineageOS the so-called nano version of GApps was installed (other options includes micro, mini, full, stock[5]). In all cases the volume of data uploaded to Google is at least 10× that uploaded by the mobile OS developer. For Xiaomi, Huawei and Realme the volume rises to around 30×. Recall that this is despite the "usage & diagnostics" option being disabled for Google services on all handsets (and also the diagnostics/analytics options also being disabled for the mobile OS developers, see Section 3.2.2).

Note however that from a privacy viewpoint it is not the volume of data that is primarily of concern, but rather the contents of that data and the frequency with which it is sent.

Crashlytics to apps on the handset, but also performs device logging/telemetry on behalf of Google.

[5] See `https://github.com/opengapps/opengapps/wiki/Package-Comparison`

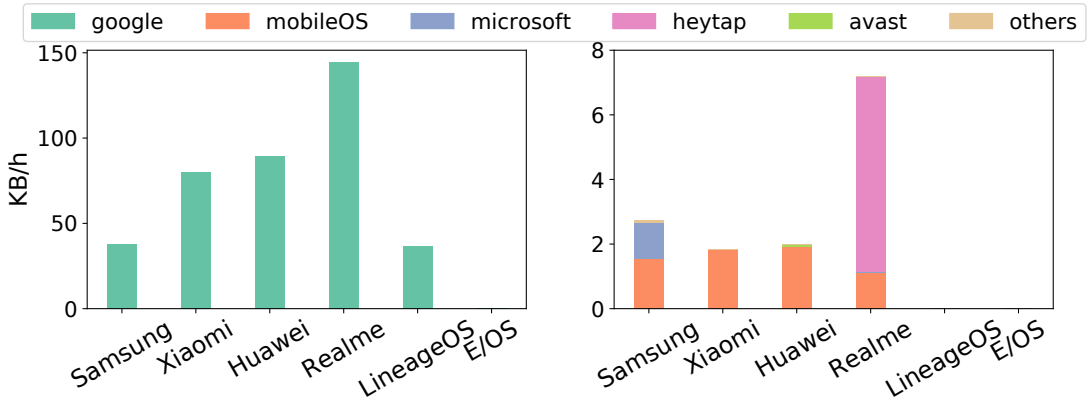


Figure 3.2: The average volume of the network traffic generated on each handset by each data collector.

Key findings: *Mobile vendors, third-party pre-installed apps and Google services are the main players in private data collection from mobile devices, in which Google services collect much more frequently than other parties.*

Q2: What Sort Of Data Is Being Collected?

The data that we observe being sent from handsets can be roughly categorized as: (i) device/user identifiers, (ii) device configuration data and (iii) event logging data/telemetry.

Device/User Identifiers We observe that most of the connections from a handset are tagged with an identifier of some sort. Single-use identifiers can be used to avoid duplicate messages being received and session identifiers can be used to link together groups of connections, e.g., when accessing authenticated resources. These types of identifier are ephemeral, i.e., they are short-lived, hard to link to a particular device and so carry little privacy risk. Longer-lived identifiers can also be used, e.g., to maintain state, and so long as the same identifier is shared by many devices, this carries little privacy risk. Google's Safe Browsing service is a good example of such an approach [91].

Unfortunately, we observe little use of such privacy-friendly identifiers in our handset measurements. Instead, we find that sending persistent identifiers in connections is ubiquitous. Table 3.1 lists the main identifiers sent in connections on each handset. Some of these identifiers are long-lived, e.g., the IMEI (which is typically engraved on the SIM slot), hardware serial number and, on Huawei handsets, the device RSA cert [71]. These identifiers persist across factory resets of the device and are effec-

tively permanent and indelible. Others, such as the Google Advertising Id and VAID, are user-resettable either manually or by a factory reset of the phone. But in practice that means they rarely change and act as strong device identifiers. Further, as we discuss in more detail below, most of these resettable identifiers can be relinked back to the device since long-lived identifiers are sent alongside them.

This means that connections from the same handset can generally be easily linked together over time, which has several consequences. One is that data on device and user behavior is linked over time, with obvious privacy implications. Another is that every time a handset connects with a back-end server it necessarily reveals the handset IP address, which acts as a rough proxy for user location via existing geoIP services. Many studies have shown that location data linked over time can be used to deanonymize, e.g., see [53, 153] and later studies. This is unsurprising since, for example, knowledge of the work and home locations of a user can be inferred from such location data (based on where the user mostly spends time during the day and evening), and when combined with other data this information can quickly become quite revealing [153].

Device Configuration Data. Sharing device hardware/system configuration data such as the device model, screen size, operating system version, radio version generally carries little privacy risk since these are common to many devices (e.g. all devices of the same model). Such data is needed when checking for software updates and selecting the right version of an app to install. Samsung, Xiaomi, Realme and Huawei all collect device configuration data, as do Google and many third-party system apps.

Table 3.2 summarizes the collection of device configuration data observed in our measurements. It can be seen that, as expected, all the devices transmit this information when checking for firmware and app updates. This data is also transmitted by one or more services of every device (although as noted above in our experiment setup the user opts out of services when asked). All the devices transmit device configuration data to their advertising endpoints, while Xiaomi and Realme also transmit this data in their analytics connections.

Brand	Firmware Updates	App Updates	Services	Advertising	Analytics
Samsung	✓	✓	✓	✓	
Xiaomi	✓	✓	✓	✓	✓
Huawei	✓		✓	✓	
Realme	✓	✓	✓	✓	✓

Table 3.2: Connection categories sending device hardware/system configuration data

List of Installed Apps. This list of apps installed on a handset is potentially sen-

sitive information since it can reveal traits of the handset user. For example, a Muslim prayer app, a period tracking app, a gay dating app, a mental health app can, respectively, reveal the user's religion, gender, sexual preference and mental health status. The newspaper apps installed can reveal political preferences. The set of apps installed is also more likely to be unique to one handset, or a small number of handsets, and so can act as a device fingerprint (especially when combined with device hardware/system configuration data). See, for example, [122, 143, 46] for recent analyses of such privacy risks. We note that in light of such concerns, Google recently categorized the list of apps installed as "personal and sensitive user data" and introduced restrictions on Play Store apps collection of this data [56], but such restrictions do not apply to system apps since these are not installed via the Google Play Store.Table 3.4 summarizes the collection of the list of installed apps observed in our measurements. It can be seen that, with the exception of Huawei, all of the devices transmit this data. Xiaomi and Realme transmit this data to app update endpoints and in both cases only resettable identifiers are sent in these connections, which is reasonable practice although it would enhance privacy to use less persistent identifiers e.g., session-based identifiers. Samsung and Realme transmit the list of installed apps to endpoints associated with vendor services. Xiaomi transmits the list of installed apps to an analytics endpoint, along with persistent identifiers, a poor privacy practice.

Brand	App Updates	Services	Analytics
Samsung		✓	
Xiaomi	✓		✓
Huawei		✓	
Realme	✓		

Table 3.3: Connection categories sending list of installed apps.

Brand	Preinstalled Apps	G-services	Analytics
Samsung		✓	
Xiaomi			✓
Huawei	✓		

Table 3.4: Connection categories sending list of installed apps.

Event Logging Data/Telemetry. Samsung and Xiaomi both log data that can reveal user interactions occurring on a handset. Third-party system apps by Google and Microsoft also carry our event logging that can reveal user interactions. Heytap, Daily Motion and the mobile operator log events related to operation of their specific app.

Some logging of events is probably reasonable, e.g., to allow early detection of app performance issues (excessive battery drain, slow operation, etc.). But ongoing detailed logging of the activity on a handset, particularly user activity, can quickly become intrusive and a serious privacy concern. The last row of Table 3.1 lists the companies carrying out ongoing and frequent telemetry/event logging on each handset.

Notr that this occurs despite the user opting out of diagnostics/analytics collection on the handsets during onboarding following factory reset.

Xiaomi collects extensive event logging data/telemetry. This is mainly sent to `tracking.intl.miui.com`. The data sent is doubly-encrypted, i.e., the data is first AES encrypted and then sent over an encrypted HTTPS connection. After quite some work reverse engineering the AES key management scheme used, we managed to decrypt the data. The data consists of both timestamped individual events and timestamped sequences of events grouped together. The events logged include, for example, every opening and closing of an app window ("activities" in Android parlance) plus the duration a window is open. Since all window events appear to be logged, this can easily reveal detailed information on user handset usage. For example, Figure 3.3 shows decrypted logging data sent to Xiaomi when a phone call is received. The dialer app opens its InCallActivity window when the call arrives and closes it when the call ends. Timestamps of the open and close events, plus the duration, are sent to `tracking.intl.miui.com`. Xiaomi's system apps `com.miui.msa.global`, `com.xiaomi.discover`, `com.android.thememanager` also log events using Google Analytics.

Microsoft's Swiftkey keyboard (used on the Huawei handset) also carries out extensive event logging, sending this data to `telemetry.api.swiftkey.com`. In particular, when the keyboard is used within an app then the app name, number of characters entered and an event timestamp are sent. In this way use, for example, of the searchbar, contacts and messaging apps is logged and so can easily reveal detailed information on user handset usage. See, for example, Figure 3.4. Interactions with the keyboard, e.g., opening the clipboard, viewing/modifying the settings, are also logged. Information on Swiftkey app crashes, including stack traces, is sent to `in.appcenter.ms`.

Several Samsung system apps use Google Analytics to log user interaction events, including windows/activities viewed plus duration and timestamp. System apps instrumented in this way include `com.wssyncmldm`, `com.samsung.android.samsungpass`, `com.samsung.android.authfw`, `com.samsung.android.bixby.agent`, `com.samsung.android.game.gamehome`, `com.sec.android.app.samsungapps`.

```
POST https://tracking.intl.miui.com/track/v4
Headers
   OT_SID: 1904b90...536c63d4
   OT_ts: 1627029461128
   OT_net: WIFI
   OT_sender: com.miui.analytics
      "seq": [
        {
          "event": 1,
          "pkg": "com.google.android.dialer",
          "class": "com.android.incallui.InCallActivity",
          "ts": 1627028918422,
          "vn": "67.0.383690429",
          "stat": "app_start"
        },
        {
          "event": 2,
          "pkg": "com.google.android.dialer",
          "class": "com.android.incallui.InCallActivity",
          "ts": 1627028934973,
          "vn": "67.0.383690429",
          "duration": 16551,
          "stat": "app_end",
          "app_duration": 16551
        }
```

Figure 3.3: Xiaomi's telemetry logs the user interaction with the dialer app when receiving a phone call, including the start and end times of the call.

The app `api.omc.samsungdm.com` logs when a SIM is inserted and `samsung-directory.edge.hiyaapi.com` logs making/receiving of a phone call.

We did not observe any substantial event logging by Huawei, Realme (including Heytap), LineageOS or e/os.

On the Xiaomi and Huawei handsets the Google messaging app `com.google.android.apps.messaging` uses Google Analytics to log user interaction, including screens/activities viewed plus duration and timestamp, and logs the event that text is sent. In addition, with the notable exception of the e/os handset, Google Play Services and the Google Play Store collect large volumes of data from all of the handsets (see Figure 3.2). This has also been observed in other recent studies [92], which also note the opaque nature of this data collection (no documentation, binary encoded payloads, obfuscated code). From our discussions with Google we understand that they plan to publish documentation on this data collection/telemetry, but to date that has not happened.

Other event logging/telemetry that we observed is confined to operation of specific apps. On the Samsung handset the Microsoft OneDrive app sends data with device

```
{'event': {'metadata':
{'installId': b'\xe7\x19\xec\xa8KD\xff\xa1&E\xa3\x066G\xf6[', 'appVersion': '7.8.3.5', '
timestamp':
    {
    'utcTimestamp': 1628165014657, 'utcOffsetMins': 0}, 'vectorClock': {'major': 103, 'minor':
482, 'order': 100}
    },
    'application': 'com.google.android.apps.messaging', 'durationMillis': 6891,
    'typingStats':
        {'totalTokensEntered': 0, 'tokensFlowed': 0, 'tokensPredicted':
        0, 'tokensCorrected': 0, 'tokensVerbatim': 0, 'tokensPartial': 0, 'netCharsEntered': 3, '
deletions': 1, 'characterKeystrokes': 0, 'predictionKeystrokes': 0, 'remainderKeystrokes': 0, '
predictionSumLength': 0, 'typingDurationMillis': 837, 'emojisEntered': 0, '
totalTokensEnteredEdited': 0, 'tokensFlowedEdited': 0, 'tokensPredictedEdited': 0, '
tokensCorrectedEdited': 0, 'tokensVerbatimEdited': 0, 'tokensPartialEdited': 0},
    'languagesUsed': 0, 'termsPerLanguage': {}, 'tokensPerSource': {}, 'tokensShownPerSource':
{'': 6, 'en_GB/en_GB.lm': 16, 'user/dynamic.lm': 6},
    'userHandle': 0
}}
```

Figure 3.4: The Microsoft Swiftkey keyboard logs user interaction with the messaging app when sending a textl.

details and installed Microsoft apps to `mobile.pipe.aria.microsoft.com` and `app.adjust.com`, and uses Firebase push messaging. Events and data related to the mobile operator app `com.altice.android.myapps` are logged to `sun-apps.sfr.com` and via Google Analytics (e.g., duration app has been active, errors, stack traces). On the Realme handset events related to app `com.heytap.mcs` (launch etc) are logged to `dceuex.push.heytapmobile.com`. On the Huwaei handset events related to app `com.huawei.himovie.overseas` are logged to `pebed.dmevent.net`, and when a new app is installed the app details are sent to a scanning service at `apkrep.ff.avast.com`.

Key findings: *It is not surprising that device and user identifiers would be extensively collected. However, the analysis also reveals that many gray areas exist in terms of private data information collection, such as configuration data, list of installed apps and telemetry. Those can be collected without runtime permissions, thus users can hardly notice.*

Q3: Can Resettable Identifiers Be Relinked to Device?

In response to privacy concerns, identifiers used to track user behavior are now often resettable [177]. For example, the Google Advertising Identifier (GAID) can be reset via the Settings app on an Android handset. The idea is that by resetting such an identifier a person effectively unlinks themselves from the data collected about them in

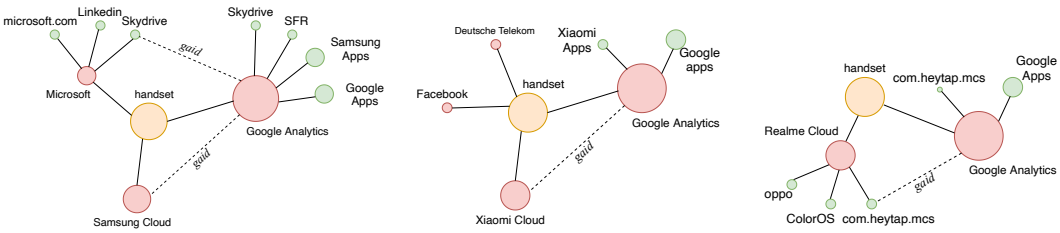


Figure 3.5: Potential for cross-linking data collection with different handsets: Samsung (left), Xiaomi (center), Realme (right). Red circles represent data collectors and green circles represent for what specific service instance the data is collected. Observe the potential of cross-linking through the Google Advertising I.

the past and starts afresh. However, this aim is largely subverted as the data collected allows relinking of the new identifier to the same physical user/handset. We find that data collection allowing the potential for relinking is commonplace.

Note that we are not in a position to know whether such relinking actually takes place. However, by observing identifiers sent together in the same data connection, we can see whether such relinking could be easily carried out, if desired.

It can be seen from Table 3.1 that Samsung, Xiaomi, Realme, Huawei and Google all collect long-lived identifiers from the handset, e.g., the IMEI (which is typically engraved on the SIM slot) or hardware serial number. These identifiers persist across factory resets of the device and are effectively permanent and indelible. If a long-lived identifier is sent in the same connection as a resettable identifier, then relinking of the resettable identifier to the handset is trivial. If one such resettable identifier can be relinked, and is then sent in a connection with other resettable identifiers, then these too can be relinked to the device. Using such an analysis we find that many of the resettable identifiers used by Samsung, Xiaomi, Realme, Huawei and Google can be relinked to the device.

The relevant identifiers are detailed in Table 3.1. Google can relink both the Google AndroidID and Google Advertising Identifier to the device. Xiaomi and Realme can relink the Google Advertising Identifier to the device, as well as all of the other identifiers commonly sent in connections. The same applies to Heytap on Realme handsets. Samsung can relink their Consumer ID, which is sent in many connections to Samsung servers, to the device. Samsung also collects Google Firebase identifiers/authentication tokens (used in conjunction with Google Analytics, etc.) and they can potentially relink these to the device via Google since the Google AndroidID is sent in Firebase connections and Google can relink the AndroidID to the device. Hence the Google Analytics data collected by Samsung system apps can potentially be relinked to the

device. Huawei sends the handset hardware serial number (a long-lived device identifier) in connections, but we observed little use of other identifiers and no potential for relinking of resettable identifiers by Huawei. We also did not observe any potential for relinking on LineageOS and e/os.

Key findings: *Technically, Google, Xiaomi, Realme and Samsung can relink resettable identifiers to devices, which questions the actual privacy practices of temporary IDs in the wild.*

Q4: Potential For Cross-linking Data Collection?

We find that typically multiple parties collect data from a handset. For example, on a Samsung handset Samsung, Google and Microsoft/LinkedIn all collect data. That raises the question of whether the data collected separately by these parties can be linked together (and of course combined with data from other sources). While we are not in a position to know whether such linking actually takes place, by inspection of the identifiers jointly collected by the parties we can see whether the potential exists for data linking.

Figure 3.5 illustrates these potential linkages as a graph.

Samsung record the Google Advertising Id, as do Google and there is therefore immediately potential for Samsung and Google to link their separate data. It is also worth noting that a number of Samsung system apps use Google Analytics to log data. Google already make some of their own data visible to third parties via the Google Analytics dashboard interface, e.g., user demographics, and so limited data sharing from Google to Samsung is likely taking place via that channel.

On the Samsung handset the Microsoft system app sends data to Microsoft servers and to `app-adjust.com`, and presumably Microsoft have access to the data that their app sends to `app-adjust.com`. The Google Advertising ID is sent to `app-adjust.com`, potentially allowing linkage to Google handset data. A LinkedIn system app also collects data. Since Microsoft own LinkedIn they may have access to that data, as well as other data held by LinkedIn.

Xiaomi records the Google Advertising Id, as do Google, and so linking of their data is possible. Xiaomi can display adverts within handset system apps and the UI and so some limited data sharing from Google to Xiaomi may be occurring via the that channel. We also note that a Facebook system app is installed in the Xiaomi handset

and the Facebook Ad SDK is embedded in a number of Xiaomi system apps, and so there appear to be connections between Xiaomi and Facebook although we saw no evidence of sharing of identifiers in our measurements.

On the Realme handset Heytap records the Google Advertising Id as do Google, and so linking of Google and Heytap data is again possible. In its connections Realme sends an identifier supplied by a Heytap server (the registrationId is sent by `shorteuex.push.heytapmobile.com`) and so linkage of data collection by Realme and Heytap is possible, and via Heytap with Google.

On the Huawei handset a hash of the handset `android_id` is sent to `avast.com` and an uuid is sent to `360safe.com` but neither seem easily linked to the hardware serial number sent to Huawei servers. The Swiftkey keyboard sends the Google advertising is to `telemetry.api.swiftkey.com`, but we did not observe this ID being sent to Huawei servers.

Key findings: *Cross-linking are possible given that the same identifiers are transmitted to different party's endpoints. It remains unknown how the data is processed in the background and vendors are responsible for clarifying it.*

3.3.3 Traffic Analysis - Chinese Firmware

The mobile devices with Chinese Firmware share a large degree of similarity with Global/European Firmware in terms of identifier and telemetry collection. However, the former also collects a broader range of PII. In this section, we compare the difference between the firmware released in two regions. The comparison is limited to Xiaomi, Oppo and OnePlus, since we do not have the counterparts of Huawei and Samsung purchased in the Chinese market. Table 3.5 provides an overview of the PII collection between Chinese and European firmware.

Q5. Which type of personally identifiable information (PII) is uploaded to backend servers by Chinese firmware?

We classify important PII into four categories and in Table 3.5 present which specific identifiers or information is actually shared.

Device-specific PII It appears that the handsets studied routinely upload device-specific PII, regardless of brand and firmware. Installed applications and the version of Android OS is posted periodically to check for updates. Hardware information, includ-

PII type		Redmi (Global)	Realme (Global)	Redmi	OnePlus	Realme
Device specific	IMEI	✓	✓	✓	✓	✓
	temporary IDs	✓	✓	✓	✓	✓
	installed apps	✓	✓	✓	✓	✓
	OS version	✓	✓	✓	✓	✓
	hardware info	✓	✓	✓	✓	✓
Geo-location	MCC		✓	✓	✓	✓
	MNC		✓	✓	✓	✓
	CID			✓	✓	✓
	LAC			✓	✓	✓
	GPS			✓	✓	✓
	connected Wi-Fi		✓	✓	✓	✓
	nearby Wi-Fi			✓	✓	✓
User profile	phone number			✓	✓	✓
	IMSI		✓	✓	✓	
	ICCID			✓	✓	
	app usage		✓	✓	✓	✓
Social relationship	SMS history				✓	✓
	call history				✓	✓

Table 3.5: Specific types of PII uploaded by each handset. 'Temporary ID' represents IDs created by vendor packages or OS, which would change after factory reset.

ing phone model, CPU brand and screen size are also posted to the backend servers. Besides, persistent identifiers, such as IMEI, and resettable identifiers, including VAID, OAID and Android ID, are embedded in a number of requests for device registration, telemetry, checking for updates, etc. However, we notice that the mobile devices with the global version of firmware only transmit the IMEI to the vendor, while those with CN version also upload the IMEI to China Unicom (`dm.wo.com.cn`) and China Mobile (`a.fxltsbl.com`) for registration after factory reset and when a SIM card is inserted, despite the handsets not having a contract with any of these communication service providers. A quick search reveals that the URLs correspond to the device management platforms for the mobile operators, and apparently any Android device in China Mobile's product library (including non-contract phones) should support the SDK for device management. Although we did not find the official regulation file, the SDK is installed on all the devices with CN firmware, named `com.miui.dmregservice` on Xiaomi and `com.coloros.regservice` on OnePlus and Realme. Different from

China Unicom, the China Mobile SDK also collects and uploads the installed package list in a request to `https://a.fxltsbl.com/accept/sdkService`, for unknown purpose.

Despite the same start-up configuration on Redmi and Redmi (Global) described in Section 3.2.2, we discover in the payload to `api.ad.xiaomi.com` that the handsets have different system advertising configuration:

```
POST https://api.ad.xiaomi.com/brand/pushConfig
clientInfo={...{"locale":"zh_CN","language":"zh","country":"CN","
isPersonalizedAdEnabled":true,...}}

POST https://api.ad.intl.xiaomi.com/brand/pushConfig
clientInfo={...,{"locale":"en_US","language":"en","country":"IE""
isPersonalizedAdEnabled":false..}}
```

That said, this feature cannot be controlled by opting out of 'Send Usage Data' or 'User Experience Program', but this is an inherent difference between the CN and Global firmware. It is still unclear how this option would impact advertising behaviors though, which is a decision to be taken at the backend side.

Geolocation PII Device-related PII are posted to backend servers regardless of firmware, but we find that the uploading of geolocation-related PII appears substantially different between the global and CN firmware. For Realme (Global), only the MCC and MNC are uploaded to `confe.dc.oppomobile.com`, which allows the server to confirm the country and the mobile operator in use. Redmi (Global) sends the connected Wi-Fi SSID and MAC address to `tracking.intl.miui.com`, which leaks user coarse location if the server maintains a database with Wi-Fi access point information.[6]

However, for handsets with CN firmware, traffic analysis reveals that the full range of the geolocation-related PII is uploaded, and to a range of recipients. The MCC, MNC, CID and LAC are transmitted to China Mobile and Unicom when a SIM card is inserted. By using the CID and LAC, these Chinese mobile operators can therefore easily infer coarse user location. A notable fact in Table 3.5 is that the three handsets with CN firmware also upload GPS coordinates. This happens even for the Redmi handset for which we chose to disable location during device startup following factory reset (the OnePlus and Realme handsets do not offer this option and have location service turned once the devices are started).

[6] 3WiFi: open database of Wi-Fi Access Points passwords `https://miloserdov.org/?p=746`

Both the OnePlus and Realme handsets have the `com.coloros. weather.service` and `com.amap.android d.location` apps preinstalled, which are granted the *ACCESS_BACKGROUND_L -OCATION* permission by default, and upload the current GPS location periodically.

The weather app makes the following request:

```
1 POST https://i6.weather.oppomobile.com/weather/location/v0/sdk?appId=app-
  weather&authCode=3357..
2 { 'latitude': <anonymized>,'language': 'ZH-CN', 'longitude': <anonymized
  >,'vaid': '63..5B', 'oaid':..}
```

in which latitude and longitude are populated with encrypted GPS location and the associated AES keys. The request contains identifiers (OAID and VAID), which can be combined with data from other connections to identify the individual handset, and is sent to an Oppo server approximately 5 times per day. AMap encapsulates the encryption algorithm in a JNI library, and sends the ciphertext in the following message:

```
1 POST http://apsrgeo.amap.com/rgeo/r?q=1&language=cn
2 {"data":"...<latitude><anonymized><\/latitude><longitude><anonymized><\/
  longitude>..."}
```

The connections to this domain are recorded 13 times a day, with an ID *adiu* associated, allowing Amap's server to analyze the user's location over time. Moreover, AMap also transmits the LAC, CID, the connected Wi-Fi MAC and SSID, as well as nearby Wi-Fi MACs and SSIDs in the request shown below:

```
1 POST http://aps.amap.com/APS/r?ver=5.1&q=0&csid=d6...
2 {"mcc":<anonymized>,"mnc":<anonymized>,"lac":<anonymized>,"cid":<anonymized
  >,"wifiStatus":{...,"mainWifi":{"mac":"<anonymized>","ssid":"\"
  androidprivacy\"
3 ", "wifiList":[{"mac":"<anonymized>","ssid
4 ":"Xiaomi_8DEE",...}...]}
```

where *androidprivacy* is the Wi-Fi network we deployed for traffic collection purposes and *Xiaomi_8DEE* is a nearby Wi-Fi. Note that the actual payload is encrypted and posted in a compact format, in which only the values are populated in a byte array.

In addition, the `com.coloros.wifisecuredetect` app on the OnePlus and Realme handset transmits the connected Wi-Fi SSID and MAC address to `log.avlyun.com`.

```
1 POST https://log.avlyun.com/logupload?channel=coloros_wifi&pkg=com.coloros.
  wifisecuredetect
```

```
2 #SYSINFO;{"kernel_version":"","name":"OnePlus9R_CH","security_patch
  ":"2021-03-05","sdk":"30","incremental":"1618459936301","base_os":"","
  platform":"kona","manufacturer":"OnePlus"}
3 #WIFI;635eb9bf(timestamp in hex); androidprivacy (wifi name);<anonymized>
   (Mac address);PSK;;;...
```

The Redmi handset does not grant runtime permissions to the map and weather apps by default, but allows system services to access location information in the background after startup. The `com.miui.analytics` system app sends the MCC, MNC, GPS coordinates and nearby Wi-Fi access point to `https://tracking.miui.com/track/v4` as shown below:

```
1 POST https://tracking.miui.com/track/v4
2 {"radio": "<anonymized>", // MCC+MNC
3 "loc": "{\"w\":\"<anonymized_MAC_addr>,androidprivacy,-37,5500\",...\"wl
  \":[\"<anonymized_MAC_addr>,Xiaomi_8DEE,-31,2422\",...,\"lat\":<
  anonymized>,\"lng\":<anonymized>}
```

and `com.xiaomi.metoknlp` encrypts and transmits the GPS coordinates to Baidu Map API to retrieve street information:

```
1 GET https://api.map.baidu.com/geocoder/v2/?channel=nl.1269e&coordtype=wgs84ll
  &&from=BaiduNLP&location=<anonymized>
```

Both services run in the background. The former sends geolocation PII right after a SIM card is inserted, while the latter is observed once within the first 24 hours of leaving the device idle after factory reset. Note however that we found no device/user identifiers in the request to Baidu.

User profile PII We group user phone number and ICCID (SIM card identifier) as a part of user profile information because every phone number in China is registered under a citizen ID, thus functions as a semi-persistent identifier of the user. Due to the existence of China Unicom SDK on Redmi and OnePlus, phone number, IMSI and ICCID are transmitted within the first 10 minutes after factory reset. The Phone and Message apps on OnePlus and Realme would send contact information when receiving text/calls, which is intended for "recognizing unknown number", but also leaks app usage information because of the timestamp and call duration in such requests, as detailed in what follows.

On the other hand, Redmi collects app usage data from a much broader range with greater level of detail. We uncover that Redmi posts requests to `tracking.miui.com/track/v4` when the preinstalled Settings, Note, Recorder, Phone, Message and

Camera apps are opened and used. The app's first launch time, usage start time, end time, and the timestamps when accessing some Android activities, such as WifiProvisionSettingsActivity, NotesList Activity, EditActivity and Camera, are uploaded. If the user is accessing preinstalled Xiaomi apps consecutively, telemetry would be logged frequently, resulting in a chain of actions that the user conducts. A snippet of the relevant request is shown below:

```
POST https://tracking.miui.com/track/v4
{ "imsis": "[b2d5c6783e3fa6eef38ff1fc7dedfb10,]",..,
{"pkg": "com.xiaomi.smarthome","action": "first_launch", "fit":
1666816796000, ...},
{"pkg": "com.android.settings","ts": 1666818456958,"duration": 1424, ...},
{"pkg": "com.miui.securityinputmethod","ts": 1666818463544,"duration": 4706,
... },
{"pkg": "com.miui.notes","ts": 1666818784908,"stat": "app_start",...}...}
```

This type of telemetry is not affected by opting out of 'Send Usage and Diagnostic Data' during start-up.

Redmi (Global) also collects IMSI and app telemetry, but the other user profile PII seen above are not transmitted from Redmi (Global) and Realme (Global).

Social relationships PII The preinstalled Phone (`com.ted.numb er`) and Message (`com.android.mms`) apps on OnePlus and Realme not only transmit user's phone number, but also send the other party's phone number with the duration when making/receiving a phone call or sending/receiving a text message. A typical request generated with a phone call is shown below:

```
POST https://sms.ads.heytapmobi.com/new/v5/phone
{"header":{.."p7":"user-phone-number",...,"data":{"phone":"
caller-number","dialNumber":"caller-number",...,"duration":15,"contact
":-1,"ringtime":0,"lasttime":-1}}
```

in which call duration, ring time, last contact time and whether the caller is in the contact is posted to the backend server. This type of PII not only helps the vendors to identify individual users, but also leaks users' social relationships. For the group of population who are inclined towards these brands, manufacturers can even build a connection map among them on the backend side, and infer the social relationships between users who are not directly connected.

Key findings: *CN OS distributions transmits a much larger range of PII to backend servers than Global distribution do, despite the fact that they are developed by the same companies. This is facilitated by 1) the granting of dangerous permissions to some pre-installed apps by default; and 2) a number of preloaded third-party apps being allowed to run continuously in the background.*

Q6. Is the user notified about the transmission of PII regarding geolocation, user profile and social relationships?

During device start-up following factory reset, each handset may present the user with information on the permissions used and data collected by preinstalled apps. Also, the first time an app is opened the terms and conditions of the specific app may be shown, and permissions may be asked. Having observed the data actually sent from each handset, we now revisit the information provided to users to check whether this matches the actual data transmission we observe.

Xiaomi The following system apps transmit geolocation and user profile PII:

1) `com.miui.anlaytics`: The statement in the startup page shows that Analytics would use location service and transmits the IMSI. "If you don't agree to grant such permissions ... You can choose not to use this feature in such cases." However, no option is given to restrict the permissions of this app and this app cannot be managed in Settings > Apps.

2) `com.xiaomi.metoknlp`: We did not find any relevant statement for this app, which is granted *ACCESS_BACKGROUND_L- OCATION* and posts GPS coordinates to the Baidu Map API.

3) `com.mobiletools.systemhelper`: This app embeds the China Unicom SDK. A statement on carrier services gives the types of information (including coordinates) to be collected.

4) `com.miui.regservice`: This embeds the China Mobile SDK. Similarly to the above, a statement on carrier services details the types of information (including coordinates) that will be collected by China Mobile.

A shared feature of these statements is that the user is presented with a 'take-it-or-leave-it' choice by the vendor and mobile operators together, and the user is unable to fully control the permissions of preinstalled packages. For example, if the user would like to avoid GPS coordinates being uploaded, the only option is to turn Location off in 'Settings', which applies to all apps and so likely to cause significant inconvenience.

Note also that when starting the Redmi handset we chose to disable Location, but we found that the location service is still running once we enter the system.

OnePlus & Realme Since both handsets are shipped with ColorOS, PII transmission on these devices share a large degree of similarity. The following apps transmit geolocation, user profile and social relationships-related PII:

1) `com.coloros.weather.service`: The statement on the startup page shows that "during your use of the 'Weather Service' [...], we need to use your location information". However, the Weather app is never opened during our data collection campaign.

2) `com.amap.android.location`: We did not find any relevant statement for this app to access location information.

3) `com.ted.number`: When opening the Phone app for the first time, the user is asked to agree to terms, including permissions to read call logs to identify unknown numbers. However, the user is not offered the option to decline these terms.

4) `com.android.mms`: The above situation also applies to the Message app.

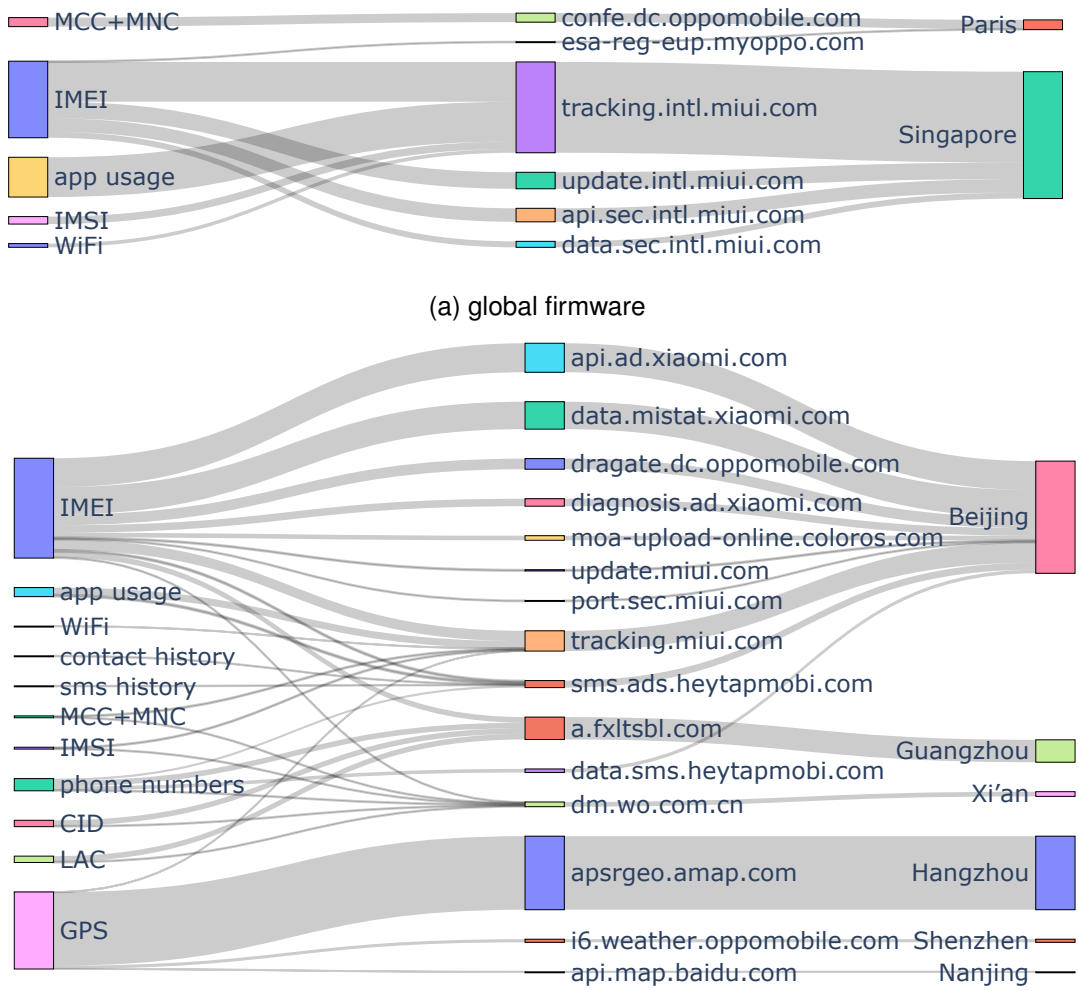
5) `com.coloros.regservice`: This embeds the China Mobile SDK. The startup statement informs users that this package has the permission to access location information and would regularly upload PII to the backend server.

6) `com.mobiletools.systemhelper` (OnePlus only): This package embeds the China Unicom SDK. A statement about the carrier's term of service details the information to be collected.

Compared with Xiaomi, we find that OnePlus and Realme are even less transparent about the use of location information. It should be noted that the Weather Service and Weather are two different packages, in which the former has *ACCESS_BACKGROUND_LOCATION* granted by default and runs in the background, and thereby we can observe coordinates being uploaded. However, if the user checks the permission status of Weather in Settings > Permission Manager, access to location information is never granted because it is the Weather Service that has the runtime permission. Other packages also trap users in a "take-it-or-leave-it" situation, where a 'decline' button is not provided.

Key finding: *In general, the user is not notified about the transmission of important PII or is not given the option to reject such transmissions when being notified.*

Q7. Who collects PII and where are they located?



(a) global firmware

(b) CN firmware

Figure 3.6: Sankey diagrams of important PII collected by the handsets running CN and Global firmware respectively.

Figure 3.6a and Figure 3.6b visualize, respectively, the PII collection in the CN and Global distributions. For the same device setup after factory reset, it can be clearly seen that the handsets running CN firmware collect a wider range of PII and transmit to more endpoints owned not only by phone vendors but also by third-party domains, in which the IMEI is the most frequently collected persistent identifier. The AMap system app on the Realme and Oneplus handsets regularly transmits GPS coordinates when the devices are idle, ranking the second in Figure 3.6b. On handsets running CN firmware, the phone vendors, weather and navigation providers and mobile operators all collect important PII in the background. It is not surprising to see that all of the servers associated with the endpoints are located in China, since the China Data Protection Law explicitly restricts personal data from being transferred abroad without

administrative consent. For handsets running the Global firmware, the IMEI is also the most frequently transmitted persistent identifier, followed by telemetry data collected by the Redmi Global firmware. However, we do not find that any preinstalled third-party apps collecting geolocation, user profiles or social-relationship PII. It should be noted that the measurement traces for the Redmi Global firmware were collected one year ago when DNS resolution of Xiaomi's domain pointed to IP addresses in Singapore, which may be different from newer DNS records.

Key finding: *Important PII transmitted by CN firmware reaches many third-party domains, whereas this is not witnessed with Global firmware.*

3.3.4 Root of Differences - Permission Analysis

The analysis above reveals the differences of data transmission between Chinese and Global firmware. In this section we aim at uncovering the cause of such behaviors from the perspective of preinstalled packages and requested permissions. Third-party app developers often collaborate with mobile device manufacturers, who embed popular third-party applications in the official firmware for a specific region. For example, it is common for Chinese brands to preinstall Chinese input apps, video streaming apps (such as Youku and Tencent TV), and domestic map apps (e.g., Baidu Map and AMap), due to the governmental ban on Google services. From a user perspective, this constitutes product bundling and preinstalled applications may excessively request permissions without the user knowing it.

Q8. How many apps are preinstalled in each handset?

The entire list of installed Android packages is maintained in `/data/system/packages.list`, and the requested permissions and whether each permission is granted, can be acquired by running `dumsys package <packgename>` on Android devices. Table 3.8 shows the details of the preinstalled packages found on the devices we study, and the average number of requested (dangerous) permissions encountered on each. Specifically, we group preinstalled packages into three categories, AOSP packages, vendor packages and third-party packages. It is the vendor who decides which type of hardware (CPU, fingerprint sensor, etc.) to use during production, thus we count hardware-supported packages including Qualcomm and Mediatek in the 'vendor pkg' category. Moreover, we also group the packages developed by parent/child companies

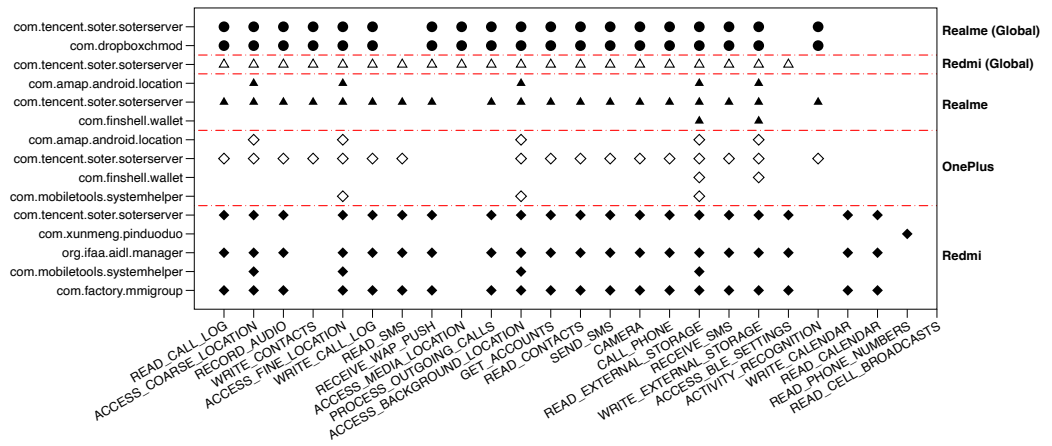


Figure 3.7: Diagram showing the runtime permissions granted by default to third-party packages by each handset.

into the 'vendor pkg' class. We find that the number of preinstalled packages among different brands with Chinese firmware is roughly the same, with vendor packages on OnePlus slightly outnumbering that on the other two brands. This is because OnePlus shares the same Android OS distribution (i.e., ColorOS) with Realme, but also loads a number of OnePlus self-developed apps. There are more than 30 third-party packages deployed in each handset with Chinese firmware, including multiple similar types of application. For example, *Redmi Note 11* is bundled with three Chinese input apps, namely, Baidu Input, IflyTek Input and Sogou Input. Both *OnePlus 9R* and *Realme Q3 Pro* preinstall Baidu Map as a foreground navigation app but also load the AMap package, which is continuously running in the background. News, video streaming and online shopping apps are bundled with all the CN firmware. It is notable that substantially fewer third-party apps are preinstalled in the Redmi (Global) and Realme (Global) OS distributions.

Key findings: *Product bundling in CN firmware is more extensive than in global firmware, and the CN firmware preinstalls multiple applications of the same type.*

Q9. How many permissions are requested by system services and preinstalled third-party apps?

We can see from Table 3.8 that the Android packages and vendor packages in the CN distribution request roughly the same number of permissions as those in the Global distribution. It can also be seen that the permissions requested by vendor packages are more than twice as many as those requested by Android packages. [179] found that

more than 80% of preloaded vendor packages are over-privileged, which is consistent with our own measurements. For example, the package for fingerprint authentication `com.goodix.fingerprint` requires permission to access Calendar, Camera, Contact, Call log and Audio recording. This appears to be a common feature across different firmware.

On the other hand, third-party packages preinstalled in all the handsets with CN distributions request many more permissions on average than Global distributions. The reason, apart from over-privileging, may be that the bundled apps request a number of permissions declared by different phone manufacturers to ensure maximal compatibility of the apps. For example, `com.taobao.taobao` on OnePlus and Realme request permissions declared by Meizu, Google, Samsung, Vivo and Huawei, although this package is installed in ColorOS. We include a detailed list of all the requested permissions and their frequency in Table 3.6 and 3.7. The number at the end of each entry denotes the average times that this permission is requested by third-party packages per handset. Table 3.6 only shows a subset of permissions which occur over 7 times on average.

Given that some custom permissions declared by different vendors serve the same purpose and could be potentially distracting, we take a closer look into the so-called dangerous ones, as defined in the official Android documentation [4] and which require runtime consent from users by default. In Figure 3.8 we plot the Empirical Cumulative Distribution Function (ECDF) of the number of requested dangerous permissions with respect to each type of preinstalled packages. A common observation in the first four subplots is a surge in the 'vendor' curve at $x \sim 20$ from ~ 0.75 to 1, meaning that 25% of the vendor packages request around 20 runtime permissions. An exception is Realme (Global) which has a version of mobile OS (realme UI v1.0) that is older than the Realme (CN) (realme UI v2.0), and package over-privilege seems more prevalent. Third-party packages on CN firmware also follow a similar pattern in that around 10% of them ask for no dangerous permissions, while the rest varies within 20 permissions. However, third-party packages on Global firmware request many fewer dangerous permissions, with 60% on Redmi Global and 50% on Realme Global not using any dangerous permissions.

permission	permission
android.permission.INTERNET (26.7)	android.permission.ACCESS_NETWORK_STATE (26.7)
android.permission.READ_EXTERNAL_STORAGE (25.7)	android.permission.ACCESS_WIFI_STATE (25.3)
android.permission.WRITE_EXTERNAL_STORAGE (25.0)	android.permission.READ_PHONE_STATE (24.7)
android.permission.VIBRATE (23.0)	android.permission.WAKE_LOCK (22.7)
android.permission.REQUEST_INSTALL_PACKAGES (22.0)	android.permission.ACCESS_FINE_LOCATION (21.7)
android.permission.ACCESS_COARSE_LOCATION (21.7)	android.permission.CAMERA (21.7)
android.permission.GET_TASKS (21.3)	android.permission.CHANGE_NETWORK_STATE (21.0)
android.permission.RECORD_AUDIO (21.0)	android.permission.CHANGE_WIFI_STATE (20.3)
com.android.launcher.permission.INSTALL_SHORTCUT (19.3)	android.permission.FOREGROUND_SERVICE (19.3)
android.permission.SYSTEM_ALERT_WINDOW (19.3)	android.permission.WRITE_SETTINGS (18.0)
android.permission.BLUETOOTH (17.7)	com.coloros.mcs.permission.RECIEVE_MCS_MESSAGE (17.0)
android.permission.MODIFY_AUDIO_SETTINGS (16.7)	com.android.launcher.permission.READ_SETTINGS (15.7)
android.permission.RECEIVE_BOOT_COMPLETED (15.0)	android.permission.READ_CONTACTS (15.0)
com.android.launcher.permission.UNINSTALL_SHORTCUT (14.3)	android.permission.FLASHLIGHT (13.7)
android.permission.READ_LOGS (13.3)	com.huawei.android.launcher.permission.CHANGE_BADGE (13.0)
android.permission.MOUNT_UNMOUNT_FILESYSTEMS (12.7)	com.oppo.launcher.permission.READ_SETTINGS (12.3)
android.permission.USE_FINGERPRINT (12.0)	android.permission.REORDER_TASKS (11.7)
com.htc.launcher.permission.READ_SETTINGS (11.3)	com.huawei.android.launcher.permission.READ_SETTINGS (11.0)
android.permission.USE_CREDENTIALS (11.0)	com.oppo.launcher.permission.WRITE_SETTINGS (10.7)
android.permission.BLUETOOTH_ADMIN (10.7)	android.permission.EXPAND_STATUS_BAR (10.3)
com.huawei.android.launcher.permission.WRITE_SETTINGS (10.0)	android.permission.WRITE_CALENDAR (10.0)
android.permission.GET_ACCOUNTS (10.0)	com.meizu.flyme.push.permission.RECEIVE (9.7)
com.meizu.c2dm.permission.RECEIVE (9.7)	android.permission.MANAGE_ACCOUNTS (9.7)
com.heytap.mcs.permission.RECIEVE_MCS_MESSAGE (9.3)	android.permission.CHANGE_WIFI_MULTICAST_STATE (9.3)
com.sec.android.provider.badge.permission.WRITE (9.3)	com.sec.android.provider.badge.permission.READ (9.3)
com.vivo.notification.permission.BADGE_ICON (9.3)	android.permission.AUTHENTICATE_ACCOUNTS (9.3)
android.permission.READ_CALENDAR (9.0)	com.sonyericsson.home.permission.BROADCAST_BADGE (9.0)
com.android.launcher.permission.WRITE_SETTINGS (8.7)	android.permission.WRITE_SYNC_SETTINGS (8.0)
android.permission.BROADCAST_STICKY (8.0)	com.android.launcher3.permission.READ_SETTINGS (8.0)
com.bbk.launcher2.permission.READ_SETTINGS (8.0)	com.htc.launcher.permission.UPDATE_SHORTCUT (7.7)
android.permission.DISABLE_KEYGUARD (7.0)	com.sonymobile.home.permission.PROVIDER_INSERT_BADGE (7.0)
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS (7.0)	

Table 3.6: The most frequently requested permissions by preinstalled third-party packages in the CN firmware.

permission	permission
android.permission.ACCESS_NETWORK_STATE (3.3)	android.permission.INTERNET (3.3)
android.permission.WAKE_LOCK (2.7)	android.permission.RECEIVE_BOOT_COMPLETED (2.7)
android.permission.READ_EXTERNAL_STORAGE (2.3)	android.permission.ACCESS_WIFI_STATE (2.3)
android.permission.WRITE_EXTERNAL_STORAGE (2.0)	android.permission.READ_PHONE_STATE (1.3)
com.google.android.c2dm.permission.RECEIVE (1.3)	android.permission.FOREGROUND_SERVICE (1.3)
android.permission.VIBRATE (1.0)	com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE (1.0)
android.permission.REQUEST_INSTALL_PACKAGES (1.0)	android.permission.ACCESS_MEDIA_LOCATION (1.0)
android.permission.GET_TASKS (1.0)	android.permission.INSTALL_PACKAGES (1.0)

Table 3.7: The requested permissions by preinstalled third-party packages in the Global firmware.

Region	Model	android pkg			vendor pkg			3rd-party pkg		
		num	# perm	# dangerous permission	num	# perm	# dangerous perm	num	#perm	# dangerous perm
CN	Redmi	152	36.0	3.1	137	117.8	8.9	36	72.9	7.1
	OnePlus	159	32.8	3.2	185	72.9	6.7	40	51.9	6.5
	Realme	147	35.6.3	3.7	163	71.5	7.8	34	56.2	6.5
Global	Redmi	169	32.4	3.4	111	108	9.2	14	9.0	1.8
	Realme	166	35.1	4.1	139	100.0	9.9	9	6.2	4.1

Table 3.8: Summary of the number of different types of packages installed in each of the handsets studied, and the average number of permissions requested in each category. Note that we group hardware-supported packages, including Qualcomm, Mediatek, and packages developed by parent/child companies into the ‘vendor pkg’ category.

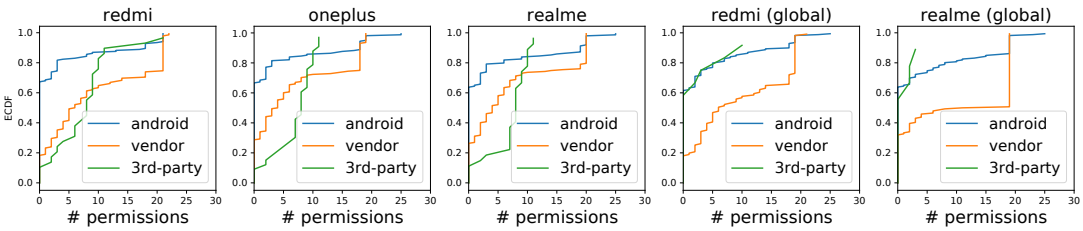


Figure 3.8: The ECDF of the number of dangerous permissions requested by each category of packages in each handset.

Key findings: *Vendor packages are over-privileged in both the CN and Global distributions. In CN distributions, pre-installed third-party apps ask for a significantly larger collection of permissions, including dangerous permissions, than in Global distributions.*

Q10. Do handset manufacturers grant dangerous (runtime) permissions by default to preinstalled third-party packages?

The number of requested permissions, to some extent, reflects the privacy awareness of app developers, but it is not necessarily an indicator of the privacy awareness of phone manufacturers. To confirm the latter, we also study whether runtime permissions would be granted to preinstalled third-party apps by default, without user interactions, which is under the control of the vendor. Note that third-party apps which circumvent permission systems on their own are not in the scope of our analysis, while interested readers can refer to [127]. We factory reset each handset to ensure that no user interaction is made with any applications, and then monitor the runtime permissions of each package via `dumpsys`. In Figure 3.7, we report the third-party apps that are granted runtime permissions by default, and the number of runtime permissions granted on each handset. It can be seen that `com.tencent.soter.soterserver`, an authentication package for WeChat Pay [157], is installed on all of the handsets and is automatically granted more than 17 runtime permissions, including access to location, recording audio, reading SMS and using the camera. A payment authentication module may use such permissions to verify user identity and receive one-time tokens. However, a range of seemingly unnecessary permissions are also granted by the handsets, including reading/writing call logs and reading the list of contacts. The `org.ifaa.aidl.manager` app is an equivalent for Alipay (another popular electronic payment system in China). Moreover, on OnePlus and Realme, the `com.amap.android.location` (Amap) app is

permitted to access background location without user consent, and this package transmits GPS coordinates periodically to the relevant backend server. `com.mobiletools.systemhelper` is a China Unicom device-registration SDK, which is also permitted to access in the background the location of the handsets. Our traffic analysis further reveals that a range of PII is transmitted by this package after factory reset.

Key findings: *Third-party packages are granted dangerous permissions by default, without the need for user interactions, resulting in user privacy exposure risks. The user is not informed and so may be completely unaware of this.*

3.4 Summary

We presented an in-depth analysis of the data sent by the Samsung, Xiaomi, Oneplus, Huawei, Realme, LineageOS and e/os variants of Android. We find that even when minimally configured and the handset is idle these vendor-customized Android variants transmit substantial amounts of information to the OS developer and also to third-parties (Google, Microsoft, LinkedIn, Facebook etc.) that have pre-installed system apps. While occasional communication with OS servers is to be expected, the observed data transmission goes well beyond this and raises a number of privacy concerns.

We also study the Chinese version of the Android OS distributions run by Xiaomi, Realme, and OnePlus handsets, and find that this firmware come bundled with a number of third-party applications, some of which are granted dangerous runtime permissions by default without user consent, and transmit traffic containing a broad range of geolocation, user-profile and social relationships PII to both phone vendors and third-party domains, without notifying the user or offering the choice to opt-out. In contrast, the data shared by the Global version of the firmware is mostly limited to device-specific information. Our research therefore highlights major differences in terms of how privacy provisions are enforced in different regions. In the next chapter, we examine privacy issues on the intermediate links of network communications and introduce a countermeasure against Internet censorship.

Chapter 4

Network Traffic Obfuscation against ML-supported Censorship

Messages transmitted between ordinary devices and servers traverse multiple hops within intermediate links, including routers and gateways owned by various Internet Service Providers (ISP). Usually, these messages are subject to a degree of screening, primarily for benign reasons, such as blocking porn for children, but the concerning trend of an increasing number of governments and control authorities implementing network traffic censorship is on the rise [139, 172, 182, 183, 51, 116, 125].

Network traffic censorship aims to restrict the citizens' access to online information that may be perceived by those regimes as politically, socially, or morally objectable. The suite of techniques applied in Chapter 3, including certificate planting and man-in-the-middle attack, represents an optimal strategy for content censorship. This strategy enables censors to achieve complete transparency over plaintext transmissions within the network. Nevertheless, the prevalence of end-to-end encryption and the role of certification authorities significantly reduce the likelihood of data interception. It is estimated that more than 95.2% of websites support TLS 1.2 or above [123]. Given that censors lack legal means to control end-user devices or to directly intercept transmissions at intermediate nodes, state actors resort to a range of tools including DPI, protocol fingerprinting, and active probing. In recent years, protocol tunneling has gained traction as a viable means to circumvent censorship. Tunneling leverages existing implementations of innocuous protocols (Skype, WebRTC, TLS, etc.) and embeds covert streams in these protocols to hide destination host identity, payload contents, etc. [10, 9, 48]. As a result, sensitive information becomes encrypted and message exchanges perfectly aligned with the behavior of the tunneling protocol. In turn, observ-

ing the tunnels barely unveils any deterministic fingerprints. Censorship is however an arms race, recent studies revealing that ML algorithms, which learn statistical features from network flows, can effectively identify 'offending' tunneled traffic, despite not exhibiting deterministic fingerprints [29, 170]. For example, although multiple multimedia tunneling tools claim unobservability, simple ML classifiers such as those based on decision tree and random forest structures can detect tunneled traffic with high confidence [8]. On the other hand, ML is also employed to devise censorship circumvention strategies. For example, Geneva [13] designs a genetic algorithm to discover if existing censorship can be evaded by tampering with canonical TCP implementations, e.g., by corrupting checksums, breaking Transmission Control Blocks (TCBs) (by injecting a `RST`), or segmenting packets with corrupted `ACK`s. While this attack targets the incompleteness of network stacks implemented by censors, in this paper we aim to push the boundary of anti-censorship one step further, where we reasonably assume the censor fixes the implementation issues and leverages ML classifiers to detect anti-censorship tools.

Since the inner workings of an ML-based classifier are largely unknown to users seeking to circumvent censorship and the censor can change the underlying neural architecture at any time (black box), the question we aim to answer in this work is: *Instead of perpetually designing new tunneling tools, can we devise adversarial attacks on black-box ML classifiers to consistently subvert ML-supported censorship?* This approach has not been well studied in the network censorship domain. In computer vision, finding adversarial examples, i.e., images that should be recognized as belonging to class A being instead misclassified as of class B, can be achieved by adding adversarial perturbations such that the modified input images remain visually similar to their original versions, but produce erroneous classification results [55, 3, 19].

Conducting adversarial attacks in the networking domain is fundamentally different. A common approach to ML-based network flow classification is to first extract multiple statistical features (packet size distributions, timing information, etc.), then feed these features to a classifier instead of raw flows [100, 8], as illustrated in Figure 4.1. Censors do not reveal what features they utilize, which poses difficulty in the first step of crafting an adversarial attack. Further, even if users may discover the set of features employed by a censoring classifier and successfully generate adversarial samples, there is no guarantee that such samples can be mapped back to legitimate flows, which renders the entire process unusable in practice. A practical adversarial attack against censoring classifiers requires manipulation at packet level, instead of feature

level, and each packet should be transmitted without adding significant delays. Early attempts [167, 114] apply attacks on complete flows and generate adversarial versions, yet each manipulated packet should be sent before new packets are received. Having a complete view of a flow to perturb is unfortunately unrealistic. The inherent imbalance between what censors can observe (flows) and what users can observe and manipulate (packets) rules out the possibility of applying existing algorithms from other domains to achieve adversarial attacks for censorship circumvention purposes.

In this chapter, we formulate the problem of finding adversarial flows against censoring classifiers as a packet sequence generation task. To solve it, we design Amoeba, a novel black-box attack through reinforcement learning, which learns to craft adversarial flows solely based on the classification results of censoring classifiers, without any further knowledge.

4.1 Adversarial Model

As use of ML gains traction in the networking domain, including for WF [120, 78, 119, 133, 149, 150] and network intrusion detection [100, 31, 107], censors are increasingly adopting ML-based classifiers to detect unwanted protocols or traffic associated with banned web services. We consider the most common setting where the censor has full control of the network gateway and enough computational power to examine every network flow traversing it. More precisely, the censor may collect network traffic generated by 'forbidden' protocols/web services along with innocuous traffic. A group of statistical features may be extracted from individual flows and fed to an ML classifier for training, as shown in Figure 4.1. The censor then deploys the ML model on the gateway to block sensitive flows, e.g., by using and maintaining a blacklist of (src_ip, src_port, dst_ip, dst_port, protocol) tuples on the firewall. That said, once a traffic flow is recognized as 'unwanted' by the censor, the pair of sockets used on the source and destination machine cannot communicate to establish a connection. The censor would not block the destination IP entirely, in order to prevent collateral damage, especially as CDNs increasingly serve thousands of service with the same address [42]. This is a reasonable practical assumption – for instance, the Great Firewall blocks port numbers instead of IP address when censoring Shadowsocks [11].

We consider broad scenarios whereby censors need not use deterministic fingerprints in the decision-making process, such as crypto scheme and SNI in TLS handshakes, since these fingerprints can be eliminated easily by fixing the protocol imple-

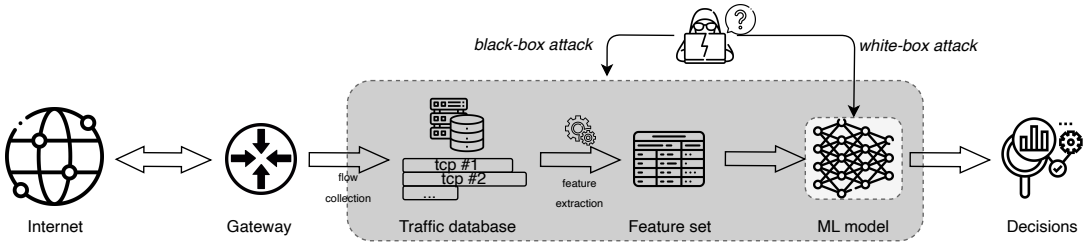


Figure 4.1: Typical traffic classification pipeline. An attacker may conduct adversarial attacks against the pipeline with different capabilities/scope. We categorize them into 1) *white-box model* for which the inner workings of the classifier (weights, gradients) are visible; and 2) *black-box model* for which attackers can only craft network flows and observe outputs without extra knowledge, such as feature engineering and model architecture.

mentation. Also, a censor would not conduct active probing, which is orthogonal to passive observation and outside the scope of our study.

We define different capabilities of an 'attacker' who attempts to circumvent ML-supported censorship as shown in Figure 4.1. The most rudimentary setting for adversarial attacks is the *white-box model*: the trained censoring classifier is available to the attacker who leverages weight and gradient information to perturb the inputs to the ML model. Under this setting, the attacker also knows the features extracted by the censor, thus perturbations are conducted directly in the feature space instead of on raw flows/-packets. A generated adversarial sample is the set of features of a flow, and converting the features back into a legitimate flow is not of this type of attacks' concern. The Carlini & Wagner (CW) attack [19, 59] uses projected/clipped gradient descent to find minimal perturbations on the inputs, such that the censoring model would misclassify. GAN-based methods [194, 114] treat the censoring classifier as the discriminator in a GAN and train a generator to produce adversarial samples.

However, given that the censor is unlikely to reveal the feature engineering process, the training technique employed and the architecture of ML models, we *define a stricter threat model* for adversarial attacks from a realistic perspective, to which we refer as *black-box attack*, as shown in Figure 4.1. Assume the attacker has access to a large number of machines with different IP addresses on both sides of the gateway, and can establish connections arbitrarily, as shown in Figure 4.2. The adversary may finely manipulate every network flow, by controlling packet sizes and packet inter-arrival times. Given that the censor may run the ML classifier almost at line rate, the attacker can quickly know whether a specific flow is forbidden by the censor. Under this setting:

1. The attacker does not know which statistical features the censor may extract from each flow;

2. The attacker does not know the architecture of the classifying ML model;

3. The ML model may not be built with NNs, but with traditional algorithms, e.g., SVM or DT, so gradient information is not guaranteed to exist.

This *black-box* setting gives the attacker very limited guidance on how to generate adversarial samples, while the inherent difference between the networking and other domains (e.g., computer vision) precludes the use of existing adversarial input manipulation techniques such as Square Attack [3], to circumvent censorship.

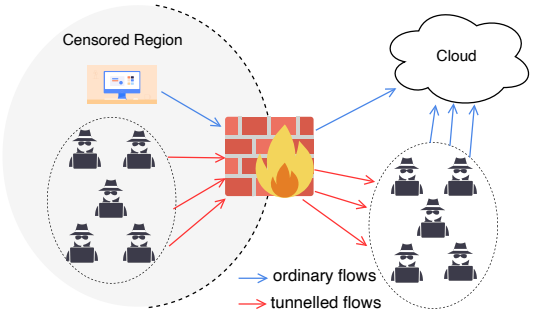


Figure 4.2: Strictest adversarial model considered for subverting Internet censorship. 'Attackers' with no knowledge of the censor's tools manipulate packet sizes and inter-packet times based on implicit feedback received (flow permitted or not), to find a tunneled traffic shaping strategy that evades censorship.

4.2 Problem Formulation

Adversarial flows that seek to subvert censorship must accommodate the original payloads and be transmissible in real-world network settings. Thus, we first define a set of practical constraints that adversarial flows must satisfy, then formulate adversarial attacks as a constrained optimization problem, which we solve with a purpose-built RL solution.

Constraints on Adversarial Attacks: We represent a network flow by a tuple $S = (P, \Phi)$, where P is a vector of n packet sizes, and Φ a vector of inter-packet delays, i.e.,

$$P = [p_1^+, p_2^-, ..., p_n^+], \;\; \Phi = [\phi_1, \phi_2, ..., \phi_n].$$

Superscript '+' indicates packets transmitted from client to server, and '−' vice versa. An adversarial sample can alter each packet size by padding or truncation, and can delay packets to deceive ML classifiers. However, the attacker must ensure that bidirectional payloads in the original flows are transmitted in the correct order. Denote $\tilde{S} = (\tilde{P}, \tilde{\Phi})$ as the adversarial version of flow S, where

$$\tilde{P} = [\tilde{p}^{+}_{1,1}, ..., \tilde{p}^{+}_{1,k_1}, \tilde{p}^{-}_{2,1}, ..., \tilde{p}^{-}_{2,k_2}, ..., \tilde{p}^{+}_{n,1}, ..., \tilde{p}^{+}_{n,k_n}],$$
$$\tilde{\Phi} = [\tilde{\phi}_{1,1}, ..., \tilde{\phi}_{1,k_1}, \tilde{\phi}_{2,1}, ..., \tilde{\phi}_{2,k_2}, ..., \tilde{\phi}_{n,1}, ..., \tilde{\phi}_{n,k_n}].$$

The sub-sequence $[\tilde{p}_{i,1}, ..., \tilde{p}_{i,k_i}]$ represents the adversarial manipulation of original packet sizes p_i, with $\{k_1, ..., k_n\}$ denoting the lengths of all sub-sequences. Since we allow for both packet truncation and padding, the length of an adversarial flow can be larger than that of the original, i.e., $|\tilde{P}| \geq |P|$, though the following constraint on packet sizes must be satisfied:

$$\sum_{j=1}^{k_i} \tilde{p}_{i,j} \geq p_i, \ \forall i \in [1, n], \tag{4.1}$$

which ensures that each original packet can be transmitted without data loss. It is straightforward to derive constraints on timestamps:

$$\tilde{\phi}_{i,1} + \sum_{t=2}^{k_{i-1}} \tilde{\phi}_{i-1,t} \geq \phi_i, \tilde{\phi}_{i,j} \geq 0, \ \forall j \in [1, k_i], i \in [1, n]. \tag{4.2}$$

Finding Adversarial Samples: Let $e(\cdot)$ be a feature extraction function that takes an arbitrary flow S and outputs d-dimensional features. Denote $f : \mathcal{R}^d \rightarrow [0, 1]$ a binary classifier. f can be a neural network using a sigmoid as the activation function in the final layer, so its output $y = f(e(S))$ is a real number between 0 and 1. Alternatively, f can be a traditional ML algorithm (SVM, decision tree, etc.), which directly outputs discrete classification results ($\{0, 1\}$). If using a NN-based classifier, the censor would use a decision function

$$C(y) = \begin{cases} 1, & \text{if } y \geq 0.5; \\ 0, & \text{otherwise.} \end{cases}$$

That said, if the predicted score is larger than 0.5, the flow is to be blocked. A flow $\tilde{S}$ is regarded as an adversarial version of S if $C(f(e(\tilde{S}))) = 0$. The task of finding $\tilde{S}$ can

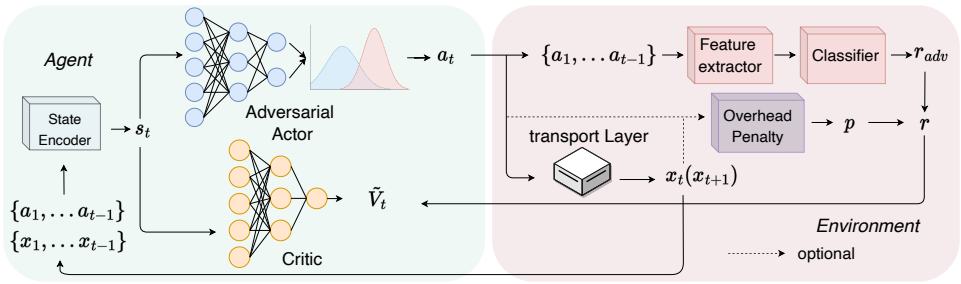


Figure 4.3: The architecture of the proposed adversarial reinforcement learning algorithm – Amoeba.

be rephrased as a constrained optimization problem:

$$\min C(f(e(\tilde{S}))) \text{ s.t. } \sum_{j=1}^{k_i} \tilde{p}_{i,j} \geq p_i, \;\; \tilde{\phi}_{i,1} \geq \phi_i, \;\; \tilde{\phi}_{i,j} \geq 0, \forall j \in [1, k_i], \;\; \forall i \in [1, n].$$

Are upper bound constraints necessary? Different from the computer vision domain, we do not impose upper bound constraints on both payload and timing. The adversarial examples $\tilde{x}$ of *an image* x must satisfy an l_p-norm bound, i.e., $||\tilde{x} - x||_p \leq \varepsilon$ [3], because $\tilde{x}$ should not tamper with the semantics of the original image x from a human perspective. An image of a panda should still 'look like' a panda after adversarial perturbation. However, in the networking domain, as long as the original payload is transmitted, and the sender and the recipient can interpret messages identically, the semantics remain the same. Therefore, minimizing data overhead and timing delays are not hard constraints for the problem we solve, but optional requirements that users may have in order to prevent performance degradation, for which we also offer a solution in Section 4.3.2.

4.3 Amoeba Architecture

Traditional adversarial attacks do not comply with the specifics of network flows, because (1) the length of adversarial samples are variable, according to each flow, and the optimal length is unknown; and (2) one should be able to send adversarial samples packet-by-packet, whereas existing attacks generate the feature set of an entire flow at once, without considering the practicalities of transmission.

Instead, **we regard finding adversarial versions of network flows as a sequence generation process**, which takes an input (a packet and the associated timestamp in

the original flow) at each timestep and outputs adversarial manipulations of that input. The adversarial packets can be transmitted in almost real-time, rather than waiting for the entire flow to finish first. Each packet in a flow should be morphed to maximize the chances that the complete flow in the future will be misclassified, which requires an algorithm to look ahead of time and progresses through binary signals received from the censor. Given these requirements, RL is particularly well-suited to our task, where we treat the output of the censoring classifier as a reward that guides the RL agent to learn a packet sequence generation policy. We design Amoeba to generate adversarial flows that circumvent censorship. Amoeba models censor decisions as a negative reward function, and trains an agent to interact with the censor in discrete timesteps. At each step t, the agent receives a packet from the transport layer, takes an adversarial action (effectively modifying the size and timing of the packet), and obtains a reward based on how good that action is. The agent aims to maximize the future rewards when generating adversarial samples. Note that **Amoeba does not change the implementation of any existing protocol** in terms of handshake, error handling and acknowledgment, but simply alters the 'shape' of each packet with payload to deceive ML classifiers. In other words, an adversarial TCP flow is still a legitimate TCP flow. Amoeba comprises four major elements: Network Environment, State Encoder, an Adversarial Actor and a Critic (see Figure 4.3). Next, we provide a RL primer, before diving into our solution.

4.3.1 Reinforcement Learning Primer

Our algorithm takes a reinforcement learning approach with an agent interacting with the environment in discrete timesteps. At step t, the environment gives an *observation* $x_t = (p_t, \phi_t)$, representing an original packet to send with size p_t and inter-packet delay ϕ_t. For each flow, the actor maintains a vector of previous observations $[x_1, x_2, ..., x_{t-1}]$, as well as a vector of previous actions $[a_1, a_2, ..., a_{t-1}]$, with each *action* $a_i = (\tilde{p}_i, \tilde{\phi}_i)$ representing the manipulation of an original packet x_i. In this paper, we use *actions* and *adversarial packets* interchangeably. The *state* at step t is the history of both the observations and the actions, i.e., $s_t = (x_1, a_1, ..., x_{t-1}, a_{t-1}, x_t)$. Note that $s_t \neq x_t$, because the actor needs a broad understanding of the current environment based on what has been generated up to that point. The actor follows a policy π, which maps a state to a probability distribution over the actions $\pi(s_t)$. An action is randomly sampled $a_t \sim p_\pi(s_t)$, and given to the environment, leading to a *reward* $r(s_t, a_t)$ and the next

observation x_{t+1}. An *episode* indicates the entire process of generating an adversarial sample given a flow, i.e. $(x_1, a_1, ..., x_T, a_T)$. The aim of the actor is to select actions at every timestep in a way that maximizes future rewards:

$$\max \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t) \right],$$

where $Q^\pi(s_t, a_t)$ is known as the *action-value function* that produces the discounted total future reward. Approximating Q values directly suffers from high variance in practice. Thus, a baseline is always subtracted from Q while keeping the objective unbiased:

$$\max \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right],$$

in which *Advantage* $A(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$. Here, the second term is called the *state-value function* $V^\pi(s_t) = \mathbb{E} a_t \sim \pi[Q(s_t, a_t)]$, which represents the expected future reward from step t, and $A(s_t, a_t)$ intuitively indicates how much better the current action a_t is than the average.

4.3.2 Network Environment

The network environment offers observations and rewards, given new actions.

Generating Observations In practice, observations (packets) originate from the buffer in the transport layer. When there is no traffic obfuscation in place, the payload in the buffer would be encapsulated in packets and transmitted immediately. However, to generate adversarial samples, the payload cannot be sent directly but should be passed through the adversarial actor, which decides appropriate packet sizes and timings, such that the Q value can be maximized.

Therefore, as the first step we use a transport layer emulator as shown in Algorithm 1 that reads payload with p_i bytes from the buffer as the vanilla transport layer does. To adversarially manipulate this packet (observation), $x_t = (p_i, \phi_i)$ is given to the agent, which morphs the packet based on a given policy π, truncating or adding padding to it along with some delays. Since we allow for truncation, it is possible that the adversarial packet (action) $\tilde{a}_t = (\tilde{p}_i, \tilde{\phi}_i)$ is smaller than the original one, leaving $p_i - \tilde{p}_i$ byte payload to send (Alg. 1 line 9). In that case, the emulator does not read more payload from the buffer, but generates a second adversarial packet by giving the agent $x_{t+1} = (p_i - \tilde{p}_i, \phi_{i+1})$. Such operation is repeated until the remaining payload is fully

Algorithm 1 Emulator of the transport layer

1: **Inputs:**
   MAX_UNIT := the maximum transmission unit
2: **Initialisation:**
   $read\_buffer(\cdot)$: returns payload from the buffer. The size of the payload should not exceed the given param.
   $read\_ipd()$: returns the InterPacket Delay.
   $send(\cdot)$: sends the given payload.
   $agent_\pi(\cdot)$: returns an action given an observation.
3: **while** an episode does not terminate **do**
4: $p_i \leftarrow read\_buffer(MAX\_UNIT)$
5: **while** $p_i > 0$ **do**
6: $\phi_i \leftarrow read\_ipd()$
7: $(\tilde{p}_i, \tilde{\phi}_i) \leftarrow agent_\pi((p_i, \phi_i))$
8: $send((\tilde{p}_i, \tilde{\phi}_i))$
9: $p_i \leftarrow p_i - \tilde{p}_i$ ▷ packet truncated
10: **end while**
11: **end while**

sent (Alg. 1 line 5-9). On the other hand, if the adversarial packet is larger than the original one, i.e., $\tilde{p}_i > p_i$, the emulator jumps out of the while loop as shown in Alg. 1 line 5, and reads more payload from the buffer. For example, assume the agent truncates an original packet n times, the list of the observations sent to the agent and the list of actions would be:

$$[(p_i, \phi_i), (p_i - \tilde{p}_{i,1}, \phi_{i+1}), ..., (p_i - \sum_{j=1}^{n-1} p_{i,j}, \phi_{i+n-1})], \text{ and}$$

$$[(\tilde{p}_{i,1}, \tilde{\phi}_{i,1}), (\tilde{p}_{i,2}, \tilde{\phi}_{i,2}), ..., (\tilde{p}_{i,n}, \tilde{\phi}_{i,n})].$$

Observe that **the emulator satisfies the constraint on packet sizes (Eq. 4.1) by design**, so that the adversarial actor does not have to consider it while learning the policy. Also, the observation x_t and the associated packet size p_i do not share the same subscript because the emulator may read from the buffer once, but uses multiple timesteps to send the payload, due to truncation. Assume a flow consisting of P bytes (excluding headers) in total, the time complexity of Algorithm 1 in the worst case is $O(P)$, i.e., truncating the original packets into P 1-byte packets. This not only introduces an unique fingerprint but also introduces large time overhead. In practice, we prevent this

extreme truncation from happening by penalizing the number of truncations, which is detailed in the next paragraph.

Reward Function Design. The reward function evaluates how good an action a_t is under the current state s_t. Since our aim is to find adversarial network flows, the reward should first reflect the judgment of the censor, i.e., $C(f(e(\cdot)))$. There are two standard strategies to assign rewards for each action-state pair. The first is not assigning intermediate rewards while the sequence is being generated, but only assigning a final reward when the episode terminates. One typical example is AlphaGo [147], which assigns either $+1$ or -1 when a round of go game ends. The other strategy is to give a reward at each timestep, which was adopted for cartpole or Mario game play. The first strategy might seem suitable for our task, since all the intermediate actions should serve the final aim, that is, the adversarial flow as a whole should be misclassified. However, this would imply that the environment knows in advance when a flow will terminate, so it defers a reward until the last packet. In reality, a flow may terminate at an arbitrary timestep due to different communication purposes or network status. Note that in our adversarial model, attackers can control each packet, meaning that they can also terminate a flow at any point. In other words, we consider it possible for the censor to make a classification decision at any timestep, as if this is the last in an episode.

Formally, consider $a_t = (\tilde{p}_{i,n}, \tilde{\phi}_{i,n})$ at timestep t is generated by the attacker given $x_t = (p_i - \sum_{j=1}^{n-1} \tilde{p}_{i,j}, \phi_{i+n-1})$ and sent over the network. The censor already witnesses $\mathbf{a}_{1:t} = [a_1, a_2, ..., a_t]$. Thus, the reward regarding distinguishability is defined as:

$$r(s_t, a_t)_{adv} = -C(f(e(\mathbf{a}_{1:t}))).$$

Besides, we also consider extra penalties in terms of data overhead and time delays. One may expect the adversarial sample is as close to the original flow as possible, i.e., introducing the smallest padding and delays, which would do minimal harm to the application performance. We therefore introduce a data overhead penalty and a time overhead penalty:

$$p(s_t, a_t)_{data} = |p_i - \sum_{j=1}^{n} \tilde{p}_{i,j}| + \lambda_{split} n. \quad p(s_t, a_t)_{time} = \tilde{\phi}_{i,n} - \phi_{i+n-1}.$$

When the size of the adversarial packet at timestep t is smaller than that of the original packet, the penalty is proportional to the number of truncations n plus the remaining

bytes to send. Otherwise, the penalty is linear in the remaining bytes. We do not use symmetric penalties for the two circumstances, because we find empirically that Amoeba is inclined to truncate packets into multiple instances of minimal size. Thus, we discourage this behavior by assigning an extra penalty when packet truncation occurs. The penalty for time delays is straightforward.

The expression of the reward function thus becomes:

$$r(s_t, a_t) = r(s_t, a_t)_{adv} - \lambda_d p(s_t, a_t)_{data} - \lambda_t p(s_t, a_t)_{time},$$

where λ_{split}, λ_d and λ_t are hyperparameters that balance each component.

4.3.3 StateEncoder

As mentioned in Section 4.3.1, the state at timestep t is the history of the observations and the actions, meaning that the length of the state would vary as t increases. However, if the agent is built with non-recurrent neural networks, such as MLP or CNN, it requires inputs of fixed size. To overcome this problem, we design StateEncoder, a two-layer, pre-trained GRU that encodes an arbitrary long network flow to a fixed-size hidden representation. As shown in Figure 4.4, to ensure that StateEncoder $\mathcal{E}$ can properly encode network flows without nontrivial information loss, we train $\mathcal{E}$ as the encoder part of a Seq2Seq Autoencoder, in which StateDecoder $\mathcal{D}$ shares the same architecture with $\mathcal{E}$. Consider a network flow $S = [s_1, ..., s_T]$ with T packets. $\mathcal{E}$ aims to map S as a representation in the H-dimensional hyperspace, $r_S = \mathcal{E}(S) \in \mathbb{R}^H$, and $\mathcal{D}$ aims to reconstruct the flow from the hidden representation, $\hat{S} = \mathcal{D}(r_S) \in \mathbb{R}^{T \times 2}$. We train the Seq2Seq Autoencoder with a Mean Squared Error (MSE) loss function, i.e.,

$$L(S, \hat{S}) = \frac{1}{T} \sum_{t=1}^{T} (s_t - \hat{s}_t)^2,$$

by the Adam algorithm [82]. The only connection between $\mathcal{E}$ and $\mathcal{D}$ is the hidden representations. Therefore, $\mathcal{E}$ has to encode the input as intact as possible, to ensure that $\mathcal{D}$ can properly reconstruct. Since the StateEncoder is designed to encode heterogeneous network flows effectively, it should be fed with as many distinct flows as possible during training, with a view to acquiring strong generalization abilities. To this end, we create a synthetic, normalized dataset with maximal variability in both packet sizes and time delays, where each packet size p_i and inter-packet delay ϕ_i in the

flows are created via:

$$p_i \sim \mathcal{U}(-1,1);\ \phi_i \sim \mathcal{U}(0,1), i \in [1,T],$$

with $\phi_1 = 0$. We assume that all the packet sizes and delays are 0-1 normalized in this dataset. p_i is sampled from $\mathcal{U}(-1,1)$ because the flow is bidirectional. We create a training set with 12,000 flows and a test set with 3,000 flows. Since a reward is given at each timestep in an episode, the StateEncoder must be able to encode a sequence with an arbitrary length. Thus, the sequence length of each mini-batch during training is randomly sampled from $[1,T]$ as shown in Alg. 2 line 5, to avoid that StateEncoder can only encode fixed-size flows. The complete training algorithm is detailed in Algorithm 2. The time complexity of a single-layer GRU in a forward pass is $O(T(h^2+hd))$, in which h represents the dimension of hidden states, d the dimension of inputs and T the number of timesteps. Although GRU and LSTM share the same computational complexity under the big O notation, the former has fewer gates and fewer hidden states to pass along time. Since encoding sequences is a time-limited task on which subsequent modules rely, we selected GRU over LSTM in our design. After training, we only preserve $\mathcal{E}$ to encode states for the adversarial actor and critic.

Algorithm 2 The training algorithm for StateEncoder

1: **Inputs:**
   $dataset := \{S_1, ..., S_n\};\ S_i := [s_{i,1}, ..., s_{i,T}]$
2: **Initialisation:**
   Denote $\mathcal{E}(\cdot)$ a two-layer GRU StateEncoder and $\mathcal{D}(\cdot)$ as the decoder with the same architecture. $\mathcal{E}$ and $\mathcal{D}$ initialized via Xavier initialization [52].
3: **while** model has not converged **do**
4: **for** S_i sampled from $dataset$ **do**
5: $S_i \leftarrow S_i[:t], t \sim \mathcal{U}(1,T)$
6: $\hat{S}_i \leftarrow \mathcal{D}(\mathcal{E}(S_i))$
7: $\mathcal{L} \leftarrow MAE(S_i, \hat{S}^i)$
8: $\mathcal{E}, \mathcal{D} \leftarrow Adam(\mathcal{L}, \mathcal{E}, \mathcal{D})$
9: **end for**
10: **end while**
11: **return** $\mathcal{E}$

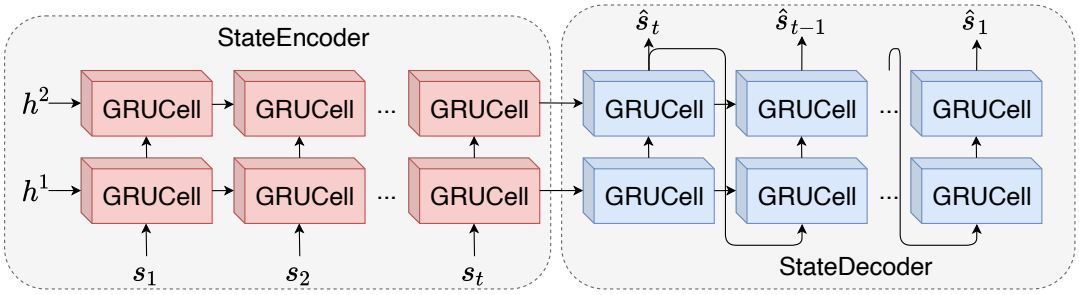


Figure 4.4: StateEncoder and associated decoder for sequence-to-sequence training. During training, StateEncoder maps an arbitrary long network flow to a fixed-size hidden representation, which is passed to StateDecoder for reconstruction.

4.3.4 Adversarial Actor & Critic

The adversarial actor aims to pick an optimal action at each timestep, such that the future rewards can be maximized. However, the action space for packet sizes is overwhelmingly large, i.e., 1,448 discrete actions for TCP and 16,384 for TLS, while the action space for time delays is infinite. Thus, we first treat both packet sizes p and time delays ϕ as continuous, and discretize them when the actor makes a choice. For example, for the TCP layer, we let the actor choose an action $(p_i, \phi_i), p_i \in [-1,1], \phi_i \in [0,1]$, and then discretize the packet size by $int(p_i \times 1{,}460)$ $byte$ and the time delay $int(\phi_i * max\_delay)$ ms, where max_delay indicates the maximum allowed delay for a packet.

We follow an actor-critic design where the actor network $\pi_\theta(\cdot)$ approximates the best action given a state, and a critic network $V_c(\cdot)$ estimates the state value. The two networks are parameterized by θ and c respectively. Specifically, the learning objective of the actor is:

$$\max \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\right]. \tag{4.3}$$

The critic network aims to approximate the state-value by minimizing the Mean Squared Error between estimated values and the discounted future rewards (R_t):

$$\min L(c) = \mathbb{E}_t[(V_c(s_t) - E_{a\sim\pi}[Q(a_t, s_t)])^2] \approx \mathbb{E}_t[(V_c(s_t) - R_t])^2]. \tag{4.4}$$

In practice, we set π_θ and V_c as MLPs and find this network structure to be effective in our task. The adversarial actor has two output units: packet size $\tilde{p}$ and inter-packet delay $\tilde{\phi}$. **To satisfy the time constraint on inter-packet delays in Eq. 4.2**, we let π_θ

output a value Δ_ϕ representing how much extra delay should be added to each packet apart from the existing delay ϕ provided by the environment, i.e., $\tilde{\phi} = \phi + \Delta_\phi$.

4.3.5 Optimization Algorithm

Optimizing RL algorithms is challenging due to high variance among trajectories and the trade-offs between exploration and exploitation that need to be achieved. A few techniques are widely used to stabilize the training process, speed up convergence, and ensure the networks are differentiable, which we also adopt in training our agent, including (1) surrogate objective function [142]; (2) reparameterization trick, and (3) parallel rollout [142]. Specifically,

1. **Surrogate Objective:** Directly optimizing Eq. 4.3 using a sampled trajectory through multiple steps of gradient ascent may lead to overwhelmingly large, and sometimes worse policy updates. Trust Region Policy Optimization (TRPO) [140] and Proximal Policy Optimization (PPO) [142] propose to use a surrogate objective function which theoretically guarantees policy improvement over stochastic gradient ascent:

$$\max \mathbb{E}_t \left[\frac{\pi_\theta (a_t \mid s_t)}{\pi_{\theta_{\text{old}}} (a_t \mid s_t)} A(s_t, a_t) \right],$$

   in which θ_{old} represents the parameters of an older version of the actor network in stochastic optimization. The surrogate objective function intuitively encourages the actions with positive advantages $A(a_t, s_t) > 0$ and discourages the opposite. We follow the PPO design to clip the ratio $I_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ (avoiding excessive update steps), and add an entropy term to encourage exploration in the action space in the final version of the objective function:

$$\begin{aligned} \max \mathbb{E}_t [\min I_t(\theta) A(s_t, a_a), clip(I_t(\theta), 1-\varepsilon, 1+\varepsilon) A(s_t, a_a))] \\ + H_{a_t \sim \pi_\theta}(a_t) \end{aligned} \tag{4.5}$$

2. **Reparametrization trick:** $\pi_\theta(\cdot)$ should approximate the distribution of actions given states but a simple MLP network only generates deterministic outputs. To overcome this issue, we assume that all the actions are sampled from a Gaussian distribution, and make π_θ generate the mean and the standard deviation of actions given states $\bar{a}_t, \sigma = \pi_\theta(s_t)$, as shown in Figure 4.3. An action then can be sampled

by:

$$a_t = \bar{a}_t + \varepsilon\sigma, \varepsilon \sim \mathcal{N}(0,1).$$

The trick ensures the actor network is differentiable, as well as generating probabilistic outputs during training.

3. **Parallel Rollout:** In order to speed up model convergence, PPO proposes to train the agent with parallel environments (N in total) where trajectories would be sampled from each environment independently with a fixed timestep T, resulting in $N \times T$ observations each time (Alg. 3 line 4). The advantage at every timestep is estimated via generalized advantage estimation [141]:

$$A_t \approx \sum_{l=0}^{\infty} (\gamma\lambda)^l [r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})],$$

in which γ is the discount factor and λ balances the bias and the variance of advantage estimation. We set $\gamma = 0.99$ and $\lambda = 0.95$. If one trajectory terminates before step T, the environment starts to generate a new one until reaching T steps and if the trajectory does not terminate after T, the advantage can still be estimated by $\hat{A}_t = \sum_{l=0}^{T} (\gamma\lambda)^l [r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})]$ (Alg. 3 line 9). Return G_t is the sum of the approximated advantage and the approximated value at each step (Alg. 3 line 10). $N \times T$ observations along with the actions, the returns and the estimated advantages are then even split into K mini-batches for stochastic optimization (Alg. 3 line 11-13).

The full training algorithm is detailed in Algorithm 3. Once Amoeba is trained, generating a new adversarial packet only requires one-step encoding by StateEncoder and the forward pass of the actor network. The critic network is not needed to generate actions.

4.3.6 Difference from Statistical Traffic Obfuscation

From an algorithmic perspective, the problem we solve may appear similar to statistical traffic obfuscation, such as TrafficMorph [65], which uses convex optimization to map the probability distribution of the packet sizes of censored traffic to that of benign traffic. Denote $X = [x_1, ..., x_n]$ as the probability distribution of censored traffic with x_i representing the probability mass of observing an i-byte packet. Similarly, let Y be the probability distribution of innocuous traffic. TrafficMorph aims to learn a

Algorithm 3 The training algorithm of Amoeba

Inputs:
λ_{split} := packet truncation overhead coefficient
λ_d := data overhead coefficient; γ := discount factor
λ_t := time delay coefficient; N := number of environments
T := the length of each rollout in the environment;
Initialisation:
Initialize π_θ and V_c via Xavier initialization [52]
Obtain StateEncoder $\mathcal{E}$ from Algorithm 2
Initialize N *Env*, each of which is provided a feature extractor $e(\cdot)$ and a pretrained classifier $f(\cdot)$
Initialize a rollout buffer with size $N \times T$
while not converged **do**
  Sample $N \times T$ observations by interacting π_θ with N *Env*
  for Each observation x_t, action a_t and reward r_t **do**
    Let $\mathbf{x}_{1:t} := \{x_1, ...x_t\}, \mathbf{a}_{1:t} := \{a_1, ...a_t\}$
    Generate the state representation
by $s_t = \mathcal{E}(\mathbf{x}_{1:t})||\mathcal{E}(\mathbf{a}_{1:t})$
    Compute $\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l[r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})]$
    Compute Return $R_t = \hat{A}_t + V_c(s_t)$
  end for
  Store each $(s_t, a_t, r_t, \hat{A}_t, R_t)$ in the rollout buffer and split them into K mini-batches $\{\mathcal{D}_1, .., \mathcal{D}_K\}$.
  Set $\pi_{\theta_{old}} \leftarrow \pi_\theta$
  for $k = 1, K$ **do**
    Compute $I_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
    Update θ via
$\nabla_\theta \frac{1}{|\mathcal{D}_k|} \sum \left[\min(I_t(\theta)\hat{A}_t, clip(I_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t) + H_{\pi_\theta}(a_t)\right]$
    Update c via $-\nabla_c \frac{1}{|\mathcal{D}_k|} \sum (V_c(s_t) - R_t)^2$
  end for
end while
return $\mathcal{E}$

transformation such that $Y = AX$, where A is a $n \times n$ weight matrix. Under the setting of TCP layer, $n = 1{,}460$. On deployment, when the algorithm observes an i-th byte packet x_i, it indexes the i-th row of A, $A_i = [a_{i,0}, a_{i,1}..., a_{i,n}]$, which is also a distribution with $a_{i,j}$ denoting the probability of morphing x_i to a j-th byte packet y_j. The algorithm randomly samples a packet size according to A_i as the morphed size. In that sense, it is similar to how RL algorithms chooses new actions given states. However, TrafficMorph is constrained in the following aspects, whereas Amoeba is not:

1. It is optimized in a way that the morphed packet distribution is similar to that of innocuous traffic, but the censor is very likely to use other features to train a classifier, such as burst behaviors, which would nullify the morphing scheme.

Amobea instead is totally censor-oriented, i.e., guided by the signals obtained from the censor classifier.

2. TrafficMorph requires the morphed packet to be larger than or equal to the source packet, so that information is not lost during morphing, whereas Amoeba allows for both truncation and padding, offering more flexibility in generating adversarial samples.

3. TrafficMorph suffers from large sample space complexity in the TCP layer because of the $n \times n$ weight matrix A, whereas Amoeba overcomes this problem by making the action space continuous and discretizing each action afterward.

In other words, Amoeba is a robust solution targeting censor classifiers directly under a more rigorous threat model.

4.4 Experiments and Results

In this section, we empirically evaluate the effectiveness of applying Amoeba on two popular types of anti-censorship systems, namely Tor network and generic TLS tunneling:

1. Tor Network is an anonymity system that utilizes relay nodes with onion protocol to conceal user location and prevent network surveillance [158]. The traffic routed inside the Tor network is encrypted by TLS and only the exit node has access to the original traffic, which is forwarded to the real destination. However, Tor is proven to be distinguishable by ML classifiers due to the fixed-size cells of the onion protocol [170].

2. V2Ray is a generic TLS tunneling tool that tunnels arbitrary TCP/UDP packets inside TLS connections [163]. Users of this type of systems usually do not demand anonymity but only seek to bypass firewalls. Thus, these tools are widely used in countries that employ censorship, such as China. We use V2Ray as the supporting tunneling system rather than its alternatives [160, 49, 83], given that it has the largest community of both maintainers and users, and it is also widely supported by 3^{rd} party clients across multiple platforms [162].

Both system types are vulnerable to ML classifiers due to the fact that the statistical features of the tunneled flows deviate from real TLS/HTTPS traffic.

4.4.1 Dataset Collection and Preprocessing

We collect two real-world datasets to evaluate our approach. Specifically, we set up a Tor client on a campus machine running Ubuntu 22.04, and a Tor bridge on a Google Cloud E2 instance running Ubuntu 22.04. The same setup is employed for a V2Ray client and V2Ray proxy server. We consider the censor sits between the Tor (or V2Ray) client and the first relay node (or V2Ray server) and distinguishes sensitive flows. To collect a realistic Tor dataset for evaluation, we crawl the landing pages of Alexa top 25,000 websites with and without Tor network respectively, obtaining 41,631 TCP flows (`Tor Dataset`). We use tshark to group packets into TCP flows and extract packet sizes and associated timestamps, where backward packet (server-to-client) sizes are represented with negative numbers to preserve the transmission direction. The same operation is repeated with and without the V2Ray tunnel, generating 43,461 TLS flows in total, named `V2Ray Dataset`. Different from the `Tor Dataset`, we utilize tshark to group packets into TLS flows, and extract TLS record sizes and timestamps. For this dataset, we consider the censor conducts deep packet inspection up to the TLS layer and extracts features from TLS flows instead of TCP flows. The maximal TLS record is 16 KB, i.e., Amoeba is required to explore a much larger action space.

Ethical Statement: All the data is collected in a controlled environment without real users attempting to evade censorship. There are no ethical implications in this work.

Each dataset is separated into a *clf_train_set* (40%), an *attack_train _set* (40%), a *validation_set* (10%) and a *test_set* (10%). We use the *clf_train_set* to train censoring classifiers, which are then evaluated on the *test_set*. After that, each trained censoring classifier is deployed in the Environment in Figure 4.3 to generate rewards. We use the *attack_train_set* to train Amoeba instead of using *clf_train_set*, because the attacker may have no access to the dataset owned by the censor in practice. The *validation_set* is utilized to tune the hyperparameters of Amoeba. After training, Amoeba and the benchmark algorithms are evaluated on the *test_set* against the trained censoring classifiers.

4.4.2 Censoring Classifier Selection

We adopt a range of state-of-the-art traffic analysis models as censoring classifiers:

Deep Fingerprinting (DF) [149] is a state-of-the-art CNN-based deep learning model that automatically extracts features from raw network flows and performs WF.

Stacked Denoising Autoencoder (SDAE) [133] follows a MLP-based encoder-decoder architecture to extract latent features from network flows directly for WF.

LSTM [133] is a multi-layer recurrent neural network that takes arbitrary long network flows as input to perform WF. LSTM is designed to learn long-term dependencies, and therefore can better interpret timeseries data such as consecutive packets.

CUMUL [119] separates different classes of data by using SVM with a radial basis function (RBF) kernel to find the hyperplane that maximizes the margin between classes.

The original versions of DF, SDAE and LSTM are fed with packet directions only (i.e., $(-1,1)$), and vanilla CUMUL leverages the cumulative representation of network traces without timing features. For consistence, we tailor these classifiers to utilize the flow representation in Section 4.2 as input. That said, these classifiers do not need an external feature extractor (i.e., $e(\cdot)$ is an identical function).

Tree-based models [8]: Traditional ML models, such as DT and Random Forest (RF), exhibit promising performance in detecting multimedia tunneling protocols. Tree-based approaches possess better interpretability compared to DL models, since the decision-making process can be visualized as a set of tree-like rules. We follow [8] to extract 166 features from each network flow, covering bidirectional packet/timing statistics, burst behaviors, percentile features and flow-level information, and use them to train the DT/RF.

4.4.3 Benchmark Algorithms

We choose three advanced *white-box* adversarial attacks as benchmark algorithms for our evaluation:

CW Attack [19] uses projected gradient descent to find minimal perturbations on the inputs, while maximizing the probability of the inputs being misclassified. The CW attack iteratively queries the classifier for a single input, until an adversarial sample is found.

NIDSGAN [194] regards the censoring classifier as the discriminator in a GAN architecture, and trains a generator to learn minimal perturbation patterns needed to fool the discriminator. The generator directly produces adversarial samples given inputs, without needing iterative updates.

BAP [114] also aims at training generator-like NNs, but is more flexible in allowing inserting packets into a given flow, i.e., the length of an adversarial sample is not always

identical to the input, posing larger difficulties for censoring classifiers.

We do not consider black-box benchmark algorithms [3, 14, 21], since existing ones are infeasible under our threat model where feature extraction is performed (see Figure 4.1). We implement the NN-based classifiers, CW attack, NIDSGAN, BAP and Amoeba in Pytorch [121], and import the rest of the classifiers from the scikit-learn package in Python. We use the Adam algorithm [82] with default settings and learning rate 5×10^{-4} to optimize the NNs. Detailed hyperparameter selection is documented in Section 4.4.9.

4.4.4 Evaluation Metrics

To evaluate the effectiveness of our solution against ML classifiers, we measure their accuracy and F1 score metrics, which are based on True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN):

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}, F1 = 2\frac{precision \times recall}{precision + recall},$$

where $precision = TP/(TP+FP)$ and $recall = TP/(TP+FN)$. Accuracy indicates the proportion of samples correctly classified, and F1 score computes the harmonic mean between precision and recall. The former represents how likely an algorithm would give true alarms, and the latter indicates how sensitive an algorithm is towards positive samples.

We use also use another three metrics to evaluate Amoeba, namely Attack Success Rate (ASR), i.e., the percentage of adversarial samples being misclassified, data overhead and time overhead:

$$data\ overhead = \frac{padding}{original\ payload + padding}$$
$$time\ overhead = \frac{delays}{delays + total\ transmission\ time}$$

in which total transmission time is the time difference between the last and first packet in a flow.

4.4.5 Performance of StateEncoder

The performance of our StateEncoder impacts the adversarial actor in terms of understanding and interpreting the actual state at each timestep. There is no simple method to

evaluate StateEncoder alone, since the hidden representations are in high-dimensional space and information loss during encoding is intractable. Nevertheless, we can obtain an upper bound of the information loss by examining the Noromalized Mean Absolute Errors (NMAE) of the Seq2Seq model (consisting of StateEncoder and StateDecoder):

$$\text{NMAE}(S,\hat{S}) = \frac{1}{T \times N} \sum_{n=1}^{N} \sum_{t=1}^{T} \frac{|s_t^n - \hat{s}_t^n|}{s_t^n}.$$

s_t^n is the packet t in flow n and $\hat{s}_t^n$ the reconstructed packet. We show the NMAE of the Seq2Seq model composed by StateEncoder and StateDecoder in Figure 4.5, which helps us understand to what extent the encoded hidden representations can sustain the information of the original flows. It can be seen that the NMAE of flow reconstruction would increase as the flow length increases, although this is not obvious when the flows have less than 40 packets and the average NMAE in $[1,40]$ is around 9%. When the flow length is longer than 40, the NMAE gradually increases from 9% to 19% with an outlier of 28.95% when the flow length equals 48. An intuitive explanation of the NMAE in our case is that, for example, when a flow has 60 packets, each packet size p is encoded as a value between $p \pm 0.19p$ in the hidden representation. Although not perfect, these experiments demonstrate that this level of precision is actually adequate for Amoeba to learn an effective policy. Note that 90.5% of Tor flows in the dataset have less than 60 packets. To ensure that long flows can be encoded properly in practice, an engineering solution is splitting long flows before a pre-set threshold or using deeper networks to encode flows.

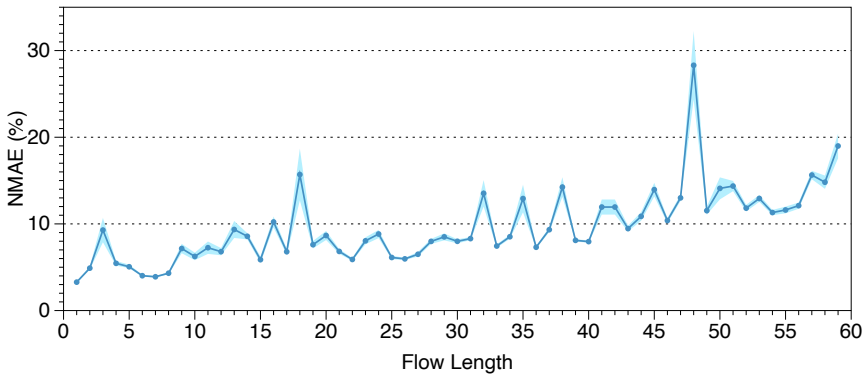


Figure 4.5: Normalized reconstruction errors (with error bars) of StateEncoder + StateDecoder.

4.4.6 Obfuscation Efficacy

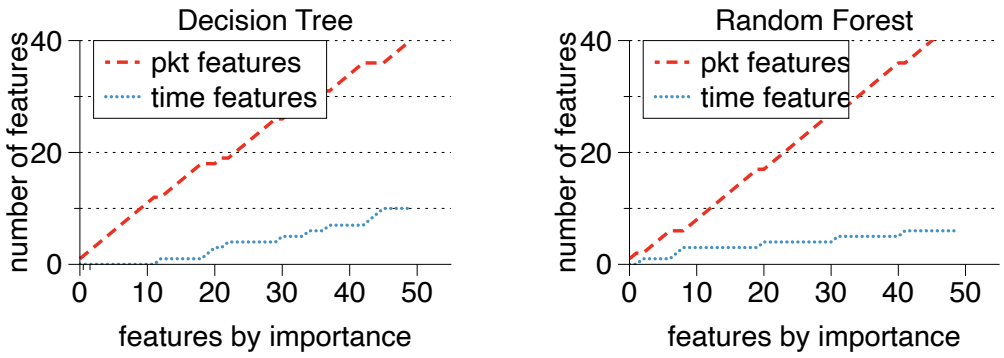
Table 4.1 presents the performance of each classifier detecting Tor and V2Ray traffic respectively, as well as the efficacy of adversarial attacks targeting these classifiers. In the absence of adversarial manipulations, the selected classifiers yield almost perfect accuracy and F1 scores (third column) as expected on the *test_set*, since both anti-censorship systems generate unique statistical patterns during communications. For example, when observed on the TCP layer, Tor traffic mostly consists of packets of (multiples of) 536 bytes, which is the size of an encapsulated onion cell, giving ML classifiers high confidence to detect. V2Ray's TLS-tunneled flows can be differentiated from HTTPS flows, because for HTTPS, once the TLS connection is established, the inner communications are all HTTP requests/responses; while for TLS-tunneled flows, the inner communications may involve a TLS handshake between browser and web server. This TLS-in-TLS pattern would not be witnessed in normal browsing traffic without a tunnel, which gives ML classifiers opportunities to learn the discrepancies based on the statistical features.

On the other hand, the selected white-box adversarial attacks are effective in generating adversarial features of network flows. It is not surprising that the CW attack can reach $\sim$92%ASR on average with $\sim$37% data and $\sim$18% time overhead (fourth column). This attack explores misleading perturbations by leveraging the weights and gradients of the censoring classifiers, and iteratively optimizes an adversarial sample for each input (network flow). However, the practicality of CW is questionable in the networking domain, since it requires 1) a complete flow as input; and 2) multiple rounds of queries to the censoring classifiers until a legitimate adversarial flow is found.

NIDSGAN and BAP overcome the second issue by training a neural network to generate perturbations for arbitrary inputs in advance, and in the deployment stage adversarial flows can be generated in one go. NIDSGAN has however, limited flexibility, since the length of adversarial flows must be equal to the length of input flows. If the censoring classifiers are able to learn directional features from sensitive flows, simply adding perturbations to each packet without inserting new packets would not change directional features, potentially leading to the failure of NIDSGAN. BAP utilizes a dedicated NN to learn at which positions in a flow if new packets should be inserted, as an approach to disturb directional features. Based on the results in Table 4.1, we remark that NIDSGAN and BAP have their own merits, but can also be unstable when

Dataset	Attack Threat Model Censoring Alg.	None *None* F1	None *None* Accuracy	C&W *white-box* ASR (%)	C&W *white-box* DO (%)	C&W *white-box* TO (%)	NIDSGAN *white-box* ASR (%)	NIDSGAN *white-box* DO (%)	NIDSGAN *white-box* TO (%)	BAP *white-box* ASR (%)	BAP *white-box* DO (%)	BAP *white-box* TO (%)	Amoeba *black-box* ASR (%)	Amoeba *black-box* DO (%)	Amoeba *black-box* TO (%)
Tor	SDAE	0.99	0.99	88.34	21.60	0.00	22.67	21.51	12.2	57.20	40.96	14.10	89.0	64.8	3.72
	DF	0.99	0.99	97.88	26.68	23.94	94.13	31.8	7.28	89.46	35.95	12.49	87.5	59.0	3.79
	LSTM	0.99	0.99	90.49	86.64	8.37	2.86	1.75*	1.24*	93.86	38.88	18.65	98.2	58.1	2.26
	DT	1.00	1.00										96.5	29.0	5.69
	RF	1.00	1.00		N/A			N/A			N/A		92.0	29.1	3.73
	CUMUL	0.99	0.99										93.0	44.5	3.55
V2ray	SDAE	0.99	0.99	99.54	25.24	24.88	26.04	22.99	20.23	79.92	26.76	5.99	93.8	43.2	5.49
	DF	0.99	0.99	84.33	49.31	49.89	95.44	22.9	9.17	62.57	25.13	0.00	96.8	46.1	7.45
	LSTM	0.99	0.99	96.61	16.10	2.51	93.32	38.44	15.23	91.56	16.98	29.78	89.2	7.73	1.46
	DT	1.00	1.00										97.2	40.2	8.44
	RF	1.00	1.00		N/A			N/A			N/A		99.4	53.97	8.30
	CUMUL	0.99	0.99										96.4	51.6	10.48

Table 4.1: Performance of different classifiers in detecting Tor flows without attack; performance of Amoeba in crafting adversarial flows. For comparison, we also show the Attack Success Rate (ASR) of CW, NIDSGAN, and BAP attacks under different threat models (DO – data overhead; TO – time overhead). DO/TO with * indicate estimated values, since the attacking algorithms add perturbations in the feature space. The estimated values reported represent the maximal perturbation allowed for data and timing features respectively.



(a) important features ranked by DT. (b) important features ranked by RF.

Figure 4.6: Difference between packet and timing features among top-50 important ones used by DT/RF on the `V2Ray dataset`. x-axis arranges features by importance; y-axis shows the number of specific type of features in top n.

confronting different NN architectures. Since both methods generate perturbations for an entire flow, it would be difficult to learn how the changes of a small number of packets in a flow would impact the final classification results. In contrast, Amoeba is designed to observe the classification result upon every new packet in an adversarial flow, which provides fine-grained information to infer the decision boundary of classifiers.

Table 4.1 reveals that our proposed **Amoeba reaches ∼94% ASR on average against multiple types of classifiers**, being capable of exploring the decision boundary of a classifier even if they are not NN-based (and thus offer no gradient information which is required by existing attacks), including DT, RF and SVM/CUMUL. Compared with white-box methods, Amoeba **follows a much stricter threat model where feature engineering and model architecture are invisible, and it is also more stable against different classifiers**. The data overhead of the adversarial flows are in a similar range, between 43.2–64.8%, except for adversarial samples against DT/RF on `Tor Dataset` (where it is lower, yet a comparison with gradient-based methods infeasible) and those for LSTM on the `V2Ray Dataset`. The time overhead of adversarial flows is consistently $<$10.5%.

4.4.7 Impact of Network Environment

A shared observation on the results with both datasets is that adversarial flows possess greater data overhead than time overhead. The reason is that censoring classifiers leverage more on packet features than on timing features to make decisions. We vi-

sualize important features used by DT and RF in Figure 4.6, where the x-axis lists the top 50 important features in descending order, and the y-axis shows the number of packet and timing features respectively. Observe that packet features in general are overwhelmingly more important than timing features. Practically, network flows may suffer different degree of congestion depending on route and time, while packet/record sizes in a flow are purely determined by the client and the server, thus more reliable for the censoring classifiers. As a result, Amoeba makes more efforts to reshape sizes than timings.

Train/Test Pkt Drop Rate	0%	2.5%	5%	7.5%	10%
0%	**87.5**	-8.2	-8.1	-6.4	-7.4
2.5%	-0	**88.8**	-0	-0.2	-0
5%	-2.0	-1.6	**94.2**	-1.2	-1.2
7.5%	+0.8	-1.8	-1.4	**94.2**	-0.4
10%	-1.2	+0.6	-1.2	-0.8	**92.0**

Table 4.2: Performance (difference) [%] of Amoeba under different network environments.

In a more extreme setting where not only network congestion exists but packets are also dropped due to overwhelming volume of traffic in the network, packet retransmission would be needed to tackle data loss. To evaluate the impact of different packet drop rates on the performance of Amoeba, we additionally collect `Tor Datasets` multiple times where we enforce packet drop rates for bidirectional traffic between 0% and 10%. The same data preprocessing/split convention is followed. We train Amoeba against DF on the *attack_train_set*s collected under different packet drop rates and then evaluate it on different *test_set*s. The results are shown in Table 4.2, in which the numbers on the columns represent the packet drop rates under which the training sets are collected, and those on the rows indicate the packet drop rates under which the test sets are collected. The ASR [%] of Amoeba trained and tested under the same environment are shown on the diagonal of the table in bold, and the rest of the numbers indicate the performance difference in % when cross-evaluating Amoeba in different environments.

Amoeba trained with data experiencing 2.5%–10% packet drop rates exhibits particularly robust performance when perturbing network flows collected from other environments (2^{nd} to 5^{th} rows). However, if the training set does not incorporate retransmitted packets, it would be more difficult for Amoeba to perturb network flows

collected with non-zero packet drop rates (1st row). This is not surprising since the dataset collected with 0% drop rate is less heterogeneous than that with non-zero drop rates. This set of results reveals that network environment is an important factor when collecting network flows, and if the dataset can reflect the heterogeneity of the network, then Amoeba is less sensitive to changes in the network environment.

4.4.7.1 What is the cost of using Amoeba and can that be reduced?

Effectiveness against CUMUL/DT/RF aside (where the benchmarks considered don't work), Amoeba's ASR is higher than or on par with that of *white-box* attacks at the cost of a) higher data overhead, and b) 2 to 10 times more interactions with the censoring classifiers (see Table 4.3). The reason is two-fold: 1) Amoeba is a black-box algorithm and therefore requires more queries by nature; 2) Given a flow S, Amoeba is designed to interact with the classifier at least $|S|$ times and observe associated rewards, whereas BAP only needs to interact once in a training epoch.

Censoring clf	NIDSGAN	BAP	Amoeba
SDAE	103,614	162,822	302,880
DF	88,812	29,604	100,960
LSTM	59,208	29,604	289,920
CUMUL	N/A	N/A	100,320
DT	N/A	N/A	235,680
RF	N/A	N/A	257,760

Table 4.3: The number of queries needed for NIDSGAN, BAP and Amoeba until convergence on `Tor Dataset` (Results on `V2Ray` are at the same scale).

However, in practice it may not be always possible to perform countless queries to censoring classifiers. We therefore attempt to reduce the number of interactions needed by randomly masking the rewards when training Amoeba. In the vanilla version of the training algorithm, Amoeba expects to receive a part of the reward r_{adv} for each subsequence of the generated flows, with 1 denoting good and 0 for sensitive. We mask r_{adv} with a probability p_{mask} from 0% to 90% during training, and the masked r_{adv} is considered to be 0.5 instead, representing unknown feedback. Amoeba is trained with 300,000 timesteps, and the actual number of queries would be $300,000 \times (1 - p_{mask})$. The best ASR against various classifiers with different p_{mask} applied is shown in Figure 4.7. Due to ambiguous feedback during training, Amoeba experiences large fluctuations and would be sometimes trapped in a local optimum, resulting in an ASR drop. Nevertheless, the mean ASR only decreases from 93% to 74.4% when p_{mask}

increases from 0% to 90%, proving the efficacy of Amoeba even if the rewards are highly noisy.

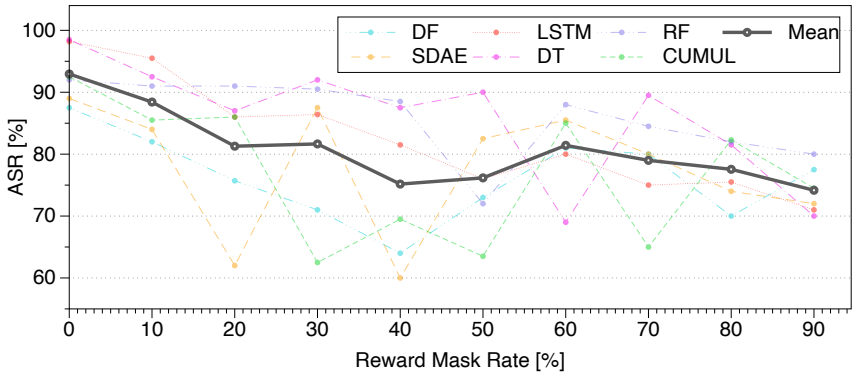


Figure 4.7: The impact of reward mask rate on ASR.

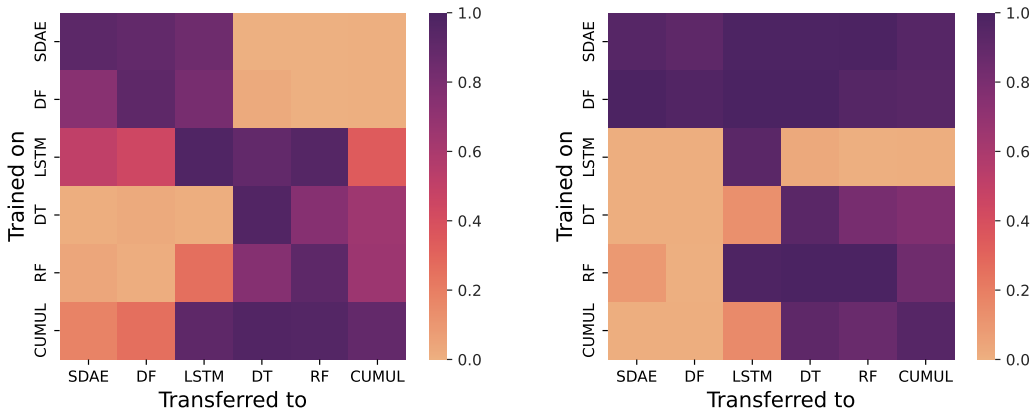
4.4.7.2 Are adversarial samples transferable?

Here we investigate whether adversarial flows generated by Amoeba against one classifier can also deceive other models without retraining. To this end, we store all the adversarial samples obtained from each model and feed them to the rest of the classifiers for both datasets. We plot success rates as a heat map in Figure 4.8, where we find that **adversarial flows targeting similar architectures are transferable with high success rate**, such as SDAE and DF, and DT and RF, meaning that these pairs of classifiers are likely to learn a similar decision boundary.

The adversarial samples targeting LSTM on the `V2Ray Dataset` are exceptional with only 7.73% data overhead on average. It is likely that Amoeba uncovers a unique and efficient strategy to attack sequential models on this dataset, but cannot be easily generalized to other censoring classifiers.

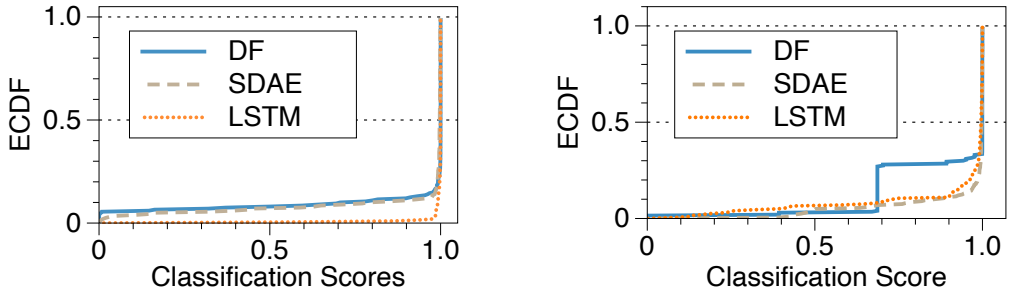
4.4.8 How is the Quality of Adversarial Samples?

When attacking NN-based classifiers, the architecture and the weights are invisible to Amoeba, but our algorithm can still explore the decision boundary effectively and find qualified adversarial flows. Figure 4.9 plots the Empirical Cumulative Distribution Function (ECDF) of the classification scores with respect to adversarial flows against different NN-based classifiers on both datasets, where the majority of the scores are close to 1 (benign) rather than 0.5. This means that during training, Amoeba does not



(a) Results on `Tor Dataset` (b) Results on `V2Ray Dataset`

Figure 4.8: Transferability of adversarial flows on both datasets. Adversarial examples generated by Amoeba against each model on the y-axis and tested on other models on the x-axis. Color of each cell represents ASR.



(a) Results on `Tor Dataset` (b) Results on `V2Ray Dataset`

Figure 4.9: ECDF of classification scores wrt. adversarial flows against different NN-based classifiers. Left plot shows the scores obtained on `Tor`, right for `V2Ray`.

choose actions randomly in the action space without a proper strategy, but **can understand where the decision boundary lies in the black box and generates adversarial flows just as innocuous traffic**, from the perspective of ML-based classifiers.

4.4.9 Hyperparameter Selection

Amoeba is a complex model with a range of hyperparameters and it would be difficult to conduct exhaustive search in the full hyperparameter space. To select hyperparameters for Amoeba, We first choose the search space by our experience and build the model in a block-by-block fashion. StateEncoder requires pretraining and therefore the associated hyperparameters are decided initially, followed by the architecture of actor and critic. λ_{data}, λ_{time} and λ_{split} plays an important role in the reward function

and largely determines the final ASR and overhead rates. We notice that Amoeba is not sensitive to λ_{time} but the results may vary greatly given different λ_{data}. Since `Tor Dataset` and `V2ray Dataset` have different largest transmission units (TCP segement and TLS record), the optimal λ_{data} are 0.2 and 2 respectively. λ_{split} determines how frequently the packet should be truncated. Our experimental results indicate that when $\lambda_{split} > 0.1$, Amoeba would tend not to truncate packets at all. If directional features need to be disturbed, λ_{split} should be set smaller than 0.1. We set $\lambda_{split} = 0.05$ eventually so that packet truncation would occur but is not so frequently that exceeds the capability of StateEncoder. We choose the set of hyperparameters in Table 4.4 which is good enough to provide high ASR and acceptable overhead rates, but there may exist better selections.

Hyperparameter	Search Space	Value
optimizer	Adam, SGD, RMSProp	Adam
learning rate	[0.0001, 0.01]	0.0005
λ_{split}	[0.01, 0.1]	0.05
λ_{time}	[0.1, 2]	0.2
λ_{data} for `Tor`	[0.1, 5]	0.2
λ_{data} for `V2ray`	[0.1, 5]	2
Actor/Critic layer number	[2, 5]	4
Actor/Critic layer dim	[32, 1024]	512→ 64→ 32 →output
StateEncoder architecture	[GRU, LSTM]	GRU
StateEncoder dim	[256, 1024]	256
StateEncoder layer	[1, 4]	2

Table 4.4: Hyperparameter selection for Amoeba.

4.4.10 Limitations

Having demonstrated the efficacy of Amoeba, here we discuss the feasibility of deploying our solution in practice. Integrating the algorithm in the transport layer and morphing each packet at line speed is the most ideal way of usage. We run a single-step action inference of our model on a NVIDIA K80 GPU for 10 times and obtain an average inference time of 0.370 ± 0.001 ms, which despite small, may be considered non-negligible in the sequence generation process.

To better understand this, Figure 4.10 shows the density and the box plot of the inter-packet delays between *every two consecutive packets in the same network di-*

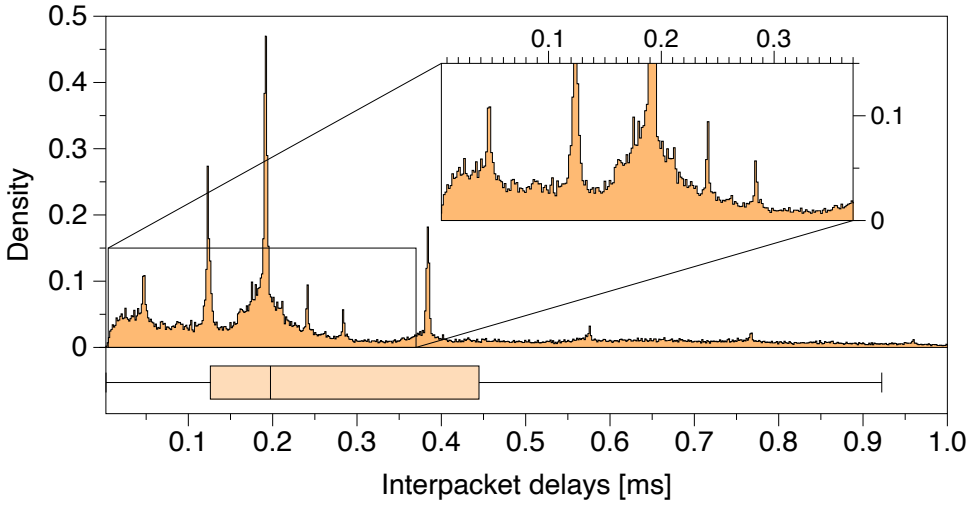


Figure 4.10: The distribution (upper) and box-plot (bottom) of the inter-packet delays between every two consecutive packets in the same network direction.

rection in our dataset, where 67.5% of the inter-packet delays are less than 0.37 ms. The time needed for inference would challenge the deployment of the algorithm in an online manner. However, it is still possible to utilize it once Amoeba is well-trained against a censoring classifier. Specifically, we can generate a number of adversarial flow profiles, which only consist of packet sizes and inter-packet delays without real payloads. The profiles would be saved in a database and synchronized with both client and server proxies. During communications, both parties embed actual payload into flows exactly as the flow profiles instruct. If one end has no payload in the buffer but the profile indicates a packet should be sent, then a packet with dummy payload would be transmitted to align with the pre-generated adversarial flow. Although this approach may further increase data overhead, since the flow profiles are not generated based on the current states, it ensures that ML-supported censorship can be successfully circumvented. Besides, more engineering efforts, such as matching optimal adversarial flow profiles with IP addresses, can be explored for better user experience.

4.5 Summary

Embedding covert streams into a cover channel is a common approach to circumventing Internet censorship, due to censors' inability to examine encrypted information in otherwise permitted protocols (Skype, HTTPS, etc.) at intermediate links. However, recent advances in machine learning enable detecting a range of anti-censorship systems by learning distinct statistical patterns hidden in traffic flows. Therefore, de-

signing obfuscation solutions able to generate traffic that is statistically similar to innocuous network activity, in order to deceive ML-based classifiers at line speed, is difficult.

In this section, we formulated a practical adversarial attack strategy against flow classifiers as a method for circumventing censorship. Specifically, we cast the problem of finding adversarial flows that will be misclassified as a sequence generation task, which we solve with Amoeba, a novel reinforcement learning algorithm that we design. Amoeba works by interacting with censoring classifiers *without any knowledge of their model structure*, but by crafting packets and observing the classifiers' decisions, in order to guide the sequence generation process. Our experiments using data collected from two popular anti-censorship systems demonstrated that Amoeba can effectively shape adversarial flows that have on average 94% attack success rate against a range of ML algorithms. In addition, we showed that these adversarial flows are robust in different network environments and possess transferability across various ML models, meaning that once trained against one, our agent can subvert other censoring classifiers without retraining. Finally, we made the case for Amoeba's deployment as a transport layer extension. In the next chapter, we shift our attention to the security problems facing recipients and propose a novel DL-based NIDS.

Chapter 5

Network-Based Intrusion Detection via Deep Learning

The two ends of network communications as illustrated in Figure 1.1 encounter a common challenge, namely discerning the purpose of the communications. Senders are in the position to initiate connections, and thereby primarily concerned with determining whether the network traffic contains PII without obtaining user consent. On the other hand, recipients must carefully examine incoming requests in an effort to filter out malicious network activities. Owing to the vast volume of traffic received relative to the traffic originating from a single device, manual inspection on a flow-by-flow basis becomes impractical. Specifically, the volume of illicit network traffic continues to grown dramatically, with the number of high-profile attacks including DDoS, botnet, and ransomware rising by over 45% annually [152], and the losses incurred expected to exceed 6 trillion US dollars in 2021 [166]. Traditional NIDS largely apply finite rules, preset by human experts, to detect anomalies. This approach lacks flexibility and is often prone to subversion [12]. ML is increasingly used to detect cyber intrusions, due to its ability to discover complex statistical patterns hidden in data streams, which can aid in discriminating anomalies based on feature differences [15].

ML is a powerful tool, yet adopting it meaningfully for security purposes is not straightforward. ML techniques used in areas including imaging and natural language processing have been directly applied to NID (e.g., [97]), without adequate analysis of their suitability for this task. For instance, reconstruction-based algorithms like autoencoders were originally designed to learn to recreate benign samples that contain similar patterns, e.g., the same object type in images [180]. However, when deployed for intrusion detection, whether an autoencoder is able to reconstruct heterogeneous

benign traffic originating from various applications is rarely discussed [107]. Secondly, widely-used evaluation methodologies involve training and testing NID models on a single dataset, collected in the same controlled environment. This makes it difficult to assess if the trained models can truly generalize to previously unseen traffic mixes [151]. Moreover, detecting high-volume attacks promptly, before a target system becomes overloaded and unable to thwart malicious traffic with potential to cause severe damage following early system compromise, is difficult. This capability is however critical to the availability and revenue of online businesses [28].

In this chapter, we address the above challenges and propose **NetSentry**, a novel DL-based NIDS that reliably detects a range of malicious traffics with similar patterns, indicative of incipient high-impact network attacks.

5.1 Threat Model and Problem Formulation

We start from the key observation that in practice traffic flows shall not be considered in isolation, either as benign or malicious. There exist important temporal correlations among different cyberattacks, especially those with high-impact, which rarely occur independently. For instance, assume that an adversary has zero knowledge of a potentially vulnerable target. Conducting a successful webshell injection attack has at least two prerequisites:

1. port scanning against the target, so as to uncover that it runs a web serve;
2. web API enumerating, to verify if file upload is allowed.

That is, the attacker must follow a certain sequence of actions (each an attack itself), which would create distinct network traces at various stages. We remark that essential correlations among different kinds of network attacks have not been explored before, but are potentially useful to design a reliable NIDS.

Hence, we decompose network attacks from the perspective of an active adversary, and summarize them into different attack chains. These typically start with gathering information about a target and conclude when a specific technical goal was achieved. We consider three key attack chains, namely botnet, web intrusion, and ransomware, revealing that they are supported by a similar methodology. While our attack chain view may appear on the surface related to earlier Botnet infection modeling, where attack stages are fingerprinted [60], our modeling approach and subsequent NIDS design are fundamentally different. This is because *the attack chains we consider aim to*

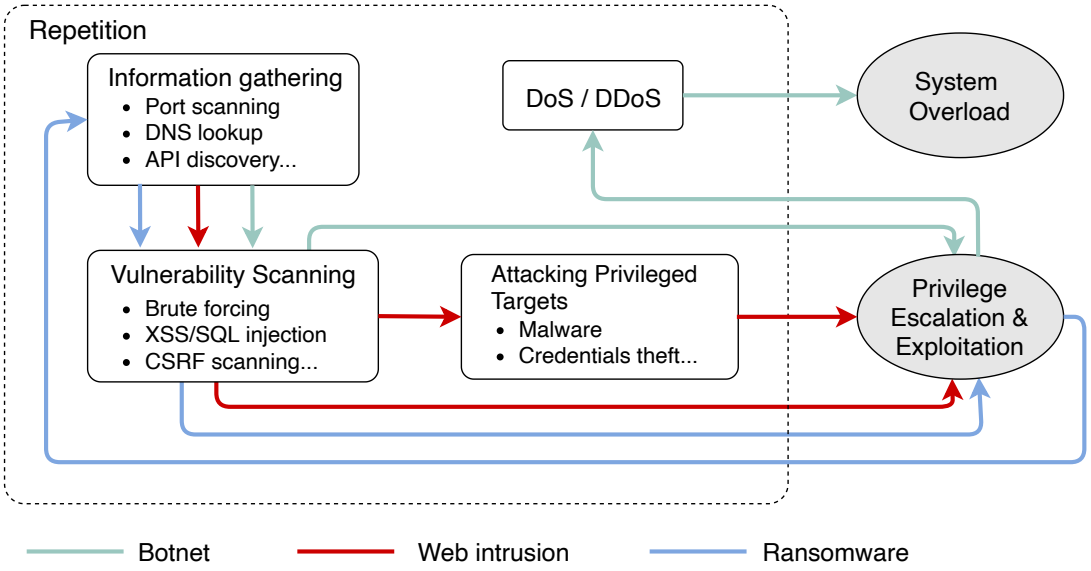


Figure 5.1: Attack chains employed by large-scale high-impact threats, e.g., botnets, web intrusion, and ransomware. Observe that similar steps are repeated by all, which **NetSentry** exploits for detection.

reveal the common stages shared by different large-scale cyberattacks, so as to impede a specific range of cyber intrusions by interrupting any of these early stages.

5.1.1 Attack Chain Analysis

We particularly focus on two network attack goals, which bring severe damage to target systems. The first is to obtain system privileges temporarily or permanently, by exploiting various security flaws. The second is to overload the system by occupying all its resources. Instead of looking at each attack type individually, we investigate what processes, i.e., attack chains, an adversary must follow to achieve any of these goals, when having zero knowledge about a target. We consider three unique attack chains that are specific to botnet, web intrusion, and respectively ransomware, as shown in Figure 5.1.[1]

Botnet/Mirai. Botnets are collections of Internet-accessible devices hijacked by an attacker, and are usually employed to carry out large-scale, high-impact DDoS attacks. Mirai is one of the most notorious instances in recent years, with ∼600,000 devices infected at its peak [5]. Subsequent Mirai variants expand the attack surface to SSH, HTTPS, FTP, etc., but inherit the methodology of the canonical version.

Mirai follows a chain-like methodology that entails *information gathering* → *vul-*

[1]We note that certain sophisticated attacks may have longer attack chains that expand to application level [74]. However, by detecting their early stages, their later exploitation actions can be prevented.

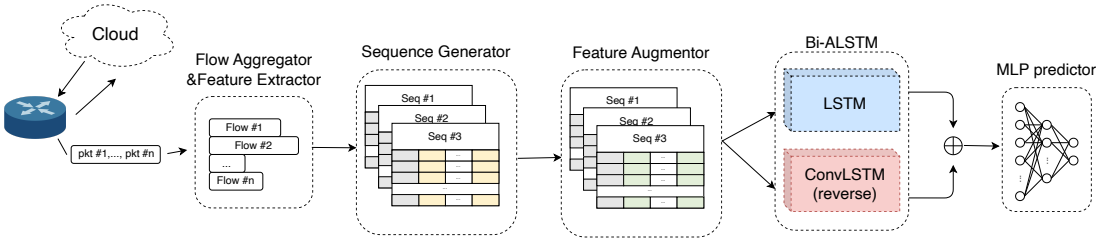


Figure 5.2: NetSentry architecture. Feature Augmentor only applied during training. $\oplus$ is fusion operation detailed in Eq. 5.13.

nerability scanning $\rightarrow$ *privilege escalation* $\rightarrow$ *DDoS*. Specifically, (1) TCP SYN packets are sent towards the entire IPv4 address space, on ports 23 and 2323 (Telnet); (2) after identifying potential victims, Mirai attempts to bruteforce the victims' credentials using a dictionary – this process is deemed as vulnerability scanning; (3) upon successful login, a victim's IP and credentials are forwarded to a report server that infects the victim with malicious code; (4) newly infected devices become members of the botnet, and either participate in victim discovery or DDoS attacks [5].

Web intrusion. Web applications integrate a technology stack that includes storage, web engines, Operating Systems (OSs), and communications. Hence, various vulnerabilities are often exploited. Web intrusion can be jointly modeled with the Intrusion Kill Chain [72] and the OWASP Web Penetration Guideline [135], where the former outlines a general intrusion process, while the latter provides specific attack vectors.

The attack process entails *information gathering* $\rightarrow$ *vulnerability scanning* $\rightarrow$ *attacking privileged targets* $\rightarrow$ *exploitation*; or *information gathering* $\rightarrow$ *vulnerability scanning* $\rightarrow$ *privilege escalation*. Web intrusion resembles botnet and ransomware in terms of vulnerability discovering approach, but differs at the later stage, since exploitation is always target-specific, e.g., a XSS attack targets web app users, while SQL injection targets Web APIs. Our focus is on the early stage when the attacker tries to breach a trusted boundary, since later steps occur at system level and are invisible to a NIDS.

Ransomware/WannaCry. Ransomware is a relatively new type of threat that blocks user access to their private data until a ransom is paid to the attacker. For instance, by exploiting EternalBlue [169], WannaCry gains system access via the Server Message Block (SMB) protocol on Windows systems, encrypts user data, and spreads itself to other hosts [79]. The attack chain of WannaCry follows a loop consisting of: *information gathering* $\rightarrow$ *vulnerability scanning* $\rightarrow$ *privilege escalation & exploitation* $\rightarrow$ *information gathering*.

We focus on the procedure ransomware uses to discover and control new victims (instead of the file encryption applied), as the attack is conducted via the network. WannaCry employs repeated TCP scanning on port 445 (serving the SMB protocol) [79]. Targets are further fingerprinted and remote access is achieved by injecting code via a crafted packet, which would be mishandled by SMBv1. WannaCry then encrypts the data on the victim machine and discovers other vulnerable targets.

Our attack chains analysis revealed that information gathering, vulnerability scanning and DoS are applied across various types of attacks and share the same semantic. Latent network attacks, such as malware downloading, code injection, Cross-Site Request Forgery (CSRF), and other zero-day attacks, which can obtain system privileges and proceed to exploitation, always follow massive vulnerability scanning, since this is the most efficient way to discover weak entry points. A common argument is that zero-day attacks are heterogeneous, which poses difficulties to any detection logic. However, we argue that *as long as automated activities can be recognized in time, the subsequent zero-day attacks can be blocked,* in order to minimize the chances that attackers may uncover weaknesses and compromise a system. As such, we keep the scope of our detection targets narrow, yet well-directed, as suggested in [151]. With this in mind, we design NetSentry, a NIDS that effectively tackles cyber intrusion by detecting risks at an *early stage*. This also applies to tactics that deviate from the standard attack chains described, as long as they incorporate any common stages to achieve the same end goal.

We maintain that NID, especially of automated attacks, should be treated as a time-sensitive task. Here we consider 'time-sensitive' those network intrusions exhibiting temporal correlations among *consecutive traffic flows*, which could potentially exert substantial impact on the decision-making process. This is because a single traffic flow, whose features are extracted as a datum, may not fully reflect the intention of the communications. A straightforward example is a TCP flow encapsulating a complete HTTP request. Assume the flow is terminated quickly after the server responds. Without looking at previous and subsequent traffic, it is impossible to assert whether the flow was initiated by a legitimate user or by DoS tools. Conversely, if we observe a series of statistically similar communications between a pair of hosts, the confidence of classifying them as malicious becomes higher. Network attacks generated with the same tool usually serve the same purpose. Although they would encapsulate different payloads in consecutive flows for obfuscation or fuzzing purposes, this difference is invisible to a NIDS that only has access to protocol headers and timing information.

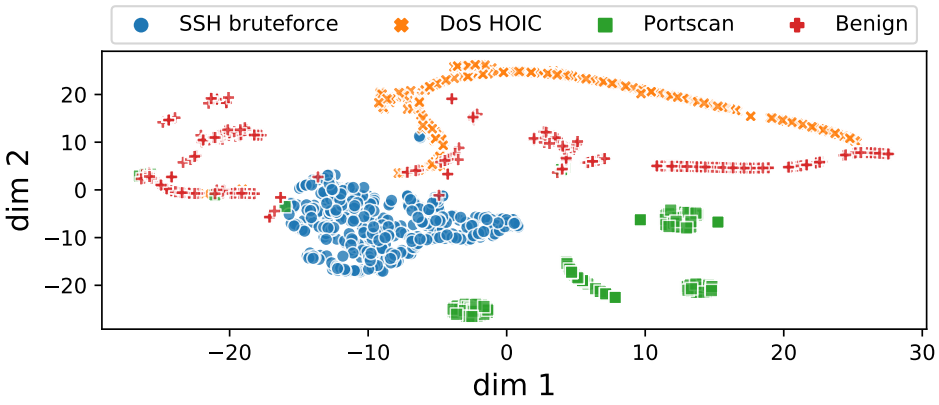


Figure 5.3: T-distributed stochastic neighbor embedding (t-SNE) of different network attacks and benign traffic.

Thus, we leverage sequential neural models in NetSentry to learn such similarity of successive flows generated with automated tools.

Since we focus on automated attacks at the early stage of intrusion, unsupervised detection solutions are unsuitable. This is because they rely on uncovering outliers that noticeably deviate from benign traffic represented by dense area (inliers) in feature spaces. However, potentially large-volume automated attacks can statistically form new dense areas as well, which makes their identification impossible. To exemplify this, in Figure 5.3 we plot the t-distributed stochastic neighbor embedding (t-SNE) [103] of a subset of traffic flows in the CSE-CIC-IDS2018 dataset [40], which we use in our study. t-SNE is commonly used to allow the visualization of high-dimensional data via non-linear dimensionality reduction, e.g., to a 2-dimensional space [192]. In our case we set the perplexity to 40 when computing the t-SNE. Note the clusters corresponding to most types of attacks can be distinguished clearly.

As explained above, detecting automated cyberattacks is a time-sensitive task and hinges on temporal correlations between network flows. Let $X := \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_\alpha\}$ and $Y := \{y_1, y_2, ..., y_\alpha\}$ denote a sequence of flows between a pair of hosts, and respectively their corresponding correct prediction. Then time-sensitive intrusion detection can be formalized as

$$\widetilde{\theta} = \arg\max_{\theta} P_\theta \left(y_1, y_2, ..., y_a | \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_a\right), \tag{5.1}$$

where the sequential model is parameterized by θ. It should be noted that the input flows to the model are always the communications between a pair of hosts so that tem-

poral correlations may exist, whereas previous, time-invariant NIDS aim at obtaining classification models that processes a single flow each time, i.e.,

$$\widetilde{\gamma} = \arg\max_{\gamma} P_{\theta}\left(y_i | \boldsymbol{x}_i\right), \tag{5.2}$$

Equation 5.1 would degrade to 5.2 if only one network flow is observed between a pair of hosts, but otherwise is a generalized version of the classification task for sequential inputs.

5.2 System Design

NetSentry is a NIDS that examines the statistical features of network flows and detects illicit traffic via an ensemble of sequential neural models. A traffic flow is built by grouping packets according to a five-tuple (Src IP, Dst IP, Src Port, Dst Port, Protocol). Recall that automated tools tend to initiate multiple almost identical flows towards targets during a short period of time. This means that the statistical features of malicious flows share a large degree of similarity. Thus, monitoring the similarities and discrepancies of consecutive flows between pairs of hosts plays an essential role in recognizing anomalies. To learn relevant temporal correlations of the traffic flows and to differentiate malicious patterns, NetSentry incorporates 4 key building blocks, as shown in Figure 5.2, namely:

- Flow Aggregator & Feature Extractor: groups packets into flows and extracts associated statistical features;
- Sequence Generator: groups flows originating from the same pair of hosts into fixed-length sequences, to be fed as inputs to anomaly detection logic;
- Feature Augmentor: increases the variability of a fraction of malicious traffic features that are non-essential in anomaly detection, but if left unchanged may increase the risk of model becoming trapped in local optima;
- Anomaly Detector (Bi-ALSTM): an ensemble of two asymmetric LSTM-based neural networks operated bidirectionally, taking flow sequences as input to detect malicious traffic.

Next, we explain the inner workings of each component, then detail our bidirectional sequential neural model in Section 5.2.3.

Feature Type	Direction	Name
timing-based	forward	Flow IAT*, packets/sec
	backward	Flow IAT*, packets/sec
	bi-direction	duration, Flow IAT*, packets/sec, bytes/sec, active time*, idle time*
protocol-based	forward	# packets, packet length*, PSH counts, URG counts, header length, initial TCP window size, avg segment size, subflow†
	backward	# packets, packet length*, PSH counts, URG counts, header length, initial TCP window size, avg segment size, subflow†,
	bi-direction	packet length*, flag counts§, down/up ratio, protocol
ID-based	None	flow ID, src IP, dst IP, src port, dst port, timestamp

Table 5.1: Features used in NetSentry. * means (min, max, avg, std) are computed for a given property. † indicates where (avg packets, avg bytes) are computed. § indicates (FIN, SYN, RST, PSH, ACK, URG, CWE, ECE) are counted in flows.

5.2.1 Feature Extractor and Sequence Augmentor

NetSentry employs a two-step process to extract numerical or categorical information (features) of the traffic observed, i.e., packet grouping and statistics computation. The former involves aggregating into flows packets generated between same pairs of applications, which can be achieved by monitoring origin, destination, and protocol fields.

Since NetSentry operates at the network layer and is not guaranteed to have access to packet payloads, we confine consideration to features that encompass timing statistics and protocol information. We find that employing popular open-sourced tools for feature extraction, such as CICFlowMeter [44] (which should be able to extract 80+ statistical features), is problematic. Indeed, CICFlowerMeter uses a faulty mechanism to identify the end of TCP flows, which results in benign traffic often being mislabeled as malicious, and vice versa.

With the CICFlowMeter feature extractor, if a new incoming TCP packet has a FIN flag set, the packet is immediately deemed to be the last packet in that *flow*. Obviously, this does not strictly follow the four-way handshake of TCP connection teardown. We show in Figure 5.4 that the premature assessment of termination leads to mislabeling, which is especially relevant to automated attacks such as DoS.

Assume that A is performing simple HTTP DoS attacks targeting B, quickly reusing a same port 8888. Also assume each time it is B who decides to terminate the TCP connections. Then, applying the mechanism described above on two consecutive flows

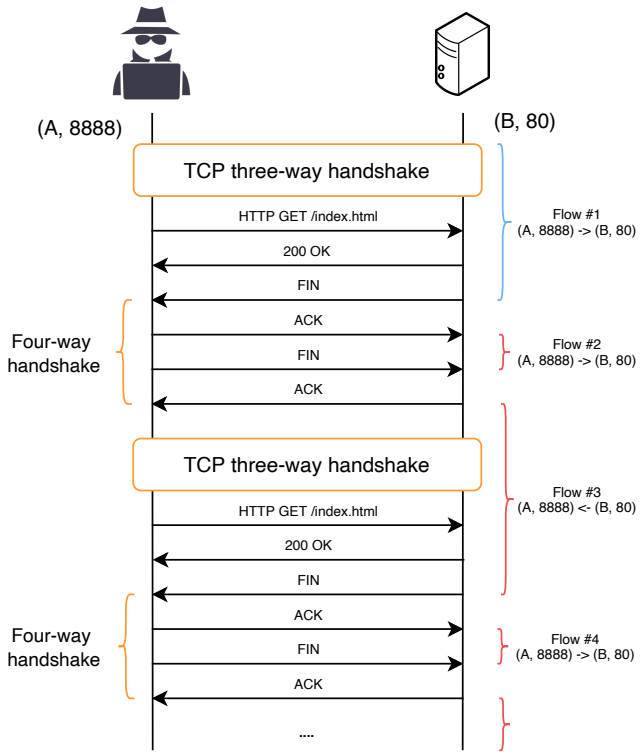


Figure 5.4: Incorrect flow labeling due to wrong TCP termination rules. Blue curly braces indicate the flow is labeled correctly; red curly braces indicate mislabeling.

from (A, 8888) to (B, 80) would generate 4 complete flows and an incomplete one, In this case, any flows from (A, 8888) to (B, 80) should be labeled as DoS. However, Flow #2 and #4 only consist of two packets (ACK and FIN) which cannot reflect any malicious purpose, while Flow #3 that should be labeled as malicious, is marked benign because of its wrongly perceived direction. We confirm that this type of mislabeling occurs for DoS-Hulk attacks in the publicly available CSE-CIC-IDS-2018 dataset [40] (which we use after correct relabeling), but further instances may exist.

We fix this logic error (along with other programming bugs encountered) in CICFlowMeter, by following a complete four-way handshake to terminate TCP flows. The timeout mechanism is preserved for stateless protocols. We also note that the original tool further extracts partial features that are not well-defined. Hence we revise the code and only output 69 features per flow, as summarized in Table 5.1.

Network anomaly is often nuanced, as a single network flow may be benign on its own, but observing multiple similar instances active at the same time may strongly suggest an automated attack in progress. Therefore, it is necessary to observe consecutive flows between hosts (applications) to further confirm malicious activity.

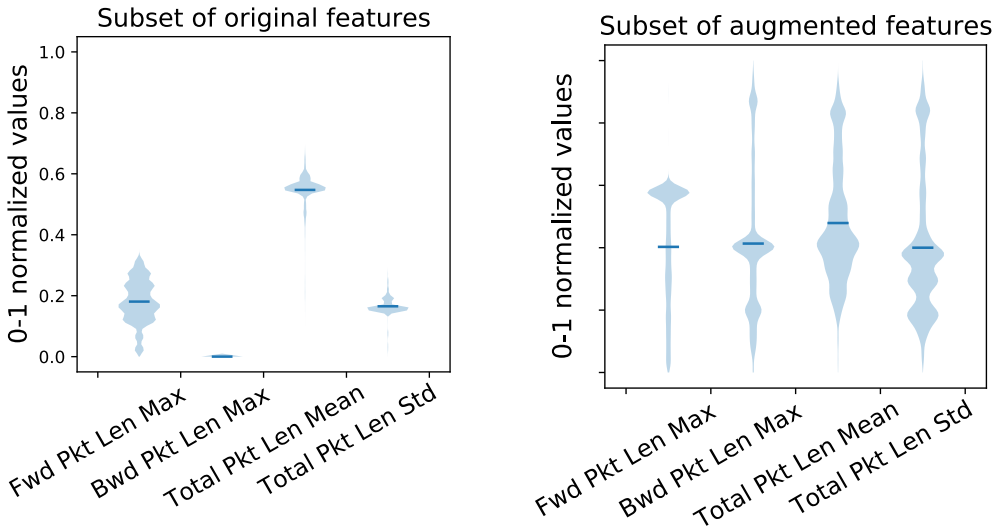
As such, a **sequence** in NetSentry is defined as successive flows using the same

Algorithm 4 NetSentry's sequence generation algorithm incorporating sliding windows and timeout thresholding.

1: **Inputs:**
   Tiemout value τ, and window size α
2: **Initialisation:**
   conn_table: A connection table storing the incoming flows from the feature extractor.
   seq_list: A FIFO list buffering the inputs for the feature augmentor or the anomaly detector.
3: **while** True **do**
4: $start\_time \leftarrow now()$
5: **for** $flow$ from the feature extractor **do**
6: $conn\_table[flow.id].add(flow)$
7: **if** $len(conn\_table[flow.id]) > \alpha$ **then**
8: $seq\_list.add(conn\_table.remove(flow.id))$
9: **end if**
10: **if** $now() - start\_time > \tau$ **then**
11: $seq\_list.addAll(conn\_table.removeAll())$
12: $start\_time \leftarrow now()$
13: **end if**
14: **end for**
15: **end while**

protocol between a pair of hosts, that is aggregated by (Src IP, Dst IP, Protocol). This is because with automated attacks, many-to-one (DoS, brute forcing) and many-to-many (port scanning) port attacks between a pair of hosts are common. Therefore, we regard a sequence through grouping not by the tuple used for flow aggregation, but by the origin and destination addresses, along with protocol type.

We adopt a flexible approach to generating sequences, which is a combination of sliding window and timeout thresholding techniques, as described by Algorithm 4. NetSentry allows two user-defined parameters for this purpose, namely window size α and timeout value τ, and maintains a connection table with two columns: *ID* and *seq_list*. The *ID* of each flow is a 3-tuple (Src IP, Dst IP, Protocol) and for each *ID*, a FIFO *seq_list* is maintained, storing the flows with the same *ID*. A newly generated flow is added to the *seq_list* determined by its *ID* (Alg. 4 line 6). Once the length of any *seq_list* is larger than α (Alg. 4 line 7), the elements in the list are regarded as a sequence to be passed to the neural model. Meanwhile, after every τ seconds, the entire connection table is emptied regardless of the length of the *seq_lists* (Alg. 4 line 10). Any list whose length is less than α is padded to α for the purpose of alignment. This design is customizable: Assume we observe the communications between N pairs of hosts, each of which generates T connections on average. The larger α is, the more comprehensive the context that the ensemble model obtains, but the higher the



(a) Before Augmentation (b) After Augmentation

Figure 5.5: Violin plots of 4 features before and after data augmentation. All values are normalized between 0 and 1.

memory requirements (the space complexity is $O(N\alpha)$); the smaller τ is, the more timely classification can be achieved, while the total time complexity remains the same, i.e., $O(NT)$.

5.2.2 Feature Augmentation

Since data used for ML training are largely collected in controlled environments, synthetically generated attacks may not offer an accurate view of network threats occurring in real-world [151], which prevents the model from learning a reliable decision boundary. For example, a victim HTTP server was set up to produce the CSE-CIC-IDS2018 dataset [40]; during HTTP DoS generation, during HTTP DoS generation, all flows encapsulated the same backward payloads from victim to attacker, resulting in little variability in payload-related features (see Figure 5.5, left). In reality, it is hard to predict how the victim would respond, and we show in Table 5.3 that such artificially low variability leads to poor generalization abilities for a range of supervised models.

We mitigate this problem by augmenting a collection of payload-related features to emulate a more realistic network environment. Specifically, we set up an HTTP victim server and an attacking client. The client only makes single requests with the `Keep-Alive` header over one TCP connection, to emulate HTTP DoS attacks. The size

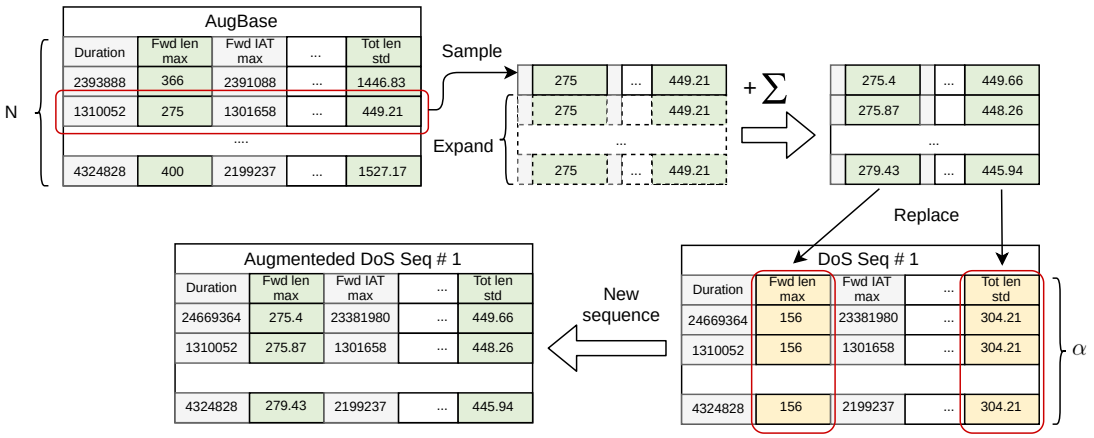


Figure 5.6: Illustration of the data augmentation process. Yellow cells denote payload features in original training data; green cells represents payload features from `AugBase`. Σ is a noise matrix with each element sampled from $\sigma \sim \mathcal{N}(0,5)$.

of each request and response is sampled from two discrete uniform distributions:

$$http\ request\ size \sim \mathcal{U}(100,400),$$

$$http\ response\ size \sim \mathcal{U}(100,15000).$$

In total, we generate 2,000 flows. A graphical illustration of the augmentation process is depicted in Figure 5.6. For each sequence that contains single-request HTTP DoS attacks (excluding Slowloris), (1) a flow is randomly sampled from the `AugBase` set and expanded to sequence length α; (2) random noise $\sigma \sim \mathcal{N}(0,5)$ is added to each payload-related cells to mimic minor differences among flows in a sequence; (3) finally, payload-related features in the original sequence are replaced by the new features generated at Step 2. By applying such augmentation, the new payload features of different flows in the same sequence would not differ much, but the features among different sequences would look considerably different. The distributions and the means of a subset of payload features in the augmented set are shown in Figure 5.5 (right). We use augmented data only for training.

It is also worth noting that the augmented data need not originate from any real traffic, since we only change parts of the features. However, what the model can learn from the augmented set is that *(i)* payload features possess high variability within some attacks (DoS in our case) whose logic does not rely on specific payloads, thus payload features should not be utilized for decision; and *(ii)* the rest of features (payload-irrelevant) are more valuable in distinguishing augmented attacks. We *choose not to remove payload-related features altogether because they may be important for the model to differentiate other types of attacks*, such as Slowloris, which only sends a small amount of payload during a long span. Note that the data augmentation tech-

nique is only effective on supervised models. Semi-supervised learning algorithms do not benefit from this technique because they only take benign data for training whereas augmentation is conducted on attacks. In Section 5.3, we demonstrate that augmentation boosts the performance of several supervised models.

5.2.3 Bidirectional Asymmetric LSTM

In this subsection, we introduce a variant of recurrent model – Bi-ALSTM. Utilizing recurrent models for sequential data is an intuitive approach for capturing inherent temporal dynamics. Network intrusion detection at the flow level does not constitute a real-time task. This is because feature extraction and classification occur only once the flow has been terminated. Consequently, we do not regard computational efficiency as a limiting factor, but focus on designing an algorithm with a robust capability for processing temporal information. As a result, we opt out of using simpler but faster architectures, such as RNN and GRU. We overview the different blocks that lay foundations for our Bi-ALSTM, then explain the ensembling and why our approach is essential for high-performance classification.

LSTM. As a variant of RNN, LSTM [68] incorporates gating functions to simulate the update of memory units along time and has shown excellent ability to model long-term dependencies in sequential data [154, 86]. An LSTM maintains two states: a cell state c_t and a hidden state h_t, which is computed based on inputs up to timestamp t, i.e., $\mathbf{x}_1, ..., \mathbf{x}_t$ ($\mathbf{x}_i \in \mathbb{R}^d$). To maintain long-term dependencies, the LSTM cell has *input* (i_t), *forget* (f_t), and *output* (o_t) gates controlling the information flowing through at different timestamps. Gates are modeled by single-layer neural networks with parameters $W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, W_{ho} \in \mathbb{R}^{h \times d}$ and associated biases, i.e.,

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right), \tag{5.3}$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right), \tag{5.4}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh\left(W_{xc}x_t + W_{hc}, h_{t-1} + b_c\right), \tag{5.5}$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right), \tag{5.6}$$

$$h_t = o_t \circ \tanh\left(c_t\right), \tag{5.7}$$

where σ denotes the sigmoid function and $\circ$ represents element-wise product. When a new input x_t is given to the LSTM, the cell state c_t is updated with information from x_t and the previous cell state c_{t-1}. The proportion of x_t and c_{t-1} in the new cell state is determined by i_t and f_t, as in Eq. 5.5. h_t can be perceived as a non-linear transformation

on c_t, and are always used for downstream tasks, such as classification. In NetSentry, an MLP $g_\phi(\cdot)$ is used to approximate the probability of y_t given h_t.

When it comes to intrusion detection against automated network attacks, the order of attack flows in a sequence is less important. Hence, at any timestamp t, both previous inputs and subsequent inputs may be equally highly correlated with x_t. *The traditional LSTM can only model temporal information unidirectionally as time evolves.* In other words, the hidden representation h_t only comprises the context before or at timestamp t. To generate more comprehensive hidden representations for anomaly detection, a Bidirectional LSTM (Bi-LSTM) which runs two LSTMs separately (one forward, one backward), and whose hidden states are concatenated before given to $g_\phi(\cdot)$, can be used. The objective of Bi-LSTM is thus:

$$\begin{aligned}&\max_{W,b,\boldsymbol{\phi}} p(y_1, y_2, ..., y_a | \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_a, W, b, \boldsymbol{\phi}) \\ &= \max_{W,b,\boldsymbol{\phi}} \prod_{t=1}^{\alpha} p(y_t | \boldsymbol{x}_1, ..., \boldsymbol{x}_\alpha, W, b, \boldsymbol{\phi}) = \max_{W,b,\boldsymbol{\phi}} \prod_{t=1}^{\alpha} g_\phi(h_t),\end{aligned}$$

where W and b are the weights and biases in LSTM cells and $\boldsymbol{\phi}$ the parameters of the MLP. Bi-LSTM is one of the benchmarks we consider in evaluating our work.

ConvLSTM. Convolutional LSTM (ConvLSTM) [181] was first proposed to model spatio-temporal data, such as radar echo maps, whose spatial correlations cannot be extracted by fully connected layers in LSTM. ConvLSTM tailors the convolution operation into LSTM by replacing matrix multiplication operations in (5.3)–(5.7) with convolution, as follows:

$$i_t = \sigma\left(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + b_i\right), \tag{5.8}$$

$$f_t = \sigma\left(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + b_f\right), \tag{5.9}$$

$$\mathcal{C}_t = f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh\left(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \tag{5.10}$$

$$o_t = \sigma\left(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + b_o\right), \tag{5.11}$$

$$\mathcal{H}_t = o_t \circ \tanh\left(\mathcal{C}_t\right), \tag{5.12}$$

in which $*$ denotes the convolution operator, $\mathcal{X}_t \in \mathbb{R}^{d\times 1}$ is the input, and W_{xi}, W_{hi}, W_{xf}, $W_{hf}, W_{xo}, W_{ho} \in \mathbb{R}^{k\times c}$ are the convolution kernels. The above applies to a single ConvLSTM unit, which can be further extended to multiple layers as traditional LSTM does. A pooling layer can be added between each two ConvLSTM layers to reduce computation.

An immediate concern is whether applying convolution-embedded models on network traffic data without obvious spatial information would be effective. In fact, CNN

not only gained success in computer vision [64, 184], but also in areas including web traffic fingerprinting [148] and mobile traffic forecasting [191]. As the sequential data we deal with are one-dimensional, we implement a 1-D ConvLSTM, which takes inputs with channel and length dimensions, where channel by default equals 1.

Differences from CNN-LSTM. The CNN-LSTM, or Long-term Recurrent Convolutional Networks (LRCN), is a combination of CNN and LSTM, first proposed for visual recognition. It differs from ConvLSTM in that in the former a separate CNN handles spatial information before providing input to LSTM. In contrast, the latter has a compact form. While previous studies apply CNN-LSTM to NID [76], our work is the first to leverage ConvLSTM and study the differences between the two structures. Our results in Section 5.3 reveal that ConvLSTM consistently outperforms CNN-LSTM.

Bidirectional Asymmetric LSTM. Bi-LSTM and Bi-ConvLSTM can be perceived as ensemble models with two separate LSTM units. The hidden states from two units are concatenated so that future information is accessible at the current timestamp t, which can potentially benefit the downstream classification task. Since our targets are sequences with *similar* malicious flows, it is the hidden states at the two ends of the structures that can acquire most information. The hidden representations in the middle of a sequence from two (Conv-)LSTM units yield certain amount of redundant information when the same architecture is used.

Given the fact that different architectures are likely to exploit different facets of input features for classification [124], we propose **Bi-ALSTM**, which consists of two different, asymmetric LSTM units, one for forward and the other for backward processing, to generate intermediate representations that incorporate more comprehensive temporal contexts as shown in Alg. 5 line 5-8. The hidden states from two different units are first linearly combined, then fed through an activation function Alg. 5 line 9. Precisely, denote $\mathbf{h}^t_{fc} \in \mathbb{R}^{N_1}$ as the hidden state generated by LSTM at timestamp t, and $\mathbf{h}^t_{conv} \in \mathbb{R}^{N_2}$ generated by ConvLSTM (operated backwards). The final hidden states are formed through the following fusion operation:

$$\mathbf{h}^t = \tanh\left(\frac{\mathbf{U}_{conv}\mathbf{h}^t_{conv}}{||\mathbf{U}_{conv}\mathbf{h}^t_{conv}||_2} + \frac{\mathbf{U}_{fc}\mathbf{h}^t_{fc}}{||\mathbf{U}_{fc}\mathbf{h}^t_{fc}||_2}\right), \tag{5.13}$$

where $\mathbf{U}_{conv} \in \mathbb{R}^{N\times N_2}$ and $\mathbf{U}_{fc} \in \mathbb{R}^{N\times N_1}$ are learnable parameters. $\mathbf{h}^t_{fc}$ and $\mathbf{h}^t_{conv}$ are projected to the same subspace and L_2-normalized, so that in the learning process, any single unit would not easily dominate the final results. This design allows two asymmetric LSTMs to produce hidden states with different dimensions, meaning that

two different LSTM structures can be tuned separately and flexibly before building the Bi-ALSTM. Finally, a single FC layer $g_\phi(\cdot) : \mathbb{R}^N \to \mathbb{R}^C$ with softmax function is used to approximate the probability of the samples belonging to each class.

Computational complexity: We first derive the time complexity of a single-step forward pass of LSTM. Given that the time complexity of $W_x x_t$ is $O(hd)$, it is easy to know that the time complexity inside each nonlinearity σ is $O(hd+h^2) = O(h(d+h))$. Assuming σ and $tanh$ have a constant time complexity, applying σ or $tanh$ element-wise yields the complexity of h, which can be omitted due to the existence of $O(h^2)$. Therefore, the time complexity of a single-step forward pass equals $O(h(d+h))$. Similarly, given that $W_h * \mathcal{X}_t$ has time complexity $O(dkc)$, the time complexity of a single-step forward pass of ConvLSTM is $O(dkc + dkc^2) = O(dkc^2)$. Assume the input sequence to Bi-ALSTM has the length n. The computational complexity of a single-layer Bi-ALSTM results in $O(n(h(d+h)+dkc^2))$.

We use negative log likelihood with L_2 regularization as the loss function (Alg. 5 line 10). Bi-ALSTM is stochastically optimized via back propagation through time by the Adam algorithm. The complete training process follows Algorithm 5.

Algorithm 5 The training algorithm for Bi-ALSTM

1: **Inputs:**
   $\mathcal{D} := \{S_1, ..., S_n\}; S_i := \{(\mathbf{x}_{i1}, y_{i1}), ..., (\mathbf{x}_{i\alpha}, y_{i\alpha})\}$,
   $\lambda_1 := L_2$ *weight decay*, $\lambda_2 :=$ *learning rate*.
2: **Initialisation:**
   Denote $h_{W_{fc}}(\cdot)$ and $h_{W_{conv}}(\cdot)$ as the LSTM and ConvLSTM unit parameterized by W_{fc} and W_{conv}, and predictor $g_\phi(\cdot)$ parameterized by ϕ. W_{fc}, W_{conv}, ϕ set via Xavier initialization [52].
3: **while** model has not converged **do**
4: **for** S_i sampled from $\mathcal{D}$ **do**
5: $\overrightarrow{X_i} \leftarrow (\mathbf{x}_{i1}, ..., \mathbf{x}_{i\alpha}), \overrightarrow{\mathbf{y}_i} \leftarrow (y_{i1}, ..., y_{i\alpha})$
6: $\overleftarrow{X_i} \leftarrow reverse(\overrightarrow{X_i})$
7: $\overrightarrow{H}_{i,fc} \leftarrow h_{W_{fc}}(\overrightarrow{X}_i)$
8: $\overrightarrow{H}_{i,conv} \leftarrow reverse(h_{W_{conv}}(\overleftarrow{X}_i))$
9: $\overrightarrow{H}_i \leftarrow \overrightarrow{H}_{i,fc} \oplus \overrightarrow{H}_{i,conv}$ ▷ Eq. 5.13
10: $\mathcal{L} \leftarrow NLL(softmax(g_\phi(\overrightarrow{H}_i), \overrightarrow{\mathbf{y}_i}) + \lambda_1(||W_{conv}||_2^2 + ||W_{fc}||_2^2 + ||\phi||_2^2)$
11: $\phi, W_{conv}, W_{fc} \leftarrow Adam(\mathcal{L}, \phi, W_{conv}, W_{fc}; \lambda_2)$
12: **end for**
13: **end while**
14: **return** W_{fc}, W_{conv}, ϕ

5.2.4 Abstract Labeling

Applying supervised methods for NID commonly involves training a multi-class classifier that seeks to detect as many types of malicious attacks as possible. However, we argue that this approach would lead to an overfitted model because the *ambiguity of the true attack labels* is widely overlooked. To understand this, consider a classifier is trained to differentiate two types of network attacks: SQLmap fuzzing vs. Web password Bruteforcing, both of which repetitively initiate HTTP requests to a given API. Although the two types of attacks would incorporate different payloads (SQL scripts and user/password pairs respectively), this discrepancy appears negligible on a statistical level, because the contents of payloads would not be extracted. Given that both trigger database I/O operation, the extracted timing information would be hardly differentiable.

Thus if a trained model can distinguish between such attacks, it has to be overfitted and learn a decision boundary that is unique to the dataset, rather than truly understand the differences. Attack labels usually indicate the purposes and techniques behind them, but when it comes to network layer, the attack realizations, i.e., traffic flows, would not be clearly dissimilar. In this regard, using sequential models to distinguish as many types of attacks as possible is unrealistic.

Given that it is hard to correctly predict every type of automated cyberattack with a network-based algorithm, reverting to a binary detector seems sensible. However, since we augment DoS attacks, we decide to use **abstract labeling** in order to evaluate the augmentation technique and to avoid the aforementioned overfitting issue. Specifically, we assign a number of abstract, generic labels, including *benign*, *DoS*, *portscanning* and *bruteforcing & fuzzing*, as ground truth during training. On one hand, this approach can clearly illustrate the influence of our feature augmentation technique. On the other hand, the model would not put effort in distinguishing the subtle differences between, e.g., DoS HOIC and DoS LOIC, which may not be separable by a network-based algorithm.

5.3 Performance Evaluation

We implement NetSentry in PyTorch and train the model on a GeForce Titan X GPU. To build the Bi-ALSTM, we use an LSTM unit for the forward pass and a ConvLSTM unit for the backward pass. The LSTM unit has the following structure:

$dropout(0.5) \rightarrow lstm(65,48) \rightarrow lstm(48,48)$, where the arguments in $lstm(\cdot,\cdot)$ denote the input size and the hidden size. The ConvLSTM unit encompasses $convlstm1D(1,3,3) \rightarrow convlstm1D(3,6,3) \rightarrow maxpool()$, in which the arguments in $convlstm1D(\cdot,\cdot,\cdot)$ represent the input channel size, output channel size, and kernel size. The fused hidden states are passed through: $dropout(0.3) \rightarrow MLP(32,5)$. The L_2 penalty and learning rate are set to 0.5 and 0.001 respectively.

For the Flow Aggregator/Sequence Generator, apart from our TCP termination fix (see Section 5.2.1), we set flow timeout to 30s and subflow duration 5s in CICFLowMeter. The Sequence Generator uses timeout $\tau = 30$s and window size $\alpha = 10$.

5.3.1 Network Intrusion Datasets

We experiment with two datasets published by the Canadian Institute for Cybersecurity (CIC), as described below.

CIC-IDS-2017 [144] contains most common cyberattacks, including bruteforcing, heartbleed, botnet, (D)DoS, Infiltration, and Web attacks. Traces were collected in a LAN, with benign traffic generated by profiling normal online behaviors of 25 users on different OSs, including Win Vista, Win 7, Win 8, Win 10, Mac OS, and Ubuntu 12. Attacks are produced by 4 different machines with one running Kali Linux and three Win 8.1. The traffic collection spans 5 working days, and in total 51.1 GB of pcap files are open-sourced. The feature sets of the corresponding `pcap` files were published, but as we reveal in Section 5.2.1, these were extracted incorrectly. Hence, we only use the raw capture files in our experiments.

CSE-CIC-IDS2018 [40] was generated in a much larger environment where an organizational LAN with five subnets simulated give different departments. 450 machines act as normal users and 50 as attackers. The dataset contains a wider range of benign traffic, including HTTPS, HTTP, SMTP, POP3, IMAP, SSH, and FTP, and contains a larger attack collection (17 types). The dataset spans 10 days.

Self-collected traffic – since *FTP-Bruteforce* and *DoS-SlowHTTPTest* attacks were erroneously collected in CSE-CIC-IDS2018, the generated traffic merely contains SYN and RST packets. To mitigate this, we collected FTP-bruteforcing traffic ourselves, generating 4,050 flows. We decide not to collect *DoS-SlowHTTPTest* traffic, since a similar type of attack, i.e., *DoS-Slowloris*, exists in CSE-CIC-IDS2018. We remove the mislabeled traffic and merge the self-collected traffic with the CSE-CIC-IDS2018 dataset. In the rest of our paper, we use CSE-CIC-IDS2018 to refer to the merged

dataset. We employ our revised version of CICFlowMeter to generate network flows based on the `pcap` files. The statistics of both datasets are shown in Table 5.2.

	# Features	# Instances	Anomaly ratio	Automated attack ratio
IDS-2017	69	2,607,289	0.2	0.189
IDS-2018	69	8,786,169	0.1806	0.1802

Table 5.2: Statistics of datasets used for experimentation. ID-based features in Table 5.1 not included in the training; 'protocol' is hot-encoded. Thus, only 65 features used in total.

Cross-Evaluation We adopt a rigorous evaluation methodology, aiming to show the true generalization ability of our design, by **cross-evaluation**. Normally, a dataset is split into training and testing subsets, and the results on the test set compared across different algorithms. However, network environments are heterogeneous and data collected in one environment may not accurately reflect the diversity seen in practice. To test if an algorithm can truly distinguish the same type of malicious traffic in a different network topology, we also evaluate it on a second, unseen dataset. CSE-CIC-IDS2018 is split into training (70%) validation (15%) and test (15%) sets. To maintain time consistency, training data is selected from events that took place before the test data. CIC-IDS-2017 is purely used for cross-evaluation, after the model was trained on the former.

5.3.2 Benchmarks

For comparison, we implement a range of benchmarks, including basic ML/DL structures (MLP, CNN, autoencoder, RIPPER [87], Decision Tree); state-of-the-art anomaly/ intrusion detectors, i.e., One Class Neural Nets (OC-NN) [134], Kitsune/KitNET [107] and Deep Autoencoding Gaussian Mixture Model (DAGMM) [195]; and three Bi-LSTM variants [76].

OC-NN [134] and DAGMM [195] are offline semi-supervised algorithms for general anomaly detection. OC-NN aims to learn a mapping for the benign samples to a kernel space where the majority of them can be enclosed by a hypersphere. During the testing phase, the distances from the samples to the center of the hypersphere represent the anomaly score of the data. Different from OC-NN, DAGMM models the benign data from a probabilistic perspective with a mixture of Gaussian distributions. The negative probability of the data being sampled from the PDF represents the anomaly score.

Kitsune [107] is an online semi-supervised NIDS. It uses an ensemble of shallow autoencoders to learn the features of benign data in different subspaces; a final autoencoder fits the correlations of the reconstruction errors from the shallow autoencoders. The neural architecture is named KitNET. During testing, the reconstruction errors are computed to represent the degree of abnormality. For a fair comparison, KitNET is trained in an offline manner with more than one epoch.

For semi-supervised algorithms (OC-NN, DAGMM, KitNET and Autoencoder), an anomaly ratio α needs to be preset, indicating the proportion of anomalous samples, and during the testing phase, the data with the top $\alpha \times 100\%$ of the anomaly scores are classified as anomalous. The anomaly ratio is set to 0.189 and 0.1802 on CIC-IDS-2017 and CSE-CIC-IDS2018 respectively, which is the same percentage of automated attacks in the two datasets.

RIPPER [87] and Decision Tree are two basic machine learning models, where the first one aims to generate a simple ruleset for classifications while the second embeds rules in a tree by recursively finding the best splits.

The structures of Bi-LSTM and Bi-ConvLSTM resemble the units in Bi-ALSTM. For CNN-Bi-LSTM, an extra CNN block, with the structure: $Conv1D(1,3,3) \rightarrow MaxPooling \rightarrow Conv1D(3,6,3) \rightarrow MaxPooling$, is implemented. The arguments in $Conv1D(\cdot,\cdot,\cdot)$ represent input channels, output channels, and kernel sizes.

Prior to testing, we retrain all benchmarks with all the features in the CSE-CIC-IDS2018 dataset, which is richer than the datasets used for training in the original papers.

5.3.3 Evaluation Metrics

The average precision, recall and F1 score are commonly used to evaluate the performance of anomaly detection algorithms. These metrics can be measured based on the True Positives (TP), False Positives (FP), True Negatives (TN) and recall are computed as $precision = TP/(TP+FP)$, $recall = TP/(TP+FN)$. The precision indicates how likely the algorithm would give true alarms, and the recall measures how sensitive the algorithm is towards anomalies. There exists a trade-off between precision and recall, and to obtain an overall performance measure, their harmonic average is computed, i.e., the F1 score: $F1 = 2 \times \frac{precision \times recall}{precision + recall}$.

We do not measure accuracy i.e., the percentage of the correctly classified samples, which is unlikely to reveal the algorithms' true NID performance: consider a dataset

Algorithm	CSE-CIC-IDS2018			CIC-IDS-2017 (X-eval)		
	precision	recall	F1	precision	recall	F1
RIPPER	0.9983	0.0981	0.1786	0.0873	0.0106	0.0190
Decision Tree	0.9989	0.9990	0.9990	0.5385	0.3717	0.4398
MLP	0.9989	0.9962	0.9976	0.6736	0.4631	0.5435
CNN	0.9947	0.9951	0.9949	0.7705	0.6344	0.6958
Autoencoder	0.7783	0.7500	0.7639	0.4362	0.4197	0.4278
OC-NN	0.9722	0.5310	0.6868	0.7844	0.5136	0.6208
Kitsune	0.6310	0.6081	0.6193	0.4086	0.3932	0.4007
DAGMM	0.8666	0.8253	0.8454	0.4159	0.3116	0.3576
Bi-LSTM	0.9990	0.9979	0.9985	0.7258	0.4209	0.5317
CNN-Bi-LSTM	**0.9996**	0.9982	0.9989	0.8813	0.3750	0.5261
Bi-ConvLSTM	0.9984	0.9971	0.9977	0.8721	**0.9693**	0.9178
Bi-ALSTM	0.9994	**0.9990**	**0.9992**	**0.9116**	0.9446	**0.9275**
Algorithm + augmentation	CSE-CIC-IDS2018			CIC-IDS-2017 (X-eval)		
	precision	recall	F1	precision	recall	F1
RIPPER	0.9980	0.0934	0.1709	0.4998	0.1837	0.3687
Decision Tree	0.9989	**0.9993**	**0.9991**	0.5897	0.8556	0.6914
MLP	0.9989	0.9963	0.9976	0.7540	0.8690	0.8071
CNN	0.9925	0.9847	0.9886	0.7453	0.8687	0.8021
Bi-LSTM	0.9991	0.9956	0.9973	0.8555	0.9777	0.9125
CNN-Bi-LSTM	0.9996	0.9966	0.9981	0.8479	0.9683	0.9041
Bi-ConvLSTM	0.9996	0.9975	0.9985	0.8728	0.9780	0.9222
Bi-ALSTM	**0.9997**	0.9976	0.9987	**0.9190**	**0.9800**	**0.9485**

Table 5.3: Precision, recall, and F1 score for Bi-ALSTM and benchmarks. NB: only supervised algorithms can be trained with augmented data and only HTTP (D)DoS attacks are augmented; semi-supervised methods only require benign traffic for training.

with 80% benign and 20% malicious instances; a model that classifies everything as benign has the same accuracy as a model that correctly recognizes all but 20% of the benign traffic.

For the ML-based algorithms that output an anomaly score for each test instance rather than just the predicted class, system administrators may choose a threshold higher than 0.5, which guarantees that the classifier has a lower FPR. We plot the Receiver Operating Characteristic (ROC) curve for sequential models, to evaluate their performance when the anomaly threshold is varied in $[0, 1]$. The ROC curve is obtained by plotting the False Positive Rate (TPR) against FPR. The closer to 1 the Area Under Curve (AUC) is, the better the classifier performs.

Given that our datasets consist of multiple types of cyberattacks, we further plot the

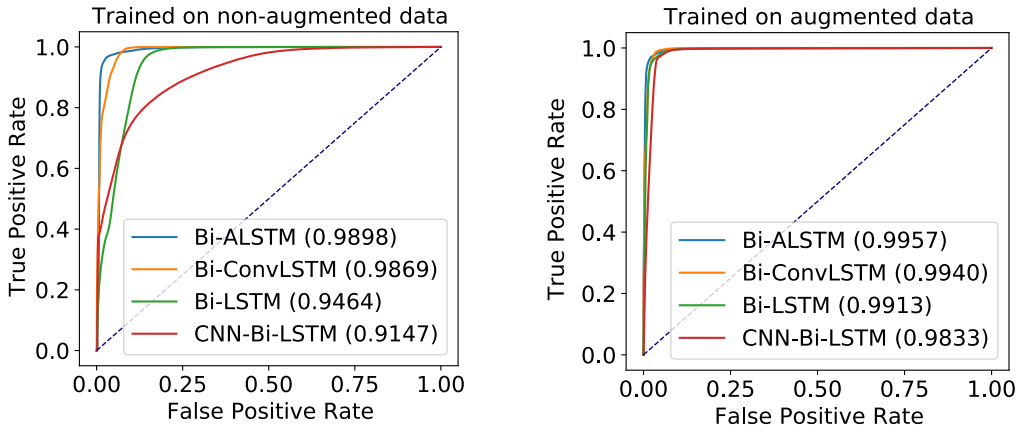
ECDF of the anomaly score with respect to each type of traffic on the crossed evaluated dataset, to illustrate the confidence of each sequential model.

Beyond the metrics for classification performance, we also care about the computational overhead of our design, and therefore report Multiply-Accumulate Operation Counts (MACs), the number of parameters and the concurrent processing capacity of each model on a edge GPU. MACs and the parameter numbers reveal the complexity of different algorithms at a micro level, while the concurrent processing capacity on GPU can reflect the computational bottlenecks.

5.3.4 Overall Results

We summarize our comparison in terms of threat detection performance between our Bi-ConvLSTM/-ALSTM models and the benchmarks considered, in Table 5.3. In the upper half, the different algorithms are trained on non-augmented data. The performance of the benchmark algorithms and those adopted by our NetSentry is similar on the CSE-CIC-IDS2018 dataset, most of them attaining average metrics above 0.99. CNN-Bi-LSTM slightly outperforms other algorithms in terms of precision, while Bi-ALSTM yields the highest recall and F1 score. An interesting finding is that the semi-supervised algorithms for general anomaly detection may not be suitable for network intrusion detection. Autoencoder, OC-NN and DAGMM cannot compete with basic supervised ML algorithms. One of the core assumptions for semi-supervised anomaly detection is that the algorithm can learn the characteristics of benign data, by estimating the probability, reconstructing the benign samples or finding an appropriate hyper-boundary enclosing them. However, network traffic is highly heterogeneous, serving with different protocols various applications, such as email, web browsing, streaming, etc. It remains questionable whether the aforementioned assumption holds on such a large range of 'benign data'. Besides, detecting malicious traffic, especially automated attacks (which is our objective), appears to be a time-sensitive task. Therefore, observing a single instance may be insufficient to make reliable decisions. Consequently, existing anomaly detection algorithms tend to ignore this, which leads to modest results.

The advantage of Bi-ConvLSTM and Bi-ALSTM can be clearly seen on cross-evaluation results, where our models maintain consistently competitive performance. Both attain F1 scores above 90%, while other supervised algorithms, including Bi-LSTM and CNN-Bi-LSTM exhibit a significant performance drop (F1 score around



(a) Without Augmentation (b) With Augmentation

Figure 5.7: ROC curves of LSTM-based algorithms. AUC behind labels.

50%). We notice that though both CNN-LSTM and ConvLSTM are proposed to handle spatio-temporal data, there is an obvious difference in performances, both in terms of F1 score (Table 5.3) and ROC (Figure 5.7) on intrusion detection. As shown in Figures 5.8 (a), (b), Bi-LSTM and CNN-Bi-LSTM do not learn a reliable decision boundary between benign traffic and DoS attacks without the augmented data. Bi-ConvLSTM clearly outperforms them, yet still exhibits a high probability of classifying DoS as port scanning attacks, as illustrated in Figure 5.8(c), whereas Bi-ALSTM is the most reliable (Figure 5.8(d)).

5.3.5 Impact of Feature Augmentation

The data augmentation procedure we propose is highly effective in helping the models generalize well. *When the supervised models are trained with the augmented dataset, a remarkable performance gain can be observed in the cross-evaluation results* (Table 5.3, bottom half). The F1 scores of the benchmarks increase by at least 16%, and Bi-LSTM and CNN-Bi-LSTM even jump to 90%. The improvements of Bi-ConvLSTM and Bi-ALSTM are less noticeable since outstanding results can be achieved even without augmented data, but still, the former reaches the highest recall and Bi-ALSTM is the most robust in terms of overall performance.

Observing confusion matrices in the second column in Figure 5.8, all models reveal roughly the same pattern, as opposed to the corresponding results on the first. This confirms that *augmentation encourages models to learn the associated timing info rather than payload features* in the classification task.

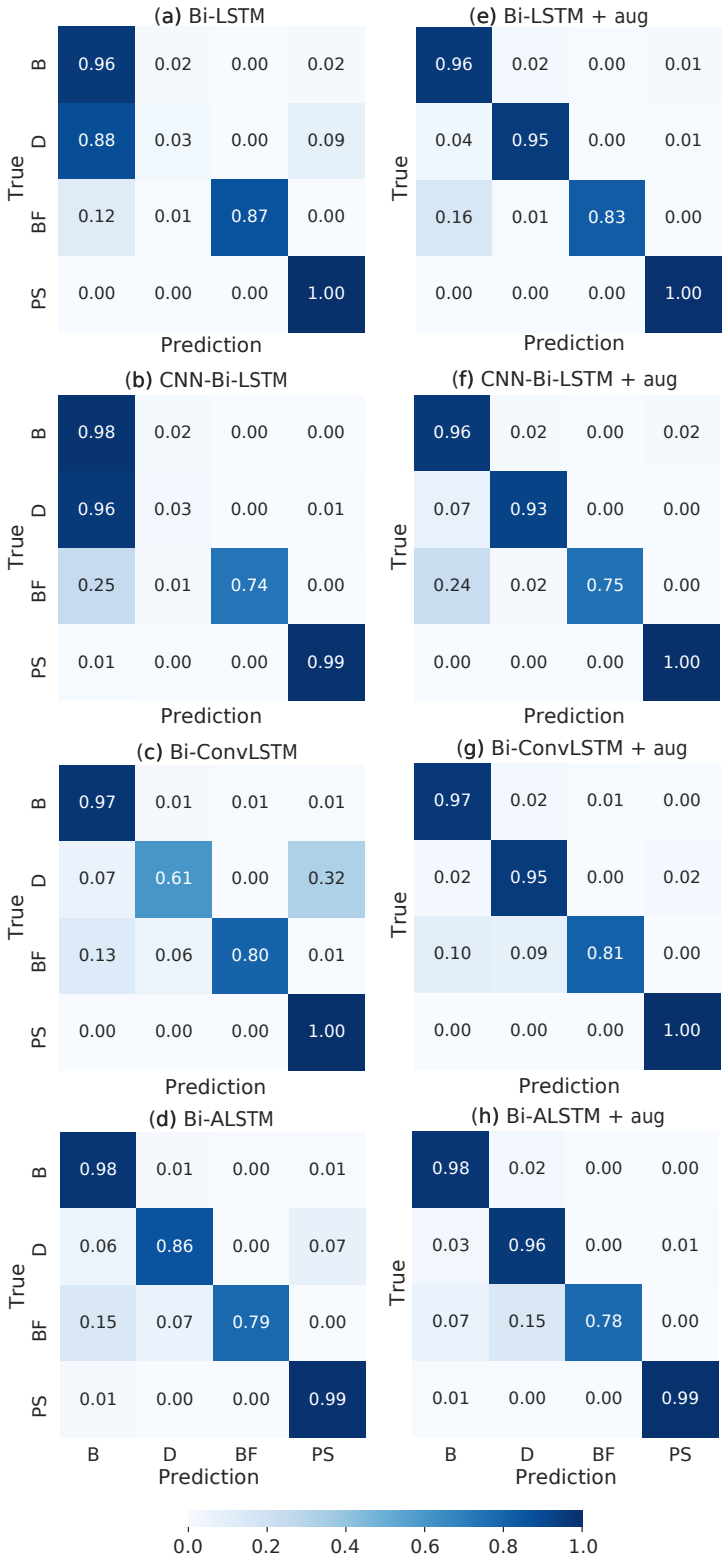


Figure 5.8: Normalized confusion matrices (row values add to 1) for LSTM-based models cross-evaluated on CIC-IDS-2017. Models trained with/without augmented dataset (left/right). B represents **B**enign, D **D**oS, BF **B**ruteforcing and **F**uzzing, and PS **P**ort**S**caning. Numbers on diagonals are recalls.

5.3.6 Impact of Feature Arrangement

We investigate the influence of the feature arrangement and kernel size on the performance of ConvLSTM. For this, we experiment with 3 different kernel sizes, namely (3, 5, 7), and two sets of feature arrangements. Specifically, the 1D feature vectors are logically ordered and randomly shuffled. Note that most of the features listed in Table 5.1 in the Appendix are computed for forward traffic only, backward traffic only, and bidirectionally. Logically ordered features means that they follow an alternating order of forward, backward, and bi-direction. In each experiment, we train both unidirectional ConvLSTM and Bi-ConvLSTM with the augmented dataset for 10 epochs and repeat the process 5 times. The mean and error bars of the F1 score on the cross-evaluation dataset (CIC-IDS-2017) are illustrated in Figure 5.9.

Intuitively, one might expect ConvLSTM would only work with sequential data possessing clear spatial information, such as videos. However, we find that ConvLSTM is robust to 1D traffic features regardless of their arrangement. Indeed, the results in Figure 5.9 demonstrate that there is no significant gap between the model trained with logically ordered features or randomly shuffled ones. For most cases, the mean in the former case is slightly higher than in the latter, while the error bars show a large degree of overlap.

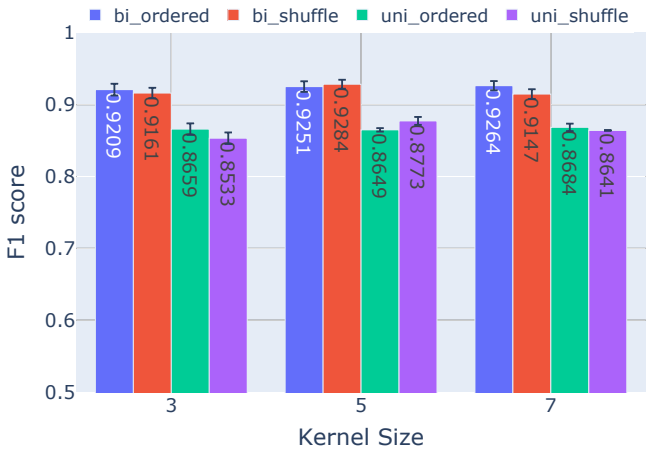


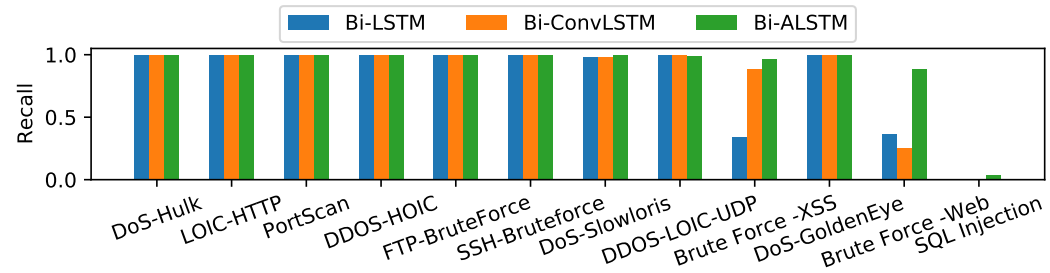
Figure 5.9: The F1 scores attained by (Bi-)ConvLSTM with different kernel sizes and different order of features.

We also find that although F1 scores tend to rise sightly with larger kernel sizes when the ConvLSTM is trained with the ordered feature arrangement, this increase is not significant. Considering the growth in computation overhead with using a larger kernel size, we argue that training both Bi-ConvLSTM and Bi-ALSTM with a kernel size equal to 3 (which was also the case for the results presented in Table 5.3) is

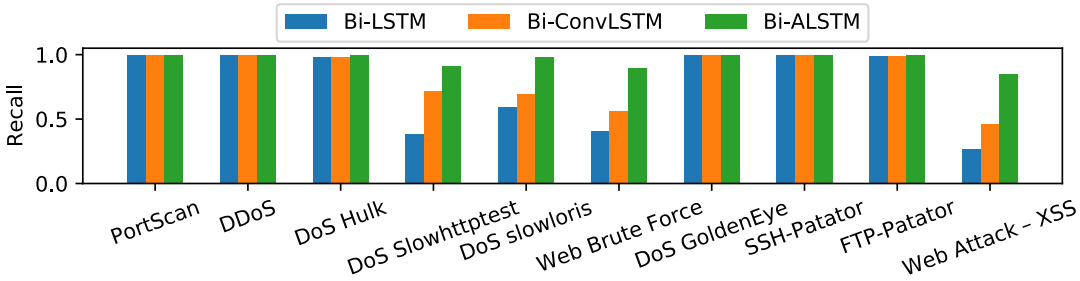
sufficient.

5.3.7 Performance Gains of Bi-ALSTM

Bi-ALSTM not only yields the highest overall detection rate (recall), but also reliably detects each type of cyberattacks, as illustrated in Figure 5.10. Both Bi-LSTM and Bi-ConvLSTM have difficulty recognizing web bruteforcing, XSS, and Slowloris attacks, whereas *Bi-ALSTM attains up to 3× higher detection rates*. The only exception is SQL injection, which cannot be detected by all the algorithms. This is because there are only 53 instances of this attack, merely accounting for 0.0006% of the entire dataset, which is insufficient for the model to learn a reliable decision boundary for classification.



(a) Evaluated on CSE-CIC-IDS2018



(b) Evaluated on CIC-IDS-2017

Figure 5.10: Detection rate (recall) of each type of traffic.

We evaluate the quality of the anomaly scores approximated by Bi-(Conv)LSTM and Bi-ALSTM. The anomaly score is the value output by the model. Since the activation function of the last layer is softmax, the output is squeezed between $[0, 1]$ and the higher the value, the more anomalous a flow is regarded. System administrators routinely customize an anomaly threshold to lower the FPR. Figure 5.11 plots the ECDF of each type of traffic in CIC-IDS-2017 given by the three algorithms, in which the blue line corresponds to benign traffic. The black dashed line is the threshold that sets the FPR to 1.5%, and the area under the other lines to the left of the threshold line represents the proportion of attacks that would be misclassified. We find that

Bi-ALSTM delivers the lowest False Negative Rate (FNR) (2.63%) compared with Bi-LSTM (10.17%) and Bi-ConvLSTM (5.87%).

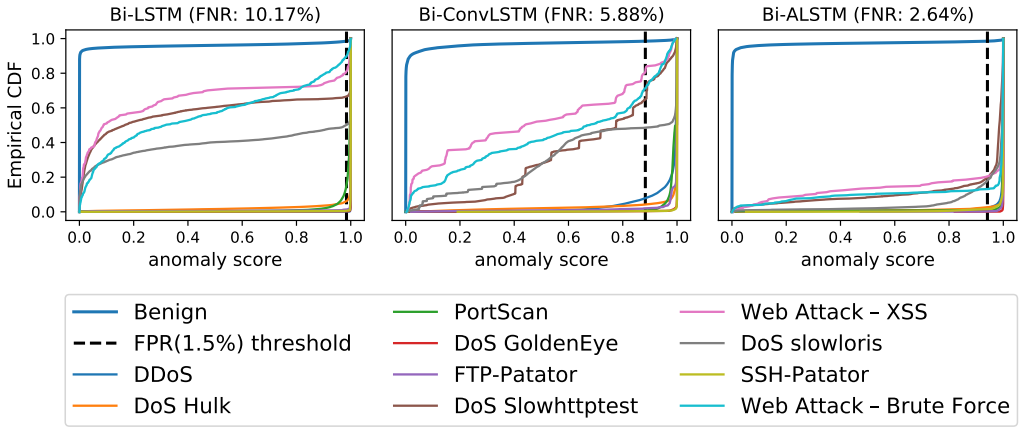


Figure 5.11: ECDF of the anomaly scores with respect to each type of traffic in CIC-IDS-2017 given by Bi-LSTM, Bi-ConvLSTM and Bi-ALSTM. All models trained with augmented dataset.

5.3.8 Impact of Adversarial Perturbation

Adding small perturbations to input data may lead to misclassification by ML models [23]. Nevertheless, the existing adversarial attacks often require access to model gradients, structures, or numerous queries for weight approximation. In reality, a ML-based NIDS would not disclose the details of its neural model and tolerate countless queries. [192] demonstrate the possibility of attacking ML-based intrusion detection algorithms by heuristic-based methods without knowing model's information, but it would still take 100∼11,000 queries to generate one adversarial sample. Note that NetSentry is intended for continuous and repetitive network attacks, meaning that similar queries would trigger alarms much earlier than discovering a valid adversarial sample.

Adversarial perturbations are likely to modify every individual feature to create malicious samples, which is not always practical in the networking domain, since the modified flows are not guaranteed to stem from any real traffic. Consider instead a more pragmatic attack scenario where the adversary slows the attack speed by increasing the time between the packets sent. To evaluate the potential impact of this adaptive attack on NetSentry, we first pre-process both CSE-CIC-IDS2018 and CIC-IDS-2017 datasets as follows: (1) we group the packets belonging to malicious activity in the `pcap` traces into flows; (2) in each flow, we alter the timestamps of the attacker's packets by expanding the time gap between the previously received/sent packet and the

		Test (CSE-CIC-IDS-2018)			
	m	1	2	4	8
Train (IDS-2018)	1	0	-0.08%	-0.02%	-0.02%
	2	-0.32%	0	-0.16%	-0.24%
	4	-0.35%	0	0	-0.01%
	8	-0.09%	+0.2%	0	0
		Cross Test (CIC-IDS-2017)			
	m	1	2	4	8
Train (IDS-2018)	1	0	-0.05%	-0.11%	-0.58%
	2	+0.1%	0	+0.2%	-0.39%
	4	0	-0.01%	0	-0.35%
	8	-0.49%	-0.56%	-0.11%	0

Table 5.4: Percentage Error wrt. F1 score. Attacker's packets slowed down by factors $\{1,2,4,8\}$. $m = 1$ for original timing.

current one, by a fixed multiplier $m \in \{1,2,4,8\}$ (packets are not delayed if $m = 1$); and (3) we alter the timestamps of the victim's packets to ensure the time gaps between these and the attacker's packets still match those in the original flows.

PortScan attacks are excluded from both datasets because the majority only consists of 1–2 packets, and applying the logic above will not change their timestamps at all. We choose a set of multipliers $m \in \{1,2,4,8\}$, where the attacker's packets are not delayed if $m = 1$. As such, We obtain three altered versions of the CSE-CIC-IDS2018 and CIC-IDS-2017 dataset. Each variant of CSE-CIC-IDS2018 is split into a training set (70% of samples) and a test set (30%), and we augment all the training sets as detailed in 5.2.2, then retrain the Bi-ALSTM. The altered CIC-IDS-2017 datasets are used for cross evaluation. We measure the Percentage Error (PE) with respect to the F1 score, to understand to what extent the model would degrade when facing malicious traffic that is purposely slowed down by different factors, to attempt evasion. Formally, PE wrt. F1 score is defined as:

$$PE_{i,j}^{F1} = \frac{F1_{i,j} - F1_{i,i}}{F1_{i,i}} \times 100\%,$$

where the first subscript denotes the multiplier $m = i$ applied in the dataset used for model training, and the second subscript to slow-down factor in the set used for testing. As shown in Table 5.4, we find that the maximum PE on the CSE-CIC-IDS2018 is never above 0.35% and the maximum PE on CIC-IDS-2017 is below 0.58%. This demonstrates that *manipulating the attack timing has no effective impact on the detection performance of the proposed NetSentry.*

Model	MACs(k)	Parameters(k)	Edge GPU (Mflow/s)
MLP	5.7	5.8	41.4
CNN	3.2	1.5	73.7
Autoencoder	10.3	10.6	22.9
OC-NN	5.2	5.2	45.4
Kitsune	0.7	0.8	337.1
DAGMM	5.3	5.4	44.5
Bi-LSTM	102.2	100.7	2.3
CNN-Bi-LSTM	116.8	112.7	2.0
Bi-ConvLSTM	51.8	2.7	4.5
Bi-ALSTM	66.8	41.4	3.5

Table 5.5: Computation overhead in terms of MACs per inference instance and number of parameters for each model. The last column presents the number of flows (in millions) an edge GPU (Jetson Nano) can process per second.

5.3.9 Computational Overhead

While NetSentry is primarily designed as an offline NIDS, employing it for online NID is also feasible. Table 5.5 details the MACs the benchmark models and our Bi-ConvLSTM/-ALSTM structures require for a single traffic flow inference, as well as their number of parameters. (CNN-)Bi-LSTM are the most computationally expensive, given that multiple fully-connected layers are embedded in the LSTM unit. In contrast, *Bi-ConvLSTM/-ALSTM are relatively lightweight, both involving fewer computations and parameters*. Deploying NetSentry as an online system next to routers or organizational gateways equipped with a GPU or TPU should thus be straightforward.

Given that edge AI platforms are now available, e.g., Nvidia Jetson Nano [117], running NetSentry on constrained small-business/ home routers is within reach. Results in Table 5.5 reveal that Bi-ConvLSTM/-ALSTM can handle 4.5/3.5 Mflows per second, which confirms our practicality assessment.

Another key merit of NetSentry is that the system inherently analyses consecutive traffic between pairs of hosts, which is easy to integrate into an Intrusion Prevention System (IPS), without the need for collecting statistics of potentially malicious hosts until reaching full confidence about decisions to enforce. Recall that our system directly gives prediction results about the traffic flows generated between two hosts during a short interval, offering comprehensive contexts to the IPS with low FP risks. Dynamic firewall rules can also be updated effortlessly, since the atomic processing input of NetSentry originates from the same pair of hosts.

5.4 Summary

ML techniques are increasingly adopted to tackle ever-evolving high-profile network attacks, including DDoS, botnet, and ransomware, due to their unique ability to extract complex patterns hidden in data streams. These approaches are however routinely validated with data collected in the same environment, and their performance degrades when deployed in different network topologies and/or applied on previously unseen traffic, as we uncover. This suggests malicious/benign behaviors are largely learned superficially and ML-based NIDS need revisiting, to be effective in practice. In this chapter we dived into the mechanics of large-scale network attacks, with a view to understanding how to use ML for NID in a principled way. We revealed that, although cyberattacks vary significantly in terms of payloads, vectors and targets, their *early stages*, which are critical to successful attack outcomes, *share many similarities and exhibit important temporal correlations*. Therefore, we proposed to treat NID as a time-sensitive task and introduced NetSentry, perhaps the first of its kind NIDS that builds on Bi-ALSTM, an original ensemble of sequential neural models, to detect network threats before they spread. We cross-evaluated NetSentry using two practical datasets, training on one and testing on the other, and demonstrated F1 score gains above 33% over the state-of-the-art, as well as up to $3\times$ higher rates of detecting attacks such as XSS and web bruteforce. Further, we put forward a novel data augmentation technique that boosts the generalization abilities of a broad range of supervised deep learning algorithms, leading to average F1 score gains above 35%. Lastly, we shed light on the feasibility of deploying NetSentry in operational networks, demonstrating affordable computational overhead and robustness to evasion attacks.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis makes key contributions in solving network security and privacy issues from the perspective of senders, intermediate links and recipients. Starting with Android OS privacy analysis, we disclosed that despite the existence of GDPR and other data protection laws, customized Android OS would still collect private information from users who are not given a choice to disable such features. Besides, regional differences exist in terms of private data transmission practices. Android OS distributions in the EU are relatively compliant to the law, whereas their Chinese counterparts collect a broader range of information and transmits to more diverse endpoints.

Moreover, the thesis presented Amoeba, an RL-based adversarial attack against ML-supported censorship. Amoeba is a full black-box attack which does not need the architecture and weights of the ML censoring models, but can progress through observing the classification results and crafting flows that disguise the ML engines. We demonstrated the efficacy of Amoeba in network environments that suffer from different degrees of delay and noisy feedback. Besides, Amoeba can achieve the same level of ASR as other white-box attacks.

As a major component in the network architecture, ISPs or organizational gateways processes countless network flows on a daily basis, many of which are large-scale network attacks and thus should be blocked at an early stage, to prevent potential upcoming damages. We proposed NetSentry, a NIDS that specifically targets the early stage of the network attacks, and which exhibits excellent classification performance and promising generalization ability under different network environments, outperforming existing state-of-the-art algorithms and systems.

6.2 Future Research Perspectives

Safeguarding network security and privacy requires efforts from different angles, as the current network architectures are already extremely heterogeneous. Following the contributions of this thesis, we identify a number of important research issues that remain unsolved, which I summarize in this section.

6.2.1 Privacy Practices in the iOS Ecosystem

Android is a relatively open mobile OS, both in terms of the ecosystem itself and the vendors' attitudes towards root permission management. While some companies, such as Huawei, have become more conservative about unlocking the bootloader to ensure the security and integrity of their devices, we were still able to identify devices designed by popular brands that support unlocking and rooting the phones. This has resulted in the privacy analysis presented in Chapter 3. In comparison, for iOS devices, jailbreaking is necessary in order to acquire root permissions. However, the associated techniques are often lagging, typically by several months, behind the release of the latest iOS version. This leads to delays in examining the privacy practices of the iOS system. The latest analysis of the data transmission is conducted on iOS 13.6.1 [90], whereas major privacy-related updates were introduced in later versions, such as the permission control of the clipboard and private relay. The extent of private information collection in newer devices compared to previous systems remains unclear. Furthermore, although Apple only releases one uniform version of iOS for global users, we notice regional difference still exist due to local regulations. For example, Voice over Internet Protocol (VoIP) is not supported in China, which results in the blocking of Apple's Callkit; China Personal Information Protection Law stipulates that personal information should be stored within the country, leading to iCloud in mainland China being operated by GCBD (AIPO Cloud (Guizhou) Technology Co. Ltd). Analyzing whether these regional differences could impact privacy standards is also of significant importance.

6.2.2 ML-friendly Representation of Network Flows

The application of DL in the network domain has been underway for several years, demonstrating considerable success. However, the representation of network flows, i.e., the atomic input to ML models, is still not unified and possibly the most appropri-

ate one is yet to be discovered. Three popular flow representations are listed below:

1. In the WF domain, popular representations of network flows are using raw packet and IAT sequences [149, 133, 119];

2. ML censorship tends to extract statistical features including packet size entropy, burst information, and timing information [8, 170];

3. Apart from statistical ones, NIDS also leverage protocol information and packet properties (flags) when extracting features [40, 80].

Feature engineering requires the experience of human experts and the efficacy of specific features may also be determined by the actual task, resulting in various feature sets being used simultaneously. However, existing representations still appear imperfect. The raw packet sequence representation records well the size and arrival time of each packet at the micro level, but it would be challenging for a model to learn higher-order information, such as the mean and standard deviation of uni-directional packets. Conversely, feature sets selected by human experts would not entirely describe a flow, especially the order of packets (the first a few packets may leak protocol information). For comparison, in the domain of computer vision and NLP, using RGB inputs and leveraging word embedding have become part of the routine during training. To fully exert the strength of DL and ensure fair comparison for upcoming work, we consider it necessary to explore a ML-friendly representation of network flows that contains detailed information of a flow as well as easily learnable by DL models. In Chapter 4, an autoencoding Seq2Seq module is pretrained to encode packet sequences into a fixed-length, compact format for the subsequent task and achieve promising results. Although not perfect yet, this could be a possible direction to explore and improve on further.

6.2.3 Practical Adversarial Attacks against NIDS and Defenses

We designed Amoeba as an adversarial attack against ML-supported censorship in Chapter 4, but this model may not be applied to attack NIDS directly. This is because

1. NIDS usually have multiple labels during training, meaning that a targeted adversarial framework would be needed, whereas Amoeba aims to attack binary classifiers and there is no difference between targeted and untargeted attacks. Specifically, expanding Amoeba into a target attack requires the model to progress

through a vector of rewards at each timestep:

$$\{r_{c1}, r_{c2}, ..., c_{cn}\},$$

where each r_{ci} represents the classification score of class i generated by the classifier. The algorithm that attempts to morph a source flow with class a to class b must ensure the reward vectors are close to 0 at every index except for the value at index b.

2. An attacker is only capable of altering the packet sizes and the packet sending gaps on one side of the connections, whereas in the threat model of censorship circumvention, both side of the network stack are accessible.

3. Unlike circumventing censorship, the early stage of network attacks aims to discover victim targets and involves an intensive probing, scanning, bruteforcing and fuzzing in a short time period as summarized in Section 5.1, meaning that the adversarial attack should have little data and time overhead, and consume negligible computing resources in order to fit the nature of network attacks.

With these major differences in the threat model, a more flexible and more lightweight RL algorithm is to be explored. Accordingly, the defenses should be studied as a countermeasure to the potential adversarial attacks. In Section 5.3.8, we studied the impact of changing packet arrival intervals on the performance of NIDS, but a practical adversarial attack, if developed, may alter the arrival intervals and packet sizes with a certain strategy instead of simply multiplying the time gaps with a preset factor. The efficacy of NetSentry remains unknown under the threat of future attacks.

6.2.4 Multitask Adversarial Reinforcement Learning against ML Censorship

Amoeba attacks ML-based network flow classifiers, but censorship in reality is achieved by a complex system. As shown in [172], separate self-implemented network stacks may exist in GFW, with a set of traditional censoring techniques documented in Section 2.2. It is likely that Amoeba can fool an ML module but would not pass the entropy test since the packet sizes generated fall in a small interval, making the entire entropy smaller than that of benign flows. It is thus necessary to develop multitask adversarial reinforcement learning in which rewards are composed by the results of different censoring techniques, such as Shannon entropy estimator, entropy-distribution test, or

different ML models, so that the adversarial attack can bypass the entire censorship system instead of just a single building block.

6.2.5 Towards Nash Equilibrium of Generating Adversarial Network Flows

The success of Amoeba would escalate the arms-race between censorship and circumvention techniques. The censors in the real world may collect Amoeba-generated flows and train the underlying models to detect them, nullifying the flow generation policy. As a consequence, modelling the update of censoring classifiers and restarting training Amoeba is crucial. This task appears similar to a GAN in which a generator and a discriminator are alternatively trained until reaching Nash Equilibrium. However, the generator is trained in the way of supervised learning, and Amoeba is a RL-based algorithm. It is always challenging for RL to converge, and therefore alternative training can be time-consuming and unstable. Under such a scenario, whether Nash equilibrium would be reached or the generator/discriminator just alternatively conquers the other model is worth studying, in order to shed light on the future of this arms-race.

Appendix A

Summary of Traffic Collected From EU/Global Firmware

A.1 Xiaomi

A.1.1 Xiaomi Packages

1. The handset sends the following device identifiers to `tracking.intl.miui.com`: (i) the device IMEIs, (ii) the Security DeviceID (from service miui.sedc, SecurityDeviceCredentialManager), (iii) an MD5 hash of the device Wi-Fi MAC address, (iv) the Google adid/rdid/gaid advertising ID, (v) the device VAID, (vi) the device CPUID. The IMEIs, Security DeviceID and Wi-Fi MAC address are all long-lived device identifiers that persist across a factory reset. The Google advertising ID changes on a factor reset but can be relinked to the device by Google (long-lived device identifiers such as the hardware serial number, IMEI and Wi-Fi MAC address are recorded by Google sent in connections along with the Google advertising ID). The data shared in connections to `tracking.intl.miui.com` therefore allows persistent, long-lived linking of the device data collected by Xiaomi and Google. The VAID value changes upon a factory reset, and in that sense seems akin to the Google Android ID.

2. The Google advertising ID is sent in connections to: `r.register.xmpush.global.xiaomi.com`, `api.ad.intl.xiaomi.com`, `privacy.api.intl.miui.com` and, as already mentioned, `tracking.intl.miui.com`.

3. Long-lived device identifiers that are hashes of the IMEI and Security DeviceID

are sent to: update.intl.miui.com, api.sec.intl.miui.com. The Security DeviceID is sent to find.api.micloud.xiaomi.net.

4. Detailed telemetry of user interaction with the device is sent to tracking.intl.miui.com. Note that this occurs even though the 'User Experience Program", "Send diagnostic data automatically" and 'Personalized ad recommendations" options in the device Xiaomi settings are set off (Google "Usage and Diagnostics" is set off too). This telemetry reveals, for example, the start and end time of phone calls. It also reveals user interactions with the device privacy and permissions settings, amongst others.

5. The Xiaomi com.miui.msa.global, com.xiaomi.discover and com.mi.android.globalFileexplorer system apps log handset activity using Google Analytics and Crashlytics.

6. Details of installed software are sent to tracking.intl.miui.com and global.market.xiaomi.com.

7. When a SIM is inserted into the handset the SIM IMSI (which uniquely identifies the SIM) is sent in connections to tracking.intl.miui.com. In addition, SIM details are sent to Google as observed in previous studies [89].

8. When making/receiving a phone call details of user interaction with the dialer app are sent to tracking.intl.miui.com. In addition, when making an outgoing call the phone number dialed is sent to Google at dialercallinfolookuppa.googleapis.com, and similarly when receiving a call the incoming number is also sent to dialercallinfolookuppa.googleapis.com.

9. When sending a text, the messaging app com.google.android.apps.messaging uses Google Analytics to log user interaction, including screens/activities viewed plus duration and timestamp.

10. When browsing the Settings app connections are made to privacy.mi.com, privacypolicy.truste.com, cdn.cnbj1.fds.api.miimg.com, graph.facebook.com, global.market.xiaomi.com, pagead2.googlesyndication.com. Details of user interactions with the Settings app are sent to /tracking.intl.miui.com.

A.1.2 Preinstalled Non-Xiaomi Packages

1. *Mobile Operator*. The particular handset used in these measurements was bought secondhand online and appears to originally be from German mobile operate Deutsche Telekom. The pre-installed system apps include `de.telekom.tsc`. This app sends device details and telemetry to `moaps.tmo.net` (this domain appears to be owned by Deutsche Telekom).

2. *Facebook*. Handset data is sent to `www.facebook.com` and `graph.facebook.com`. Data sent includes device details, device accelerometer and rotation measurement data, the handset IP address and the mobile carrier name. The ASHAS value sent along with this data appears to act as a long-lived device identifier (it persists across factory resets).

3. *Google* The following pre-installed Google system apps were observed to send data to Google.

   (a) Google Play Services and Google Play store make many connections to Google servers. These share persistent device and user identifiers with Google including the device hardware serial number, SIM IMEI, Wi-Fi MAC address, SIM IMSI and phone number, user email (when logged in). A substantial quantity of data is sent, in particular, to `play.googleapis.com/vn/log/batch`, `play.googleapis.com/play/log` and `www.googleapis.com/experimentsandconfigs`.

   (b) Google YouTube sends device data, including persistent identifiers and the Google adid/rdid advertising identifier and the AndroidID, to `www.googleadservices.com` and `youtubei.googleapis.com`. YouTube also uses Google Analytics to log events, and presumably also user interaction.

   (c) Connections are periodically made to `www.google.com/complete/search`. These are associated with the `com.google.android.googlequicksearchbox` searchbar app embedded in the handset UI and send a cookie which acts to link these connections to persistent device and user identifiers. Less frequent connections are made to `www.google.com/m/voice-search/down` that contain what appears to be a persistent device identifier. The `com.google.android.googlequicksearchbox` app also sends telemetry data to Google Analytics.

(d) The system messaging app `com.google.android.apps.messaging` uses Google Analytics to log user interaction, including screens/activities viewed plus duration and timestamp. For example, when sending a text the user interactions with the WelcomeActivity, HomeActivity and ConversationActivity are logged, and a `FIRST_MESSAGE_SENT` event is also recorded.

(e) When logged in to a Google account, connections are made to `mail.google.com/mail/ads`, `inbox.google.com/sync` and `www.googleapis.com/calendar` that send identifiers linked to the device and user account. Note that account login was carried out via the Google Play app only. Syncing of gmail, contacts, calendar took place without the user being asked or opting in.

(f) When logged in to a Google account, connections are also made to `instantmessaging-pa.googleapis.com`, `people-pa.googleapis.com`, `footprints-pa.googleapis.com`. It's not clear what the purpose of these connections is or what data is sent.

(g) When location is enabled, additional connections are made to `lamssettings-pa.googleapis.com`.

(h) The `com.google.android.dialer` app makes connections to `businesscalls.googleapis.com`. As already noted, when making an outgoing call the phone number dialed is sent to `dialercallinfolookuppa.googleapis.com`, and similarly when receiving a call the incoming number is also sent to `dialercallinfolookuppa.googleapis.com`.

Note that none of these apps, apart from the dialer app, were opened on the device. No popup or request to send data was observed.

4. *Google Analytics*. Several pre-installed system apps log handset activity using Google Analytics and Crashlytics. These include: `com.miui.msa.global`, `com.xiaomi.discover`, `com.mi.android.globalFileexplorer`, `com.google.android.apps.walletnfcrel`, `com.google.android.dialer`, `com.google.android.apps.maps`, `com.google.android.gm`, `com.google.android.calendar`, `com.google.android.youtube`, `com.google.android.apps.messaging`, `com.google.android.gms`, `com.android.vending`.

A.2 Samsung

A.2.1 Samsung Packages

1. The handset sends the following device identifiers to `api.omc.samsungdm.com` and `dir-apis.samsungdm.com` in a way that links them all together: hardware serial numbers (both ro.serialno and ril.serialnumber in getProp), IMEI of both SIM slots and Samsung device/consumer ID. In addition a Google Firebase authentication token is sent to `api.omc.samsungdm.com`, allowing linkage of data collection by Samsung and Google Firebase. A connection to `dir-apis.samsungdm.com` sends cell tower identifiers (LAC and UCID) that reveal the approximate device location.

2. Details of installed software are sent to `os-api.gos-gsp.io/`, which appears to be a Samsung domain associated with a software configuration service (gos may be an acronym for "game optimisation service"). The service returns app settings such as `allow_more_heat`, `boost_launch`, `enableCpuMinFreqControl`).

3. Device details, including the hardware serial number, are sent to `api.gras.samsungdm.com`. Also sent is a Google Firebase authentication token associated with com.samsung.android.sdm.config.

4. Device and app identifiers are sent to `capi.samsungcloud.com`. This appears to be associated with the Samsung Cloud service.

5. What appears to be hashes of the IMEIs of both SIM slots and of the hardware serial number are sent to `eu-kaf.samsungknox.com`, `gslb.secb2b.com/KnoxGSLB/v2/lookup/knoxguard`. Binary messages are also sent to `eu-kaf.samsungknox.com`, the contents of which are unclear (they base64 decode to binary in what seems to be a proprietary format). These connections appear to be associated with the Samsung Knox service.

6. The IMEI's of both SIM slots, the hardware serial number and the Samsung device/consumer ID are sent to `www.ospserver.net/`, which appears to be a Samsung server. Cell tower identifiers are sent that reveal the approximate device location. The Firebase authentication token for app `com.wssyncmldm` is also sent, allowing linkage of data collection by Samsung and Google Firebase.

7. The Google adid/rdid advertising ID is sent to `spapi-prd.samsungrs.com/AdConfiguration`. The connection seems to associated with the app `com.samsung.android.dynamiclock`.

8. Other Samsung servers to which data is sent include: `fota-apis.samsungdm.com`, `ota-cloud-dn.ospserver.net`, `hub-odc.samsungapps.com/`, `as.samsungapps.com/`, `dms.ospserver.net`, `sdk.pushmessage.samsung.com`.

9. Samsung's system apps that log handset activity using Google Analytics include: `com.samsung.android.app.omcagent`, `com.samsung.android.app.simplesharing`, `com.samsung.android.authfw`, `com.samsung.android.bixby.agent`, `com.samsung.android.kgclient`, `com.samsung.android.mobileservice`, `com.samsung.android.rubin.app`, `com.samsung.android.themestore`, `com.sec.android.app.billing`, `com.sec.android.app.samsungapps`, `com.wssyncmld`, `com.samsung.android.game.gamehome`. The messages sent to Google/Firebase Analytics are encoded as protobufs. We decoded these by reconstructing the protobuf definition from the decompiled Firebase code. Examples are shown below, but the messages log user interaction, including screens/activities viewed plus duration and timestamp.

10. When a SIM is first inserted into the handset following a factory reset a connection is made to `api.omc.samsungdm.com/v5/api/device/simChange` that sends the hardware serial number, Samsung device/consumer ID, the mobile operator MNC/MCC identifiers and the operator name.

11. When browsing the Settings app connections are made to `ie-odc.samsungapps.com`, `auth2.samsungosp.com`, `us-cd-gpp.mcsvc.samsung.com`, `us-rd.mcsvc.samsung.com`. The `com.samsung.android.themestore` and `com.samsung.android.samsungpass` apps send telemetry to Google Analytics and com.samsung.android.samsungpass calls `firebaseremoteconfig.googleapis.com`. The Google adid/rdid advertising identifier is sent to `us-api.mcsvc.samsung.com/`.

A.2.2 Preinstalled Non-Samsung Packages

1. *Mobile Operator SFR/Altice*. The particular handset used in these measurements was bought secondhand online and appears to originally be from French mobile

operator SFR/Altice France. The pre-installed system apps include com.sfr.android.sfrjeux and com.altice.android.myapps. Both make use of Google Analytics and custom telemetry is also sent to sun-apps.sfr.com/reportusage. The app com.sfr.android.sfrjeux and com.altice.android.myapps apps send device details and a Firebase authentication token to https://sun-apps.sfr.com (so linking data collection by SFR/Altice and Google Firebase) as well as what appears to be a persistent device identifier. It also appears to attempt to transmit the Wi-Fi SSID. The app com.altice.android.myapps additionally uses the Firebase Crashlytics and Remote Configuration services.

Note that neither of the apps were ever opened on the device, and no popup or request to send data was observed.

2. *Google*. The following pre-installed Google system apps were observed to send data to Google.

   (a) Google Play Services and Google Play store make many connections to Google servers. These share persistent device and user identifiers with Google including the device hardware serial number, SIM IMEI, Wi-Fi MAC address, SIM IMSI and phone number, user email (when logged in). A substantial quantity of data is sent, in particular, to play.googleapis.com/vn/log/batch, play.googleapis.com/play/log and www.googleapis.com/experimentsandconfigs.

   (b) Google YouTube sends device data, including persistent identifiers and the Google adid/rdid advertising identifier and the AndroidID, to www.googleadservices.com and youtubei.googleapis.com. YouTube also uses Google Analytics to log events, and presumably also user interaction.

   (c) Connections are periodically made to www.google.com/complete/search. These are associated with the searchbar embedded in the handset UI and send a cookie which acts to link these connections to persistent device and user identifiers. Less frequent connections are made to www.google.com/m/voice-search/down that contain what appears to be a persistent device identifier.

   (d) The com.google.android.googlequicksearchbox app is associated with the search bar embedded in the handset UI. It sends telemetry data to Google Analytics that logs user interaction (screens/activities viewed plus

duration and timestamp, etc.).

(e) When logged in to a Google account, connections are made to `mail.google.com/mail/ads`, `inbox.google.com/sync` and `www.googleapis.com/calendar` that send identifiers linked to the device and user account. Note that account login was carried out via the Google Play app only. Syncing of gmail, contacts, calendar took place without the user being asked or opting in.

(f) When logged in to a Google account, connections are also made to `instantmessaging-pa.googleapis.com`, `people-pa.googleapis.com`, `footprints-pa.googleapis.com`. It's not clear what the purpose of these connections is or what data is sent.

(g) When location is enabled additional connections are made to `lamssettings-pa.googleapis.com` and `mobilenetworkscoring-pa.googleapis.com/v1/GetWifiQuality`. The connection to `mobilenetworkscoring-pa.googleapis.com/v1/GetWifiQuality` hashes of the Wi-Fi MAC addresses of nearby access points, used to query a network quality database to determine the best Wi-Fi network to connect to.

(h) Connections are made to `www.gstatic.com/commerce/wallet/` but were not observed to contain persistent identifiers.

(i) Google Chrome makes connections to Google servers. These connections are consistent with previously documented behavior [91].

Note that none of these apps were opened on the device, and no popup or request to send data was observed.

3. *Microsoft*. The pre-installed Microsoft system apps connect to `mobile.pipe.aria.microsoft.com` and `app.adjust.com` (which appears to be a third-party analytics company, their website says "Adjust offers a number of analytics tools designed to give you the deepest insight into your user interaction, your marketing channels, and your campaign performance"). The data sent to `mobile.pipe.aria.microsoft.com` includes device hardware and software details together with persistent device identifiers `DeviceInfo.Id`, `DeviceInfo.SDKUid`, `cai.device.id` and `TenantId`. The document `https://docs.microsoft.com/en-us/deployoffice/privacy/required-diagnostic-data` states:

(a) `DeviceInfo.Id` - A unique device identifier to help us detect device-specific issues

(b) `DeviceInfo.SDKUid` - The device unique identifier (similar to `DeviceInfo.Id`)

The data sent to `app.adjust.com` includes device details, the Google adid/rdid advertising ID of the handset and what appears to be a persistent device identifier that acts to link connections together. Connections are also made to `config.edge.skype.com`, `skyapi.live.net`, `oneclient.sfx.ms`. Note that no Microsoft apps were ever opened on the device, and no popup or request to send data was observed.

4. *LinkedIn*. A first connection is made to `www.linkedin.com/mob/tracking` that responds by setting bcookie, bscookie and lidc cookies. The LinkedIn document `https://www.linkedin.com/legal/l/cookie-table` says:

   (a) bcookie: Browser Identifier cookie to uniquely identify devices accessing LinkedIn to detect abuse on the platform

   (b) bscookie: Used for saving the state of 2FA of a logged-in user

   (c) lidc: To optimize data center selection

   These cookies are resent in later requests to `www.linkedin.com/li/track`, along with `trackingId` values.

5. *Hiya*. Connections are made to `samsung-directory.edge.hiyaapi.com/v3/trackevents`. This appears to be associated with a third-party call management service, with connections made when making/receiving a call. Data sent includes the Google adid/rdid advertising ID, an X-Hiy-Installation-User-ID and an authorization token that when decoded contains hin and hui values that appear to be persistent device identifiers.

6. *Google Analytics*. Several pre-installed system apps log handset activity using Google Firebase Analytics. These include: `com.google.android.apps.maps`, `com.microsoft.skydrive`, `com.android.vending`, `com.google.android.googlequicksearchbox`, `com.sfr.android.sfrjeux`, `com.altice.android.myapps`

A.3 Realme

A.3.1 Realme Packages

1. The handset sends DUID/VAID, OAID and `device_id` and identifier values in connections to `shorteuex.push.heytapmobile.com`. It also sends the Google Firebase ID and authentication token associated with the `com.heytap.mcs` app. The OAID value and Google adid/rdid Advertising Id (GAID) are sent in connections to `adx-f.ads.heytapmobile.com`, linking data collection by Google and Realme. The encrypted `device_id` value is sent in MQTT encoded messages over a TCP connection. All of these identifiers change upon a factory reset of the device. However, in connections to `httpdns-euex-push.heytapmobile.com` a long-lived deviceId value is sent that persists across factory resets, along with the VAID value. While the VAID value changes on a factory reset, because the deviceId value remains unchanged the new VAID value can be relinked to the device. Since the VAID is sent along with the DUID/VAID, OAID, `device_id` and Firebase ID/token values, these also can be relinked. Since the OAID and GAID are sent together, the GAID can be relinked to the device.

   The domain `shorteuex.push.heytapmobile.com` does not appear to be registered to Realme. The SSL cert offered by `shorteuex.push.heytapmobile.com` indicates that it is operated by Bravo Unicorn Pte. Ltd, Singapore, a private holding company. This matches the company information at `https://www.heytap.com/en/about-us.html`.

2. The handset also connects to `icosaeu.coloros.com`, `ifruseu.coloros.com`, `classifyeu.apps.coloros.com` and `ifotaeu.realmemobile.com`. These domains are associated with Oppo, the parent company of Realme, and Realme. In connections a guid value and a registrationId value are sent. The registrationId changes upon a factory reset. However, the registrationId value is obtained in the response to a request to `shorteuex.push.heytapmobile.com` that sends the `device_id` value, and so as noted above can be relinked to the device. The guid value does not change upon a factory reset.

3. The encrypted guid value and handset IMEI are sent to `esa-reg-eup.myoppo.com/ImeiEncryptRegister.ashx` by app `com.coloros.activation`.

A custom reversible encryption scheme is used, and we have confirmed that the sent values can be decrypted to recover the guid and device IMEI.

4. Details of installed apps are sent to `shorteuex.push.heytapmobile.com` and to `icosaeu.coloros.com`.

5. The app `com.heytap.mcs` registers with Google's c2dm (Cloud to Device) service, and the Firebase ID and authorization token returned by Google are sent to `shorteuex.push.heytapmobile.com/api/push/device/updateFcmToken`. The app also registers with Google Analytics but was not observed to send data to it.

6. A connection to `httpdnseuexpush.heytapmobile.com/getdns/v1` returns a list of IP addresses, followed by occasional TCP/MQTT connections made to one IP in the list. The MQTT message ClientId acts as a persistent identifier. The MQTT message payload is a protobuf encrypted with a custom AES algorithm, but perhaps due to a programming error does not seem to include identifiers or sensitive information.

A.3.2 Preinstalled Non-Realme Packages

1. *Microsoft*: A connection is made to www.bing.com immediately after factory reset, in which the response sets a number of cookies. However, the cookies appear to be scrubbed since they are not resent in later connections, and the cookie values in the response change.

2. *Google*: The following google connections are observed:

   (a) YouTube sends device data, including device ID (a unique identifier used by YouTube only) and Google gaid to `youtubei.googleapis.com` and `www.googleadservices.com`.

   (b) Occasionally there are connections made to `https://www.google.com/m/voice-search/up` and `https://www.google.com/m/voice-search/down` with a persistent identifier `31f13aa1-fbb2-4302-bb02-b03bd076c118h`. These connections are initiated by `com.google.android.google-quicksearchbox`.

(c) After logged into google account, a range requests are made to ask for the permission of google services, including googleplay, googlenow, OAuthLogin, memento, drive, mobileapps.doritos.cookie, gcm, numberer, firebase.messaging, userinfo.email, tachyon, sierra, webhistory, peopleapi.readwrite, cryptauth, playatoms, android_checkin, gmail.publisher_first_party, reminders, gmail.ads, chat, gmail.full_access, login_manager, experimentsandconfigs, userlocation.reporting, tasks, meetings, hangouts, notifications, cclog.

(d) Afterward, connections are made to sync up calendars, tasks, mails, contacts, passwords from `calendarsync-pa.googleapis.com`, `tasks-pa.googleapis.com`, `inbox.google.com`, `people-pa.googleapis.com` and `chromesyncpasswords-pa.googleapis.com` respectively. Note that attachments in the mailbox are also downloaded.

(e) Google makes connections to get promotion text of its app, *keep note*.

(f) When location is turned on, there is a request sent to `www.googleapis.com/geolocation/v1/geolocate` to acquire the geolocation, most likely by the IP address of this handset, but it is unclear which pre-installed app made this request.

(g) com.google.android.googlequicksearchbox logs the activity name through `app-measurement.com/a`

(h) When a SIM is inserted into the handset SIM details are sent to Google as observed in previous studies [89].

A.4 Huawei

A.4.1 Huawei Packages

1. The handset hardware serial number is sent to `query.hicloud.com` and `config serverdre.platform.hicloud.com` (domains that appear to be registered to Huawei). A device certificate is also sent to `query.hicloud.com` that contains a CN value that appears to be a device identifier. When a SIM is inserted the mobile carrier ID is sent to `query.hicloud.com`.

2. Device details are sent to `servicesupport.hicloud.com`, together with a `userId` value.

3. Connections made by com.huawei.himovie.overseas:

   (a) *Daily Motion*. The app makes connections to www.dailymotion.com that send the device Google adid/rdid advertising ID as a parameter in the URL. The response sets several cookies plus an HTML document. The HTML contains embedded device/user identifiers, including the Google adid/rdid and client_id, client_secret values.

   (b) Following the connection to www.dailymotion.com, connections are made to a sequence of other third-party content servers, all with a referrer header value set to the original request to www.dailymotion.com, so presumably these are prompted by processing of the HTML document sent by www.dailymotion.com. Since the handset Google adid/rdid is embedded in the URL, this is shared with all of these third-party content servers, including static1.dmcdn.net, imasdk.googleapis.com, s0.2mdn.net, pagead2.googlesyndication.com.

   (c) Connections are also made to pebed.dmevent.net, a domain registered to Daily Motion. These connections send the handset Google adid/rdid in the referrer header, but also device details and instance_uuid, id values in the POST body that appear to be device/user identifiers, including one of the cookie values set in the response to the request to www.dailymotion.com. This connection appears to send analytics/telemetry.

   (d) Connections are made speedtest.dailymotion.com. These connections send the handset Google adid/rdid in the referrer header, as well as the cookie set by the response to the request to www.dailymotion.com. Similar connections to dmxleo.dailymotion.com is observed.

4. Connections made by com.huawei.systemmanager:

   (a) *Avast App Scanning Service*. The com.huawei.systemmanager app, which has the package com.avast.android.sdk embedded within it, makes connections to auth.ff.avast.com, shepherd.sb.avast.com, apkrep.ff.avast.com, analytics.ff.avast.com/receive3. This appears to be associated with an apk scanning service (e.g., connections are reliably generated whenever an apk is installed). The data sent in these connections are AES encrypted protobufs. When decrypted and decoded as a protobuf, periodic authentication connections to shepherd.sb.avas

t.com/V1/MD send a hash of the device android_id and an ABUID value that also acts as a persistent identifier. When decrypted and decoded as a protobuf, connections to apkrep.ff.avast.com/apk/reputation and apkrep.ff.avast.com/apk/touch send app details (filename, signing cert). The payload also appears to contain persistent device identifiers, and the encryption scheme used also allows messages from the same device/session to be linked together.

(b) *Qihoo 360.* The com.huawei.systemmanager app also has the packages com.qihoo.cleandroid.sdk, com.qihoo.cleandroid.cleanwx.sdk, com.qihoo.cleandroid.mobilesmart.sdk, com.qihoo.qvssdk, com.qihoo.security.engine, com.qihoo.protection embedded within it. The app makes connections to mvconf.cloud.360safe.com/safeupdate and mclean.cloud.360safe.com/CleanQuery periodically (every 1-2 days), and a connection to aiclean.us.cloud.360safe.com/video/clean is observed once. The messages are encrypted using a JNI C library and are encoded in a custom binary format. When decrypted it can be seen that the connections contain a persistent device identifier (labelled "IMEI" but the value is generated randomly when the app is first started after a factory reset).

5. When sending a text, the messaging app com.google.android.apps.messaging uses Google Analytics to log user interaction, including screens/activities viewed plus duration and timestamp.

A.4.2 Preinstalled Non-Huawei Packages

1. *Microsoft*

   (a) The handset uses Microsoft's com.touchtype.swiftkey keyboard. This sends data to in.appcenter.ms/logs, bibo.api.swiftkey.com with a persistent install-ID and device details. The in.appcenter.ms/logs connection appears to relate to logging of app errors and sometimes includes stack traces.

   (b) Telemetry data is sent to telemetry.api.swiftkey.com which includes device details and logs events. Data is encoded in gzipped avro format. To decode the avro data the schema was extracted from the app by using

edXposed to execute a getSchema() call on app startup and dumping the (large, about 200KB) response to disk. After decoding using this scheme it can be seen that the data sent includes the Google adid/rdid Advertising ID, a Firebase ID/token and an installId value. User interaction with apps is logged when the keyboard is used within the app, with the app name, number of characters entered and a millisecond accuracy event timestamp sent to `telemetry.api.swiftkey.com`. Such event logging was observed, for example, when using the contacts, messaging, searchbar and notepad apps. Interactions with the keyboard, e.g., opening the clipboard, viewing/modifying the settings, are also logged.

2. *Google*. The following pre-installed Google system apps were observed to send data to Google.

   (a) Google Play Services and Google Play store make many connections to Google servers. These share persistent device and user identifiers with Google including the device hardware serial number, SIM IMEI, Wi-Fi MAC address, SIM IMSI and phone number, user email (when logged in). A substantial quantity of data is sent, in particular, to `play.googleapis.com/vn/log/batch`, `play.googleapis.com/play/log` and `www.googleapis.com/experimentsandconfigs`.

   (b) Google YouTube sends device data, including persistent identifiers and the Google adid/rdid advertising identifier and the AndroidID, to `www.googleadservices.com` and `youtubei.googleapis.com`. YouTube also uses Google Analytics to log events, and presumably also user interaction.

   (c) Connections are periodically made to `www.google.com/complete/search`. These are associated with the `com.google.android.googlequicksearchbox` searchbar app embedded in the handset UI and send a cookie which acts to link these connections to persistent device and user identifiers. Less frequent connections are made to `www.google.com/m/voice-search/down` that contain what appears to be a persistent device identifier. The `com.google.android.googlequicksearchbox` app also sends telemetry data to Google Analytics that logs user interaction (screens/activities viewed plus duration and timestamp, etc.).

   (d) The system messaging app `com.google.android.apps.messaging`

uses Google Analytics to log user interaction, including screens/activities viewed plus duration and timestamp. For example, when sending a text the user interactions with the WelcomeActivity, HomeActivity and ConversationActivity are logged, and a `FIRST_MESSAGE_SENT` event is also recorded.

(e) The `com.google.android.apps.tachyon` logs events using Google Analytics.

(f) When logged in to a Google account, connections are made to `mail.google.com/mail/ads`, `inbox.google.com/sync` and `www.googleapis.com/calendar` that send identifiers linked to the device and user account. Note that account login was carried out via the Google Play app only. Syncing of gmail, contacts, calendar took place without the user being asked or opting in. When logged in to a Google account, connections are also made to `instantmessaging-pa.googleapis.com`, `people-pa.googleapis.com`, `footprints-pa.googleapis.com`. It's not clear what the purpose of these connections is or what data is sent. In addition, the following google services are authenticated: googleplay, android_video, drive.metadata.readonly, drive.labels.readonly, drive.activity.readonly, docs, drive.readonly,peopleapi.readonly, drive.apps, cloudprint, activity, drive.file, gmail.readonly, subscriptions, memento, notifications, spreadsheets, vouchers, discussions, userlocation.reporting, peopleapi.legacy.readwrite, reminders, calendar, playatoms, mobileapps. doritos.cookie, peopleapi.readwrite, OAuthLogin, sierra, webhistory, gmail. full_access, gmail.ads, gmail.locker.read, taskassist.readonly, gmail. publisher_first_party, experimentsandconfigs, tachyon, numberer, firebase. messaging, userinfo.email, gcm, android_checkin, login_manager, cryptauth, notifications.

(g) When location is enabled, additional connections are made to `www.googleapis.com/geolocation/v1/geolocate` and `mobilenetworkscoring-pa.googleapis.com/v1/GetWifiQuality`. The connection to `mobilenetworkscoring-pa.googleapis.com/v1/GetWifiQuality` sends encoded data and it's not clear what the contents are but they include the mobile operator MNC and MCC identifiers.

(h) Google Chrome makes connections to Google servers. These connections

are consistent with previously documented behavior [91].

(i) When a SIM is inserted into the handset SIM details are sent to Google as observed in previous studies [89].

Note that none of these apps were opened on the device, and no popup or request to send data was observed.

A.5 EOS

A.5.1 EOS Packages

1. The only connection made during startup after a factory reset is to `xtrapath 3.izatcloud.net` to download a GPS configuration file from qualcom, no identifiers are sent.

2. When navigating the Settings app a slimstat cookie is set when the privacy policy page is viewed. This seems to be associated with fetching of web page `https://e.foundation/legal-notice-privacy/`.

3. No connections were observed to Google servers.

A.6 Lineageos

A.6.1 Lineageos Packages

1. The handset is considerably "quieter" than the Samsung, Xiaomi, Realme and Huawei handsets. A single connection to `download.lineageos.org` sends device details but contains no identifiers. The remaining connections are associated with Google system apps, which send a similar data content to the Google apps on the Samsung, Xiaomi, Realme and Huawei handsets.

A.6.2 Preinstalled Non-Lineageos Packages

1. *Google*

   (a) Google Play Services and Google Play store make many connections to Google servers. These share persistent device and user identifiers with

Google including the device hardware serial number, SIM IMEI, Wi-Fi MAC address, SIM IMSI and phone number, user email (when logged in). A substantial quantity of data is sent, in particular, to `play.googleapis.com/play/log` and `www.googleapis.com/experimentsandconfigs` (but not to `play.googleapis.com/vn/log/batch`, unlike on the other handsets studied).

(b) Periodic connections are made to `www.google.com/m/voice-search` with a unique identifier in the url path. Less frequent connections are made to `www.google.com/complete/search` in which the headers incorporate a field: *x-client-data*. These are associated with the `com.google.android.googlequicksearchbox` searchbar app. The `com.google.android.googlequicksearchbox` app also sends telemetry data to Google Analytics.

(c) When logged in to a Google account, connections are made to `mail.google.com/mail/ads`, `inbox.google.com/sync` and `www.googleapis.com/calendar` that send identifiers linked to the device and user account. Note that account login was carried out via the Google Play app only. Syncing of gmail, contacts, calendar took place without the user being asked or opting in. In addition, the following services are authenticated: gms, googlenow, OAuthLogin, mobileapps.doritos.cookie, experimentsandconfigs, numberer, firebase.messaging, googleplay, webhistory, tachyon, reminders, userlocation.reporting, android_checkin, peopleapi.legacy.readwrite, cryptauth, login_manager, playatoms, peopleapi.readonly, gcm, notifications, calendar.

(d) Google Chrome makes connections to Google servers. These connections are consistent with previously documented behavior [91].

(e) When a SIM is inserted into the handset SIM details are sent to Google as observed in previous studies [89].

Appendix B

Summary of Traffic Collected From CN Firmware

B.1 Xiaomi

B.1.1 Xiaomi Packages

1. Device details, aaid and oaid are sent to `cn.register.xmpush.xiaomi.com` for device registration after factory reset. aaid and oaid are temporary identifiers. Hashed android ID (temporary), hashed IMEI, cpuid and device details are post to `update.miui.com`.

2. Temporary identifiers including cloudsp_fid and cloudsp_devId are transmitted to `find.api.micloud.xiaomi.net`.

3. Xiaomi device sends a request to `data.mistat.xiaomi.com/key_get`, which negotiates an encryption key (RSA encrypted) and sid with the server. The device further sends requests to `data.mistat.xiaomi.com/idservice/deviceid_get` and `data.mistat.xiaomi.com/mistats/v3` in which the message is encrypted with the key aforementioned and contains the sid, so that the server can look up the associated key for decryption. The message to `/idservice/deviceid_get` incorporates hashed IMEI, MEID, MAC address and serial number, and AAID, android ID and OAID in plaintext. Any connections to `data.mistat.xiaomi.com/mistats/v3` contains the same group of identifiers and also telemetry information that logs the access time of some activities including UsbDebuggingActivity, com.miui.notes.ui.NotesListActivity,

and com.miui.notes.ui. activity.EditActivity.

4. Traffic to `diagnosis.ad.xiaomi.com` and `api.ad.xiaomi.com` contains device details, aaid, oaid, hashed MAC address and IMEIs. Besides, connections to the latter incorporate local IP address, android ID and also the field 'isPersonalizedAdEnable' is set to true despite the fact that all opt-outs are selected during device setup. The traces collected from EU firmware populate this field as false.

5. A request to `api.developer.xiaomi.com` uploads a range of system package names and jar filenames to check update. A smaller batch of Xiaomi-related package names is uploaded to `de.idm.iot.mi.com`. However, the device also sends all the installed package names to `auth.be.sec.miui.com`, which is weird in the sense that the content does not match the domain/endpoint name at all.

6. GUID, OAID and device details are sent to `api.hybrid.xiaomi.com`.

7. The package com.xiaomi.metoknlp initiates a request to a baidu API `api.map.baidu.com` that contains the current geolocation. Note that during setup, Xiaomi provides the option of using location service or not, but disabling it does not turn off the location switch in settings.

8. Traffic to `tracking.miui.com` consists of aaid, oaid, cpuid, hashed MAC, IMEI and devices details. Moreover, the handset collects a series of telemetry and location information which are also posted to this domain. Location-wise, geolocation, MCC, MNC, nearby Wi-Fi name and Wi-Fi's MAC address are transmitted. com.miui.analytics collects telemetry about Xiaomi's preinstalled app, including settings, recording, note, phone, message, and camera. App first launch time, usage start time and end time are logged. The timestamps of calling key activities are logged as well, such as WifiProvisionSettingsActivity, NotesListActivity, EditActivity, Camera, and GrantPermissionsActivity.

B.1.2 Preinstalled Non-Xiaomi Packages

1. *China Mobile SDK*: The handset sends multiple requests to `a.fxltsbl.com` which belongs to a self-registering platform managed by China Mobile. The

requests contain IMEI, MEID, device details, and all the installed apps. If a sim card is inserted, CellId and lac are also populated.

2. *China Unicom SDK*: The handset sends a request to `dm.wo.com.cn:18080` for registration after factory reset, in which device details, IMEI, ICCID, IMSI, phone number, MNC, CellID and LAC are populated.

3. *Baidu*: The device requests `baidu.com` right after factory reset and the response routinely sets a few cookies, but they never appear elsewhere.

B.2 OnePlus

B.2.1 OnePlus Packages

1. Weather app sends OID, VAID, and OAID to oppo server `i6.weather.oppomobile.com`. Geolocation (latitude and longitude) is transmitted in subsequent requests. parameters mcc, ssid and lac exists in the requests but no values are filled in.

2. The handset sends encrypted guid and IMEI to `moa-upload-online.coloros.com`, sends devID to `lang.coloros.com` and transmits openId to `tobcustomize.coloros.com` and `component-ota.coloros.com`. IMEI and openID (=guid) are persistent identifiers while devID changes after factory reset. Although heytapmobi and coloros nominally belong to two companies, they are deeply related in terms of the traffic generated after factory reset. The handset sends device details to `https://data.sms.heytapmobi.com` and receives a list of URLs starting with `https://tedsyncfs.coloros.com/`, which is accessed later.

3. A subset of preinstalled apps are sent to `icosa.coloros.com`, `classify.apps.coloros.com` and `moa-upload-online.coloros.com`.

4. The handset transmits encrypted contents to `log.avlyun.com` with the package name `com.coloros.wifisecuredetect` in the requests. The package would be launched as soon as Wi-Fi is enabled and initiates those requests. The time gap between 'enabling Wi-Fi' and 'posting requests' is extremely small. By the time frida hooked the functions in this package, the requests are already sent.

contents are still unknown. We use EdXposed to hook the process when it is spawned.

5. The handset sends deviceId and vaid to `httpdns.push.heytapmobi.com`, sends duId and ouId to `stg-data.ads.heytapmobi.com`, and sends device details and ouId to `data.sms.heytapmobi.com`. DeviceId is a persistent identifier while the rest of them changes after factory reset. There is no evidence yet that heytapmobi can use deviceId to relink devices after factory reset, because vaid/deviceId never appear in the requests that contain duId or ouId. It is worth noting that in one request to `api-cn.open.heytapmobi.com` a header named openId is added, but the value actually is vaid. Besides, vaid that appears in heytap requests is the same as the one in `i6.weather.oppomobile.com`

6. When a sim card is inserted, observed requests to `sms.ads.heytapmobi.com/new/v5/phones`, `sms.ads.heytapmobi.com/new/api/get_sms_menu` and `log.avlyun.com` with encrypted messages. The phone number and ouID is sent to `sms.ads.heytapmobi.com/new/v5/phones` for registration. Local Wi-Fi name, mac address, device details and AVLUDID are sent to `log.avlyun.com`.

7. When making or receiving a phone call, my mobile number and ouId is post to `sms.ads.heytapmobi.com`. Phone numbers can be considered as a semi-persistent identifier, since in China each number is registered with the user's citizen ID. When receiving a phone call or a text, the caller/sender's phone number is also posted.

8. The handset sends ouId and auId to `u.bot.heytapmobi.com` when opening clock app.

B.2.2 Preinstalled Non-OnePlus Packages

1. *Amap* transmits encrypted contents to `aps.amap.com`, `offline.aps.amap.com`, `apsrgeo.amap.com` and `cgicol.amap.com`. A unique CSID is associated with HTTP requests to `aps.amap.com` but would change upon factory reset. Decrypting the post contents to `apsrgeo.amap.com`, we found that current geolocation is embedded in the message. The messages to `aps.amap.com` cannot be decrypted directly since the package calls a JNI native function for encryption. Instead, we hook an intermediate function to print out all the val-

ues, which contains nearby Wi-Fi names and their MAC addresses. Payload to `cgicol.amap.com` is loaded from a local database with some timestamps.

2. *China Mobile SDK* The handset sends requests to `a.fxltsbl.com` which belongs to a self-registering platform managed by China Mobile. The requests contain IMEI, device details, and deviceId (different from the deviceID sent to heytapmobi). Mac, lac, cellId and iccid exist in the posted json string but are not populated. The requests can be captured right after factory reset without a sim card, but cannot be observed if a sim card is inserted after factory reset.

3. *China Unicom SDK* sends device details and IMEI to `dm.wo.com.cn` for device registration if no sim card is inserted. With a sim card, additional contents including encrypted ICCID, CellID, LAC, IMSI, MISIDN and MNC would be posted, which contains the coarse location information. At the moment that the user press 'reset this device', another request to `dm.wo.com.cn` would be made with IMEI and device details populated.

4. *Sogo* The handset made a GET request to `m.sogo.com` which returns status code 302 and a cookie, but no subsequent requests are observed nor the cookie.

5. *QQ* sends an empty request to `tools.3g.qq.com` which returns cookies, but we never observe them in other connections.

B.3 Realme

B.3.1 Realme Packages

1. Weather app sends OID, VAID, and OAID to oppo server `i6.weather.oppomobile.com`. Geolocation (latitude and longitude) is transmitted in subsequent requests. Fields lac, and imei exist in the requests, but no values are filled in.

2. The handset sends device details, GUID, OUID, statUId, IMEI and telemetry (event logs and error logs) to `dragate.dc.oppomobile.com`, in which GUID and IMEI are persistent. GUID the same as what is transmitted to `irus.coloros.com`.

3. The handset sends GUID to `irus.coloros.com`, sends a subset of installed apps to `icosa.coloros.com` and sends devId to `lang.coloros.com`. Note that devId would change after factory reset while GUID is persistent.

4. GUID and sim card status (inserted or not) is posted to `esa-reg.myoppo.com` periodically.

5. Same as OnePlus, the handset transmits encrypted contents to `log.avlyun.com` with the package name `com.coloros.wifisecuredetect` in the requests. Local Wi-Fi name, mac address, device details and AVLUDID are sent to `log.avlyun.com`.

6. requests to `jits-static-cn.heytapmobi.com` contains an ID. What is it?

7. when receiving/sending a text or receiving a phone call, the device sends a request to `sms.ads.heytapmobi.com` which contains both caller's and callee's phone number, device details, duration of the call, OAID and VAID. The handset posts device details (brand, model, screen size, locale) and phone number to `sms.ads.heytapmobi.com/api/get_next_config`

8. when the clock app is opened, the handset transmits OUID and AUID to `u.bot.heytapmobi.com`.

B.3.2 Preinstalled Non-Realme Packages

1. *China Mobile SDK* The handset sends multiple requests to `a.fxltsbl.com` which belongs to a self-registering platform managed by China Mobile. The requests contain IMEI, device details, and all the installed apps. If a sim card is inserted, CellId and lac are also populated.

2. *Amap* The handset sends encrypted contents to `apsrgeo.amap.com`, `aps.oversea.amap.com`, `offline.aps.amap.com`. The request to `apsrgeo.amap.com` contains the coordinate of the current location, device details and amap identifiers, including adiu and utdid. Data posted to `offline.aps.amap.com` contain similar information except for the coordinates. The request to `aps.oversea.amap.com` incorporate device details, identifiers and also the local Wi-Fi name. adiu and utdid can be reset, while restKey are hardcoded in the package.

3. *sogo* The handset initiate a request to `m.sogo.com` which routinely set cookies but no subsequent requests observed.

4. *QQ* sends an empty request to `tools.3g.qq.com` which returns cookies, but we never observe them in other connections.

Bibliography

[1] EDXPOSED FRAMEWORK. Github - elderdrivers/edxposed: Elder driver xposed framework. https://github.com/ElderDrivers/EdXposed, 2022.

[2] AAFER, Y., ZHANG, X., AND DU, W. Harvesting inconsistent security configurations in custom android {ROMs} via differential analysis. In 25th USENIX Security Symposium (USENIX Security 16) (2016), pp. 1153–1168.

[3] ANDRIUSHCHENKO, M., CROCE, F., FLAMMARION, N., AND HEIN, M. Square attack: A query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision (2020), Springer, pp. 484–501.

[4] ANDROID OPEN SOURCE PROJECT. Android permissions. https://source.android.com/docs/core/permissions, 2022.

[5] ANTONAKAKIS, M., ET AL. Understanding the Mirai botnet. In USENIX Security (2017).

[6] ARZT, S., RASTHOFER, S., FRITZ, C., BODDEN, E., BARTEL, A., KLEIN, J., LE TRAON, Y., OCTEAU, D., AND MCDANIEL, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices 49, 6 (2014), 259–269.

[7] AZIM, T., AND NEAMTIU, I. Targeted and depth-first exploration for systematic testing of android apps. In Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (2013), pp. 641–660.

[8] BARRADAS, D., SANTOS, N., AND RODRIGUES, L. Effective detection of multimedia protocol tunneling using machine learning. In 27th USENIX Security Symposium (USENIX Security 18) (2018), pp. 169–185.

[9] BARRADAS, D., SANTOS, N., RODRIGUES, L., AND NUNES, V. Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on webrtc. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020), pp. 35–48.

[10] BARRADAS, D., SANTOS, N., AND RODRIGUES, L. E. Deltashaper: Enabling unobservable censorship-resistant tcp tunneling over videoconferencing streams. Proc. Priv. Enhancing Technol. 2017, 4 (2017), 5–22.

[11] BEZNAZWY, J., AND HOUMANSADR, A. How china detects and blocks shadowsocks. In Proceedings of the ACM Internet Measurement Conference (2020), pp. 111–124.

[12] BILGE, L., AND DUMITRAŞ, T. Before we knew it: An empirical study of zero-day attacks in the real world. In Proc. ACM CCS (2012).

[13] BOCK, K., HUGHEY, G., QIANG, X., AND LEVIN, D. Geneva: Evolving censorship evasion strategies. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019), pp. 2199–2214.

[14] BRENDEL, W., RAUBER, J., AND BETHGE, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint arXiv:1712.04248 (2017).

[15] BUCZAK, A. L., AND GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials 18, 2 (2016), 1153–1176.

[16] CAI, X., NITHYANAND, R., AND JOHNSON, R. Cs-buflo: A congestion sensitive website fingerprinting defense. In Proceedings of the 13th Workshop on Privacy in the Electronic Society (2014), pp. 121–130.

[17] CAI, X., NITHYANAND, R., WANG, T., JOHNSON, R., AND GOLDBERG, I. A systematic approach to developing and evaluating website fingerprinting defenses. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (2014), pp. 227–238.

[18] CAMINERO, G., LOPEZ-MARTIN, M., AND CARRO, B. Adversarial environment reinforcement learning algorithm for intrusion detection. Computer Networks (2019).

[19] CARLINI, N., AND WAGNER, D. Towards evaluating the robustness of neural networks. In 2017 IEEE Symposium on Security and Privacy (2017), IEEE, pp. 39–57.

[20] CHAWLA, N. V., BOWYER, K. W., HALL, L. O., AND KEGELMEYER, W. P. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research 16 (2002), 321–357.

[21] CHEN, J., JORDAN, M. I., AND WAINWRIGHT, M. J. Hopskipjumpattack: A query-efficient decision-based attack. In 2020 IEEE Symposium on Security and Privacy (2020), IEEE, pp. 1277–1294.

[22] CLOWWINDY. shadowsocks-libev. https://github.com/clowwindy/shadowsocks-libev, 2015.

[23] CO, K. T., ET AL. Procedural noise adversarial examples for black-box attacks on deep convolutional networks. In ACM CCS (2019).

[24] COMINELLI, M., GRINGOLI, F., PATRAS, P., LIND, M., AND NOUBIR, G. Even black cats cannot stay hidden in the dark: Full-band de-anonymization of bluetooth classic devices. In 2020 IEEE Symposium on Security and Privacy (S&P) (2020), IEEE, pp. 534–548.

[25] COMMUNICATIONS SECURITY ESTABLISHMENT AND CANADIAN INSTITUTE FOR CYBERSECURITY. A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). https://registry.opendata.aws/cse-cic-ids2018/, 2018.

[26] CONTINELLA, A., FRATANTONIO, Y., LINDORFER, M., PUCCETTI, A., ZAND, A., KRUEGEL, C., AND VIGNA, G. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In NDSS (2017), vol. 17, pp. 10–14722.

[27] CORTESI, A., HILS, M., KRIECHBAUMER, T., AND CONTRIBUTORS. mitmproxy: A free and open source interactive HTTPS proxy (v5.01), 2020.

[28] CYBERINT RESEARCH. British Airways Flight to DDoS Lands with Cyber Turbulence. https://blog.cyberint.com/british-airways-flight-to-ddos-lands-with-cyber-turbulence, March 2020.

[29] DENG, Z., LIU, Z., CHEN, Z., AND GUO, Y. The random forest based detection of shadowsock's traffic. In 2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC) (2017), vol. 2, IEEE, pp. 75–78.

[30] DI LUZIO, A., MEI, A., AND STEFA, J. Consensus robustness and transaction de-anonymization in the ripple currency exchange system. In IEEE International Conference on Distributed Computing Systems (ICDCS) (2017), pp. 140–150.

[31] DIALLO, A. F., AND PATRAS, P. Adaptive clustering-based malicious traffic classification at the network edge. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications (2021), IEEE, pp. 1–10.

[32] DIRO, A., AND CHILAMKURTI, N. Leveraging lstm networks for attack detection in fog-to-things communications. IEEE Communications Magazine 56, 9 (2018), 124–130.

[33] DOUGLAS LEITH, S. F. Contact tracing app privacy: What data is shared by europe's gaen contact tracing apps. In Proc IEEE INFOCOM 2021 (2021).

[34] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In 2012 IEEE symposium on security and privacy (2012), IEEE, pp. 332–346.

[35] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Protocol misidentification made easy with format-transforming encryption. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013), pp. 61–72.

[36] EGELE, M., KRUEGEL, C., KIRDA, E., AND VIGNA, G. Pios: Detecting privacy leaks in ios applications. In NDSS (2011), pp. 177–183.

[37] ENCK, W., GILBERT, P., HAN, S., TENDULKAR, V., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) 32, 2 (2014), 1–29.

[38] ENSAFI, R., FIFIELD, D., WINTER, P., FEAMSTER, N., WEAVER, N., AND PAXSON, V. Examining how the great firewall discovers hidden circumvention

servers. In Proceedings of the 2015 Internet Measurement Conference (2015), pp. 445–458.

[39] ERICSSON. Ericsson Mobility Report: Mobile data traffic increased almost 300-fold over 10 years, 2021.

[40] ESTABLISHMENT, T. C. S., AND FOR CYBERSECURITY, C. I. A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018). https://registry.opendata.aws/cse-cic-ids2018/, 2018.

[41] EU. What is GDPR, the EU's new data protection law? https://gdpr.eu/what-is-gdpr/, 2016.

[42] FAYED, M., BAUER, L., GIOTSAS, V., KEROLA, S., MAJKOWSKI, M., ODINTSOV, P., SITNICKI, J., CHUNG, T., LEVIN, D., MISLOVE, A., WOOD, C. A., AND SULLIVAN, N. The Ties That Un-Bind: Decoupling IP from Web Services and Sockets for Robust Addressing Agility at CDN-Scale. In Proc. ACM SIGCOMM (2021).

[43] FIFIELD, D., LAN, C., HYNES, R., WEGMANN, P., AND PAXSON, V. Blocking-resistant communication through domain fronting. Proc. Priv. Enhancing Technol. 2015, 2 (2015), 46–64.

[44] FOR CYBERSECURITY, C. I. CICFLOWMETER. https://www.unb.ca/cic/research/applications.html#CICFlowMeter, 2018.

[45] FORBIDDEN STORIES. The pegasus project. https://forbiddenstories.org/case/the-pegasus-project/, July 2021.

[46] FREY, R. M., XU, R., AND ILIC, A. Mobile app adoption in different life stages: An empirical analysis. Pervasive and Mobile computing 40 (2017), 512–527.

[47] FRIDA. Frida: Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/, 2020.

[48] FROLOV, S., AND WUSTROW, E. The use of tls in censorship circumvention. In Network and Distributed System Security Symposium (NDSS) (2019).

[49] GETLANTERN. getlantern/tlsmasq: A Library for Servers which Masquerade as other TLS Servers. https://github.com/getlantern/tlsmasq, 2022.

[50] GHERBI, E., HANCZAR, B., JANODET, J.-C., AND KLAUDEL, W. An encoding adversarial network for anomaly detection. In Asian Conference on Machine Learning (2019), pp. 188–203.

[51] GILL, P., CRETE-NISHIHATA, M., DALEK, J., GOLDBERG, S., SENFT, A., AND WISEMAN, G. Characterizing web censorship worldwide: Another look at the opennet initiative data. ACM Transactions on the Web (TWEB) 9, 1 (2015), 1–29.

[52] GLOROT, X., AND BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In Proc. AISTATS (2010).

[53] GOLLE, P., AND PARTRIDGE, K. On the Anonymity of Home/Work Location Pairs. In Pervasive Computing (2009).

[54] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In Advances in neural information processing systems (2014), pp. 2672–2680.

[55] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014).

[56] GOOGLE. Developer programme policy: 31 march 2021 announcement. `https://support.google.com/googleplay/android-developer/answer/10446026`, March 2021.

[57] GOOGLE. Permissions on Android. `https://developer.android.com/guide/topics/permissions/overview`, 2023.

[58] GOV.UK. What is GDPR, the EU's new data protection law? `https://www.gov.uk/data-protection`, 2018.

[59] GRANADOS, A., MIAH, M. S., ORTIZ, A., AND KIEKINTVELD, C. A realistic approach for network traffic obfuscation using adversarial machine learning. In Decision and Game Theory for Security: 11th International Conference (GameSec 2020) (2020), Springer, pp. 45–57.

[60] GU, G., ET AL. Bothunter: Detecting malware infection through ids-driven dialog correlation. In USENIX Security (2007).

[61] GU, G., ET AL. Botsniffer: Detecting botnet command and control channels in network traffic. In Network and Distributed System Security Symposium (NDSS) (2008).

[62] HAN, C., REYES, I., FEAL, Á., REARDON, J., WIJESEKERA, P., VALLINA-RODRIGUEZ, N., ELAZAR, A., BAMBERGER, K. A., AND EGELMAN, S. The price is (not) right: Comparing privacy in free and paid apps. Proceedings on Privacy Enhancing Technologies 2020, 3 (2020).

[63] HAO, S., LIU, B., NATH, S., HALFOND, W. G., AND GOVINDAN, R. Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In Proceedings of the 12th annual international conference on Mobile systems, applications, and services (2014), pp. 204–217.

[64] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (2016), pp. 770–778.

[65] HEGDE, K., AGRAWAL, R., YAO, Y., AND FLETCHER, C. W. Morph: Flexible acceleration for 3d cnn-based video understanding. In 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (2018), IEEE, pp. 933–946.

[66] HETTICH, S. Kdd cup 1999 data. The UCI KDD Archive (1999).

[67] HOANG, N. P., NIAKI, A. A., DALEK, J., KNOCKEL, J., LIN, P., MARCZAK, B., CRETE-NISHIHATA, M., GILL, P., AND POLYCHRONAKIS, M. How great is the great firewall? measuring china's dns censorship. In 30th USENIX Security Symposium (USENIX Security 21) (2021), pp. 3381–3398.

[68] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.

[69] HOUMANSADR, A., BRUBAKER, C., AND SHMATIKOV, V. The parrot is dead: Observing unobservable network communications. In 2013 IEEE Symposium on Security and Privacy (2013), IEEE, pp. 65–79.

[70] HU, W., HU, W., AND MAYBANK, S. Adaboost-based algorithm for network intrusion detection. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38, 2 (2008), 577–583.

[71] HUAWEI. EMUI 11.0 Security Technical White Paper, 2020. Accessed 31 July 2021.

[72] HUTCHINS, E. M., ET AL. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research (2011).

[73] IDC. Worldwide quarterly mobile phone tracker. https://www.idc.com/getdoc.jsp?containerId=IDC_P8397, July 2021.

[74] IFFLÄNDER, L., ET AL. Hands off my database: Ransomware detection in databases through dynamic analysis of query sequences. arXiv preprint arXiv:1907.06775 (2019).

[75] JIA, Q., ZHOU, L., LI, H., YANG, R., DU, S., AND ZHU, H. Who leaks my privacy: Towards automatic and association detection with gdpr compliance. In International Conference on Wireless Algorithms, Systems, and Applications (2019), Springer, pp. 137–148.

[76] JIANG, K., ET AL. Network intrusion detection combined hybrid sampling with deep hierarchical network. IEEE Access 8 (2020).

[77] JIN, H., LIU, M., DODHIA, K., LI, Y., SRIVASTAVA, G., FREDRIKSON, M., AGARWAL, Y., AND HONG, J. I. Why are they collecting my data? inferring the purposes of network traffic in mobile apps. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2, 4 (Dec. 2018).

[78] JUAREZ, M., AFROZ, S., ACAR, G., DIAZ, C., AND GREENSTADT, R. A critical evaluation of website fingerprinting attacks. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (2014), pp. 263–274.

[79] KAO, D.-Y., AND HSIAO, S.-C. The dynamic analysis of wannacry ransomware. In Proc. ICACT (2018).

[80] KDD. KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[81] KHOSHGOFTAAR, T. M., GOLAWALA, M., AND VAN HULSE, J. An empirical study of learning from imbalanced data using random forest. In 19th IEEE

International Conference on Tools with Artificial Intelligence (ICTAI 2007) (2007), vol. 2, IEEE, pp. 310–317.

[82] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[83] KLZGRAD. klzgrad/naiveproxy: Make a Fortune Quietly. `https://github.com/klzgrad/naiveproxy`, 2022.

[84] KOLBITSCH, C., ET AL. Effective and efficient malware detection at the end host. In USENIX Security Symposium (2009), vol. 4, pp. 351–366.

[85] KRUPP, J., ET AL. Identifying the scan and attack infrastructures behind amplification ddos attacks. In Proc. ACM CCS (2016).

[86] KUMAR, S., KUMAR, D., DONTA, P. K., AND AMGOTH, T. Land subsidence prediction using recurrent neural networks. Stochastic Environmental Research and Risk Assessment 36, 2 (2022), 373–388.

[87] LEE, W., AND STOLFO, S. Data mining approaches for intrusion detection. In USENIX Security Symposium (1998).

[88] LEITH, D. Mobile handset privacy: Measuring the data ios and android send to apple and google. In Proc SECURECOM (2021).

[89] LEITH, D. J. Mobile Handset Privacy: Measuring The Data iOS and Android Send to Apple And Google. In Proc Securecomm (2021).

[90] LEITH, D. J. Mobile handset privacy: Measuring the data ios and android send to apple and google. In Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part II 17 (2021), Springer, pp. 231–251.

[91] LEITH, D. J. Web Browser Privacy: What Do Browsers Say When They Phone Home? IEEE Access (2021).

[92] LEITH, D. J., AND FARRELL, S. Contact Tracing App Privacy: What Data Is Shared By Europe's GAEN Contact Tracing Apps. In Proc IEEE INFOCOM (2021).

[93] LI, J., ZHOU, L., LI, H., YAN, L., AND ZHU, H. Dynamic traffic feature camouflaging via generative adversarial networks. In 2019 IEEE Conference on Communications and Network Security (CNS) (2019), IEEE, pp. 268–276.

[94] LI, Z., AND QIN, Z. A semantic parsing based lstm model for intrusion detection. In ICONIP (2018).

[95] LI, Z., QIN, Z., HUANG, K., YANG, X., AND YE, S. Intrusion detection using convolutional neural networks for representation learning. In International Conference on Neural Information Processing (2017), Springer, pp. 858–866.

[96] LIN, W.-C., KE, S.-W., AND TSAI, C.-F. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-based systems 78 (2015), 13–21.

[97] LIN, Z., ET AL. IDSGAN: Generative adversarial networks for attack generation against intrusion detection. arXiv:1809.02077 (2019).

[98] LIU, H., DIALLO, A., AND PATRAS, P. Amoeba: Circumventing ml-supported network censorship via adversarial reinforcement learning. Tech. rep., 2023.

[99] LIU, H., LEITH, D. J., AND PATRAS, P. Android os privacy under the loupe – a tale from the east. In Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (2023), Association for Computing Machinery, p. 31–42.

[100] LIU, H., AND PATRAS, P. Netsentry: A deep learning approach to detecting incipient large-scale network attacks. Computer Communications 191 (2022), 119–132.

[101] LIU, H., PATRAS, P., AND LEITH, D. J. On the data privacy practices of android oems. PloS one 18, 1 (2023), e0279942.

[102] LOPEZ-MARTIN, M., ET AL. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT. Sensors 17, 9 (2017), 1967.

[103] MAATEN, L. V. D., AND HINTON, G. Visualizing data using t-SNE. Journal of machine learning research 9, Nov (2008), 2579–2605.

[104] MACHANAVAJJHALA, A., KIFER, D., GEHRKE, J., AND VENKITASUBRAMANIAM, M. l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) 1, 1 (2007), 3–es.

[105] MACHIRY, A., TAHILIANI, R., AND NAIK, M. Dynodroid: An input generation system for android apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (2013), pp. 224–234.

[106] MEIDAN, Y., ET AL. N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. IEEE Pervasive Computing 17, 3 (2018), 12–22.

[107] MIRSKY, Y., ET AL. Kitsune: an ensemble of autoencoders for online network intrusion detection. In NDSS (2018).

[108] MIRZA, A. H., AND COSAN, S. Computer network intrusion detection using sequential lstm neural networks autoencoders. In 2018 26th Signal Processing and Communications Applications Conference (SIU) (2018), IEEE, pp. 1–4.

[109] MOHAJERI MOGHADDAM, H., LI, B., DERAKHSHANI, M., AND GOLDBERG, I. Skypemorph: Protocol obfuscation for tor bridges. In Proceedings of the 2012 ACM conference on Computer and communications security (2012), pp. 97–108.

[110] MOORE, A. W., AND ZUEV, D. Internet traffic classification using bayesian analysis techniques. In ACM SIGMETRICS Performance Evaluation Review (2005), vol. 33, ACM, pp. 50–60.

[111] MOUSTAFA, N., ET AL. Outlier dirichlet mixture mechanism: Adversarial statistical learning for anomaly detection in the fog. IEEE Transactions on Information Forensics and Security 14, 8 (2019), 1975–1987.

[112] NADERI-AFOOSHTEH, A., ET AL. Malmax: Multi-aspect execution for automated dynamic web server malware analysis. In ACM CCS (2019).

[113] NASR, M., BAHRAMALI, A., AND HOUMANSADR, A. Deepcorr: Strong flow correlation attacks on tor using deep learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (2018), pp. 1962–1976.

[114] NASR, M., BAHRAMALI, A., AND HOUMANSADR, A. Defeating dnn-based traffic analysis systems in real-time with blind adversarial perturbations. In USENIX Security Symposium (2021), pp. 2705–2722.

[115] NGUYEN, T. T., BACKES, M., MARNAU, N., AND STOCK, B. Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in android apps. In 30th USENIX Security Symposium (USENIX Security 21) (2021), pp. 3667–3684.

[116] NIAKI, A. A., CHO, S., WEINBERG, Z., HOANG, N. P., RAZAGHPANAH, A., CHRISTIN, N., AND GILL, P. Iclab: A global, longitudinal internet censorship measurement platform. In 2020 IEEE Symposium on Security and Privacy (SP) (2020), IEEE, pp. 135–151.

[117] NVIDIA. Jetson Nano — NVIDIA Developer. https://developer.nvidia.com/embedded/jetson-nano, 2019.

[118] OF CHINA, N. P. C. Personal Information Protection Law of the People's Republic of China. http://www.npc.gov.cn/npc/c30834/202108/a8c4e3672c74491a80b53a172bb753fe.shtml, 2021.

[119] PANCHENKO, A., LANZE, F., PENNEKAMP, J., ENGEL, T., ZINNEN, A., HENZE, M., AND WEHRLE, K. Website fingerprinting at internet scale. In NDSS (2016).

[120] PANCHENKO, A., NIESSEN, L., ZINNEN, A., AND ENGEL, T. Website fingerprinting in onion routing based anonymization networks. In Proceedings of the 10th annual ACM workshop on Privacy in the electronic society (2011), pp. 103–114.

[121] PASZKE, A., ET AL. Pytorch: An imperative style, high-performance deep learning library. In NeurIPS (2019).

[122] PHAM, A., DACOSTA, I., LOSIOUK, E., STEPHAN, J., HUGUENIN, K., AND HUBAUX, J.-P. Hidemyapp: Hiding the presence of sensitive apps on android. In 28th USENIX Security Symposium (USENIX Security 19) (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 711–728.

[123] QUALYS. Qualys ssl labs - ssl pulse. https://www.ssllabs.com/ssl-pulse/, 2023.

[124] RAGHU, M., UNTERTHINER, T., KORNBLITH, S., ZHANG, C., AND DOSOVITSKIY, A. Do vision transformers see like convolutional neural networks? Advances in Neural Information Processing Systems 34 (2021).

[125] RAMAN, R. S., STOLL, A., DALEK, J., RAMESH, R., SCOTT, W., AND ENSAFI, R. Measuring the deployment of network censorship filters at global scale. In NDSS (2020).

[126] RAZAGHPANAH, A., NITHYANAND, R., VALLINA-RODRIGUEZ, N., AND SUNDARESAN, S. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In Network and Distributed System Security Symposium (NDSS) (2018).

[127] REARDON, J., FEAL, Á., WIJESEKERA, P., ELAZARI BAR ON, A., VALLINA-RODRIGUEZ, N., AND EGELMAN, S. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In 28th USENIX Security Symposium (2019).

[128] REARDON, J., FEAL, Á., WIJESEKERA, P., ON, A. E. B., VALLINA-RODRIGUEZ, N., AND EGELMAN, S. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In 28th USENIX Security Symposium (USENIX Security 19) (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 603–620.

[129] REN, J., LINDORFER, M., DUBOIS, D. J., RAO, A., CHOFFNES, D., AND VALLINA-RODRIGUEZ, N. Bug fixes, improvements,... and privacy leaks. In Network and Distributed System Security Symposium (NDSS) (2018).

[130] REN, J., LINDORFER, M., DUBOIS, D. J., RAO, A., CHOFFNES, D., VALLINA-RODRIGUEZ, N., ET AL. Bug fixes, improvements,... and privacy leaks. In The 25th Annual Network and Distributed System Security Symposium (NDSS 2018) (2018).

[131] REN, J., RAO, A., LINDORFER, M., LEGOUT, A., AND CHONES, D. Recon: Revealing and controlling privacy leaks in mobile network traÿc. In Proc. of the International Conference on Mobile Systems, Applications and Services (MobiSys) (2016).

[132] RIKKAAPPS. Github - rikkaapps/riru: Inject into zygote process. https://github.com/RikkaApps/Riru, 2022.

[133] RIMMER, V., PREUVENEERS, D., JUAREZ, M., VAN GOETHEM, T., AND JOOSEN, W. Automated website fingerprinting through deep learning. In Proceedings of the 25nd Network and Distributed System Security Symposium (NDSS 2018) (2018), Internet Society.

[134] RUFF, L., ET AL. Deep one-class classification. In International conference on machine learning (2018), pp. 4393–4402.

[135] SAAD, E., ET AL. OWASP Web Security Testing Guide Version 4.1. https://github.com/OWASP/wstg/tree/master/document, 2019.

[136] SAIED, A., ET AL. Detection of known and unknown DDoS attacks using Artificial Neural Networks. Neurocomputing 172 (2016), 385–393.

[137] SANGKATSANEE, P., ET AL. Practical real-time intrusion detection using machine learning approaches. Computer Communications 34, 18 (2011), 2227–2235.

[138] SANZ MAROTO, J., LIU, H., AND PATRAS, P. On the struggle bus: A detailed security analysis of the m-tickets app. In Information Security: 23rd International Conference, ISC 2020, Bali, Indonesia, December 16–18, 2020, Proceedings 23 (2020), Springer, pp. 234–252.

[139] SCHNEIDER, J. What is section 702 of fisa, anyway? https://edition.cnn.com/2018/01/11/politics/trump-fisa-section-702-surveillance-data/index.html, 2018.

[140] SCHULMAN, J., LEVINE, S., ABBEEL, P., JORDAN, M., AND MORITZ, P. Trust region policy optimization. In International Conference on Machine Learning (2015), PMLR, pp. 1889–1897.

[141] SCHULMAN, J., MORITZ, P., LEVINE, S., JORDAN, M., AND ABBEEL, P. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015).

[142] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017).

[143] SCOCCIA, G. L., KANJ, I., MALAVOLTA, I., AND RAZAVI, K. Leave my apps alone! a study on how android developers access installed apps on user's device. In Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems (2020).

[144] SHARAFALDIN, I., ET AL. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In ICISSP (2018).

[145] SHEFFEY, S., AND ADERHOLDT, F. Improving meek with adversarial techniques. In 9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19) (2019).

[146] SHENOI, A., KARTHIK, P., SABHARWAL, K., JIALIN, L., AND DIVAKARAN, D. M. ipet: Privacy enhancing traffic perturbations for iot communications. In Proceedings on Privacy Enhancing Technologies (PoPETs 2023) (2023).

[147] SILVER, D., SCHRITTWIESER, J., SIMONYAN, K., ANTONOGLOU, I., HUANG, A., GUEZ, A., HUBERT, T., BAKER, L., LAI, M., BOLTON, A., ET AL. Mastering the game of go without human knowledge. Nature 550, 7676 (2017), 354–359.

[148] SIRINAM, P., ET AL. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In Proc. ACM CCS (2018).

[149] SIRINAM, P., IMANI, M., JUAREZ, M., AND WRIGHT, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (2018), pp. 1928–1943.

[150] SIRINAM, P., MATHEWS, N., RAHMAN, M. S., AND WRIGHT, M. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (2019), pp. 1131–1148.

[151] SOMMER, R., AND PAXSON, V. Outside the closed world: On using machine learning for network intrusion detection. In IEEE S&P (2010).

[152] SPAMBAUS. Spamhaus Botnet Threat Update: Q1-2021. `https://www.spamhaus.org/news/article/809/spamhaus-botnet-threat-update-q1-2021`, 2021.

[153] SRIVATSA, M., AND HICKS, M. Deanonymizing mobility traces: Using social network as a side-channel. In ACM Conference on Computer and Communications Security (CCS) (2012), pp. 628–637.

[154] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. Advances in neural information processing systems 27 (2014).

[155] SWEENEY, L. k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10, 05 (2002), 557–570.

[156] TAVALLAEE, M., BAGHERI, E., LU, W., AND GHORBANI, A. A. A detailed analysis of the kdd cup 99 data set. In 2009 IEEE symposium on computational intelligence for security and defense applications (2009), Ieee, pp. 1–6.

[157] TENCENT. Github - tencent/soter: A secure and quick biometric authentication standard and platform in android held by tencent. https://github.com/Tencent/soter, 2022.

[158] TOR. Tor Project — Anonymity Online . https://www.torproject.org/, 2022.

[159] TOR PROJECT. Obfsproxy 3. https://support.torproject.org/glossary/obfs3/, 2022.

[160] TROJAN-GFW. An Unidentifiable Mechanism that Helps You Bypass GFW. https://github.com/trojan-gfw/trojan, 2022.

[161] US DEPARTMENT OF LABOR. Guidance on the protection of personal identifiable information, 2015.

[162] V2FLY. Awesome Tools — V2Fly.org. https://www.v2fly.org/en_US/awesome/tools.html#third-party-gui-clients, 2022.

[163] V2FLY. v2fly/v2ray-core: A Platform for Building Proxies to Bypass Network Restrictions. https://github.com/v2fly/v2ray-core, 2022.

[164] VALLINA-RODRIGUEZ, N., SHAH, J., FINAMORE, A., GRUNENBERGER, Y., PAPAGIANNAKI, K., HADDADI, H., AND CROWCROFT, J. Breaking for com-

mercials: characterizing mobile advertising. In Proceedings of the 2012 Internet measurement conference (2012), pp. 343–356.

[165] VAN KLEEK, M., LICCARDI, I., BINNS, R., ZHAO, J., WEITZNER, D. J., AND SHADBOLT, N. Better the devil you know: Exposing the data sharing practices of smartphone apps. In CHI Conference on Human Factors in Computing Systems (2017), pp. 5208—-5220.

[166] VENTURES, C. Global Cybercrime Damages Predicted To Reach $6 Trillion Annually By 2021. https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/, 2020.

[167] VERMA, G., CIFTCIOGLU, E., SHEATSLEY, R., CHAN, K., AND SCOTT, L. Network traffic obfuscation: An adversarial machine learning approach. In MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM) (2018), IEEE, pp. 1–6.

[168] VINAYAKUMAR, R., ET AL. Applying convolutional neural network for network intrusion detection. In Proc. International Conference on Advances in Computing, Communications and Informatics (ICACCI) (2017).

[169] VULNERABILITIES, C. C., AND EXPOSURES. CVE-2017-0144. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144, Sept. 2016.

[170] WANG, L., DYER, K. P., AKELLA, A., RISTENPART, T., AND SHRIMPTON, T. Seeing through network-protocol obfuscation. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015), pp. 57–69.

[171] WANG, T., AND GOLDBERG, I. {Walkie-Talkie}: An efficient defense against passive website fingerprinting attacks. In 26th USENIX Security Symposium (USENIX Security 17) (2017), pp. 1375–1390.

[172] WANG, Z., CAO, Y., QIAN, Z., SONG, C., AND KRISHNAMURTHY, S. V. Your state is not mine: A closer look at evading stateful internet censorship. In Proceedings of the 2017 Internet Measurement Conference (2017), pp. 114–127.

[173] WANG, Z., LI, Z., XUE, M., AND TYSON, G. Exploring the eastern frontier: A first look at mobile app tracking in china. In Passive and Active Measurement (2020), A. Sperotto, A. Dainotti, and B. Stiller, Eds.

[174] WANG, Z., LI, Z., XUE, M., AND TYSON, G. Exploring the eastern frontier: A first look at mobile app tracking in china. In International Conference on Passive and Active Network Measurement (2020), Springer, pp. 314–328.

[175] WANG, Z., AND ZHU, S. Symtcp: Eluding stateful deep packet inspection with automated discrepancy discovery. In Network and Distributed System Security Symposium (NDSS) (2020).

[176] WINTER, P., PULLS, T., AND FUSS, J. Scramblesuit: A polymorphic network protocol to circumvent censorship. In Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society (2013), pp. 213–224.

[177] WIRED. A simple way to make it harder for mobile ads to track you, Aug 2019.

[178] WRIGHT, C. V., COULL, S. E., AND MONROSE, F. Traffic morphing: An efficient defense against statistical traffic analysis. In NDSS (2009), vol. 9.

[179] WU, L., GRACE, M., ZHOU, Y., WU, C., AND JIANG, X. The impact of vendor customizations on android security. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013), pp. 623–634.

[180] XIA, Y., ET AL. Learning discriminative reconstructions for unsupervised outlier removal. In Proc. IEEE ICCV (2015).

[181] XINGJIAN, S., ET AL. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In NeurIPS (2015).

[182] XUE, D., MIXON-BACA, B., VALDIKSS, ABLOVE, A., KUJATH, B., CRANDALL, J. R., AND ENSAFI, R. Tspu: Russia's decentralized censorship system. In Proceedings of the 22nd ACM Internet Measurement Conference (2022), pp. 179–194.

[183] YADAV, T. K., SINHA, A., GOSAIN, D., SHARMA, P. K., AND CHAKRAVARTY, S. Where the light gets in: Analyzing web censorship mechanisms in india. In Proceedings of the Internet Measurement Conference 2018 (2018), pp. 252–264.

[184] YANG, J., XIAO, S., LI, A., LU, W., GAO, X., AND LI, Y. Msta-net: Forgery detection by generating manipulation trace based on multi-scale self-texture attention. IEEE Transactions on Circuits and Systems for Video Technology (2021).

[185] YANG, Z., YANG, M., ZHANG, Y., GU, G., NING, P., AND WANG, X. S. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013), pp. 1043–1054.

[186] YAWNING. Yawning/obfs4: The Obfourscator. `https://github.com/Yawning/obfs4`, 2022.

[187] YEN, T.-F., XIE, Y., YU, F., YU, R. P., AND ABADI, M. Host fingerprinting and tracking on the web:privacy and security implications. In Network and Distributed System Security Symposium (NDSS) (February 2012).

[188] YI, Y., ET AL. Incremental SVM Based on Reserved Set for Network Intrusion Detection. Expert Systems with Applications 38, 6 (2011), 7698–7707.

[189] YIN, C., ET AL. A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access 5 (2017), 21954–21961.

[190] ZENATI, H., FOO, C. S., LECOUAT, B., MANEK, G., AND CHANDRASEKHAR, V. R. Efficient GAN-Based Anomaly Detection. arXiv preprint arXiv:1802.06222 (2018).

[191] ZHANG, C., ET AL. Multi-service mobile traffic forecasting via convolutional long short-term memories. In Proc. IEEE International Symposium on Measurements & Networking (July 2019).

[192] ZHANG, C., ET AL. Tiki-taka: Attacking and defending deep learning-based intrusion detection systems. In ACM CCSW (Nov 2020).

[193] ZHOU, X., LEE, Y., ZHANG, N., NAVEED, M., AND WANG, X. The peril of fragmentation: Security hazards in android device driver customizations. In 2014 IEEE Symposium on Security and Privacy (2014), IEEE, pp. 409–423.

[194] ZOLBAYAR, B.-E., SHEATSLEY, R., MCDANIEL, P., WEISMAN, M. J., ZHU, S., ZHU, S., AND KRISHNAMURTHY, S. Generating practical adversarial network traffic flows using nidsgan. arXiv preprint arXiv:2203.06694 (2022).

[195] ZONG, B., SONG, Q., MIN, M. R., CHENG, W., LUMEZANU, C., CHO, D., AND CHEN, H. Deep autoencoding Gaussian mixture model for unsupervised anomaly detection. In ICLR (2018).