universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Shanon: A Wireshark-based Network Packet Capture Anonymization Tool"

verfasst von / submitted by

Mislav Culig

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. Dr.Peter Reichl, Privatdoz.

Zusammenfassung

Der Zugriff auf reale Daten ist wichtig für die Netzwerkforschung und -sicherheit. Diese Daten können in verschiedenen Logs enthalten sein, oder in Netzwerkpakettraces, die von einer Netzwerkschnittstelle aufgezeichnet wurden. Durch das Teilen von Netzwerkpakettraces werden jedoch möglicherweise Schwachstellen in den Netzwerken derjenigen, die sie teilen, aufgedeckt, und es können vertrauliche und private Informationen offengelegt werden. Um die Wahrscheinlichkeit zu minimieren, dass Schwachstellen und vertrauliche oder private Informationen aufgedeckt werden, müssen Netzwerkpakettraces vor der Freigabe anonymisiert werden. Dies bedeutet, dass so viele vertrauliche Daten wie möglich entfernt werden müssen.

Es gibt keine universelle Lösung, da möglicherweise unterschiedliche Teile der Daten für Forschung oder Analysen erforderlich sind und unterschiedliche Organisationen, die Netzwerkpakettraces bereitstellen, unterschiedliche Anforderungen an die Daten stellen, die anonymisiert werden müssen. Zu diesem Zweck gibt es derzeit viele Tools und Frameworks. Trotz der Existenz dieser Tools und der potenziellen Vorteile einer offeneren gemeinsamen Nutzung von Netzwerkpakettraces teilen wenige Autoren ihre Netzwerkpakettraces, und viele der entwickelten Tools und Frameworks scheinen nicht verwendet zu werden.

Diese Arbeit liefert einen Überblick über die vorhandene Literatur auf dem Gebiet der Anonymisierung von Netzwerkpakettraces, einschließlich Standards und Gesetzen im US-amerikanischen und EU-Raum, eine Taxonomie der Funktionen von Anonymisierungstools für Netzwerkpakettraces, sowie `libAnonLua`, eine Lua-Bibliothek zur Anonymisierung von Netzwerkpakettraces, die aus wiederverwendbaren Anonymisierungsmethoden besteht, und *Shanon*, ein flexibles, richtlinienbasiertes Anonymisierungstool, implementiert als Plugin für den Wireshark-Netzwerkprotokollanalysator. Das entwickelte Tool wird anhand eines echten Netzwerkpakettraces evaluiert.

Abstract

Access to real-world data is important for network research and security. This data can be contained in various logs, including packet traces which contain packets captured from an interface within the network. However, the sharing of packet traces potentially exposes vulnerabilities in the networks of those sharing them and may expose sensitive and private information. To minimize the chance of exposing vulnerabilities and sensitive or private information, packet traces need to be anonymized before sharing, meaning as much sensitive data as possible needs to be removed.

There is no universal solution, as different parts of the data may be required for research or analysis, and different organizations providing the traces may have different requirements when it comes to the data that needs to be anonymized. Many tools and frameworks currently exist for this purpose. However, despite the existence of these tools and the potential benefits of more open sharing of network packet captures, few authors publicly share their traces, and many of the tools and frameworks developed appear not to be in use.

This thesis contributes a review of existing literature in the field of network packet trace anonymization, including standards and United States (US) and European Union (EU) law, a taxonomy of features of network packet anonymization tools, a Lua library of reusable network anonymization primitives, `libAnonLua`, and a flexible, policy-based anonymization tool that plugs into the Wireshark network protocol analyzer, *Shanon*. The developed tool is evaluated using a real world capture of network traffic.

Acknowledgement

I would like to thank my father, who willingly went on tours of duty in the Middle East and Africa and set aside significant portions of his earnings there to help finance my studies. It was thanks to his hard work and sacrifice that I was able to pay for my bachelor studies and complete them. My mother, who, despite understanding the significant financial cost and risk of it, encouraged me to continue my studies abroad and supported me throughout them: financially, emotionally, as an expert in her field, and as a scientist. My grandmother, who was always there, at home, always around, and in a way like a second mother to me. She always said she would like to live to see me get my diploma. I am glad this wish will come true. I would like to thank two of my best friends, my brother and sister in all except blood, Luka and Marina. It was Luka who first went to study in Vienna, and whose, at first joking, invitation to come study here is the reason I have come this far. Luka, your emotional support during the first few weeks was crucial to me making it. Marina, your hard work and dedication in spite of adversity was and still is an inspiration to me to work harder, and not to give up when things do not go my way. I have found myself thinking about it many times when things were just plain crazy, and it helped me ground myself and endure. Last, but not least, I would like to thank an amazing friend I made along the way, Lily. The fun times I had gaming with you helped me unwind on difficult days and motivated me to work hard so I could justify relaxing and forgetting about all the things I still have to do. Unrelated to this thesis, but still of great importance, you taught me a lot that I did not yet know about myself, and helped make me a better person. Thank you.

It is thanks to all my friends and family that I am who I am and where I am, both those who I have mentioned here, and those who I have not. You all have my thanks for your continued support and the valuable lessons you have taught me in life.

Contents

Acronyms

ACK Acknowledgment field significant. 60, 62, 63, 65, 71

AES Advanced Encryption Standard. 85

API Application Programming Interface. 81, 82, 97, 117

ARP Address Resolution Protocol. 5, 12, 25, 39, 41, 42, 44, 51, 52, 54, 70, 116

CAIDA Center for Applied Internet Data Analysis. 35

CB Custom Block. 75

CGN Carrier-Grade Network Address Translation. 67, 70

CIDR Classless Inter-Domain Routing. 67–69, 87, 88

CPU Central Processing Unit. 101, 104

CRC Cycling Redundancy Check. 86

CSV Comma Separated Values. 80

CWR Congestion Window Reduced. 60, 65

DF Don't Fragment. 47

DHCPv6 Dynamic Host Configuration Protocol for IPv6. 57

DNS Domain Name System. 75

DSCP Differentiated Services Code Point. 45, 47, 48

ECE ECN-Echo. 60, 65

ECN Explicit Congestion Notification. 45, 47, 48, 60

ECPA Electronic Communications Privacy Act. 15, 24, 116

EPB Enhanced Packet Block. 12, 75, 76, 80, 83

EU European Union. 2, 3, 15, 16, 23, 39, 116

FCS Frame Check Sequence. 44, 45, 86

FIN No more data from sender. 60, 65, 71

GB Gigabyte. 34, 36

GDPR General Data Protection Regulation. 15, 23, 39, 116

GHz Gigahertz. 101

HMAC Hash-based Message Authentication Code. 22, 84, 85

IANA The Internet Assigned Numbers Authority. 46, 51, 58, 59, 67, 68, 76, 119

ICMP Internet Control Message Protocol. 5, 12, 39, 41, 42, 44, 52–55, 57, 72, 78, 86, 87, 94–96, 98, 116

ICMPv6 Internet Control Message Protocol for the Internet Protocol Version 6. 5, 12, 42–44, 52–55, 78, 86, 87, 93, 95, 97, 98, 106, 116

ID Identifier. 75, 81–83

IDB Interface Description Block. 12, 75, 76, 80–82, 104

IDS Intrusion Detection System. 14, 37

IEEE Institute of Electrical and Electronics Engineers. 67

IHL Internet Header Length. 43, 45, 47, 60, 78

IP Internet Protocol. 12, 14, 18, 20–25, 28, 35, 44, 47, 56, 66, 69, 78, 88, 94, 96, 112

IPv4 Internet Protocol version 4. 5, 12, 13, 22, 39, 41–48, 51–54, 57–60, 66–70, 72, 75, 78, 85–88, 92–94, 98, 100, 103, 106–109

IPv6 Internet Protocol version 6. 5, 12, 14, 21, 22, 39, 41–44, 48–54, 57, 58, 66–70, 72, 73, 75, 78, 85, 87, 88, 92–94, 96–98, 100, 102, 103, 106, 107, 109, 116

ISB Interface Statistics Block. 75

ISN Initial Sequence Number. 59

LSB Least Significant Bit. 80, 84

LTS Long-term Support. 101

MA-L MAC Addresses - Large. 67

MA-M MAC Addresses - Medium. 67

MA-S MAC Addresses - Small. 67

MAC Media Access Control. 39, 44, 66, 67, 69, 70, 107

MB Megabyte. 36, 101, 102, 106

Mbps Megabits per second. 34

MD5 Message Digest 5. 22

MF More Fragments. 47

MSB Most Significant Bit. 80, 84

MTU Maximum Transmission Unit. 54, 56

NAT Network Address Translation. 69

NDP Neighbor Discovery Protocol. 5, 12, 42, 44, 51, 52, 54–57, 70, 96, 98, 116

NRB Name Resolution Block. 75

NS Nonce Sum. 60, 65

NVMe Non-volatile Memory Express. 101

OS Operating System. 29, 72, 100, 104

OUI Organizationally Unique Identifier. 67

PBKDF2 Password Based Key Derivation Function 2. 84

PC Personal Computer. 22

pcapng PCAP Next Generation Capture File Format. 6, 15, 72, 74–76, 80, 81, 94, 95, 104, 119

PEN Private Enterprise Number. 76, 119

pinfo Packet Information. 77

PSH Push Function. 60, 65

RFC Request for Comments. 45, 46, 48, 49, 52–54, 58, 62, 63, 97

RIPE Reseaux IP Europeens. 68

RST Reset the connection. 60, 63, 65, 71

SC2D Shipping Flexible Analysis Code to the Data. 30, 31

SCA Stored Communications Act. 15, 24

SHA-2 Secure Hash Algorithm 2. 85

SHB Section Header Block. 12, 75, 76, 81

SIGCOMM Special Interest Group on Data Communication. 29

SPB Simple Packet Block. 75, 80

SSD Solid-state Drive. 101

SSL Secure Sockets Layer. 103, 112

SYN Synchronization. 59, 60, 62–65, 71

tapinfo Tap Information. 77

TB Terabyte. 101

TCP Transmission Control Protocol. 5, 12–14, 35, 39, 41–44, 53, 57–65, 71, 72, 77, 84, 86, 87, 93, 97, 98, 100, 102–104, 106, 107, 109–113, 117, 119

TLS Transport Layer Security. 103, 112

TLV Type-Length-Value. 49, 54, 72

TS Timestamp. 63

TTL Time To Live. 46–48, 72

TVB Testy Virtual Buffer. 42, 77–79

U.S.C. United States Code. 24

UDP User Datagram Protocol. 5, 12, 39, 41–44, 53, 57–60, 71, 72, 84, 86, 87, 93, 97, 100, 103, 104, 106, 107, 109, 112, 119

URG Urgent Pointer field significant. 60, 65

US United States. 2, 3, 15, 16, 23, 24, 39, 116

VLAN Virtual Local Area Network. 44

VM Virtual Machine. 101

List of Tables

List of Figures

1 Introduction

1.1 The problem

Real network data is vital for network research, education, advancing network design, maintaining secure and reliable networks, and the evaluation and development of security mechanisms [71, 56, 82, 37]. This data can be contained in various logs, such as Intrusion Detection System (IDS) logs, or packet captures or traces which contain packets captured from an interface within the network. Since packet captures or traces are also a form of logs, these terms are often used interchangeably.

The focus of this thesis are packet traces. Packet traces often contain information about individuals, enterprises and the networks themselves that should not be disclosed. In order to properly protect sensitive information in these traces while also maintaining a reasonable amount of utility it is important to be able to anonymize various aspects of the traces. Various solutions have been proposed in papers, and tools and frameworks have been developed based on these solutions. However, despite the potential benefits of more open sharing of packet captures, there do not appear to be many resources available, and many of the tools and frameworks developed appear not to be in use. This is likely due to the tools lacking features that would make them appealing to users.

The anonymization of packet traces is a difficult problem. While some fields, like Internet Protocol (IP) addresses, present a clear and obvious privacy concern, other fields like flags, or length fields can be more or less of a threat depending on the environment in which the packets were captured. Ideally anonymizing a protocol would result in a packet trace that can safely be shared without the loss of any information. In practice, this is not possible, as many methods of anonymization rely on loss of information. Maintaining the utility of a capture for research purposes while ensuring the capture does not reveal information the organization providing the capture does not wish revealed is not always possible. In addition to the complexity of anonymization itself, the complexity of network protocols is a challenge as well. In order to anonymize a protocol while preserving as much utility as possible it is necessary to understand the design of the protocol well. Protocols like TCP and Internet Protocol version 6 (IPv6) present additional challenges, as they may have additional, optional fields attached in the form of extension headers or protocol options which can modify the behaviour of the protocol. These fields need to be processed in order to maintain as much of the protocol's semantics as possible. The combined difficulty of anonymization and complexity of network protocols results in a large prob-

lem surface the entirety of which no single individual can reasonably tackle.

1.2 Aim of this thesis

The aim of this thesis is to develop a network packet capture anonymization tool. This aim will be achieved through the following tasks:

1. Identify features the tool needs to support
2. Identify protocols the tool needs to support
3. Develop the anonymization tool
4. Evaluate the anonymization tool

1.3 Contributions

This thesis makes the following contributions to the field of network packet capture anonymization:

A review of literature in the field of network packet trace anonymization. This includes a review of existing tools and frameworks, which differ from tools in their ability to adapt to the needs of a wider range of users, in Section 2.1. Existing methods of anonymization are reviewed in Section 2.2. The relevant US law, the Electronic Communications Privacy Act (ECPA) and its components: The Wiretap Act, Pen Register Statute, and Stored Communications Act (SCA), as well as EU law, the General Data Protection Regulation (GDPR), are reviewed in Section 2.3.

A taxonomy of features of network packet anonymization tools. This taxonomy identifies features that can make an anonymization tool useful to a wider audience. It is based on the features identified in existing frameworks: FLAIM [86], developed by Slagell et al., tcpmkpub [72], developed by Pang et al. and PktAnon [41] developed by Gamer et al. The taxonomy is described in Chapter 3.

A Lua library containing reusable network protocol anonymization primitives. This library, named `libAnonLua` consists of a set of constants, functions for writing pcapng files, anonymization functions, functions for calculating checksums, and helper functions written in C which can be imported

into and used in a script written in the Lua scripting language. A detailed description of the library can be found in Chapter 7.

Shanon - a flexible, policy-based anonymization tool that plugs into the Wireshark network protocol analyzer. Shanon was written in Lua as part of this thesis with the aim of supporting many of the features identified in the taxonomy of features mentioned above. It is designed to work as a plugin for the Wireshark network protocol analyzer. More detailed description of Shanon can be found in Chapter 8. An evaluation of Shanon on a real world example can be found in Chapter 9.

1.4 Structure of this thesis

The structure of the thesis follows the tasks described in Section 1.2. Chapter 2 studies the state of the art, including existing anonymization tools which leads to Chapters 3 and 4 which identify features and protocols our anonymization tool, Shanon, needs to support (Tasks 1 and 2). Chapters 5, 6, 7, and 8 describe the development of Shanon (Task 3). Shanon is evaluated in Chapter 9 (Task 4). The contents of the individual chapters are as follows:

Chapter 2 will describe the current state of the art in anonymization. The currently existing anonymization tools, methods of anonymization, the legal environment in the US and EU, attacks on anonymization, and proposed alternatives that fall outside of the scope of this work.

In Chapter 3 a taxonomy of features of anonymization frameworks is proposed based on three existing frameworks.

Chapter 4 explains the choice of protocols to initially support in the anonymization tool and provides an overview of their fields and features.

In Chapter 5 the anonymization of the various fields of the supported protocols is discussed.

Chapter 6 explains the choice of Wireshark as the basis for a new anonymization tool, the file format used by Wireshark and the usage of the Lua scripting language in Wireshark.

Chapter 7 provides an overview of the `libAnonLua` library that was developed as part of this work.

Chapter 8 describes Shanon, the anonymization tool developed as part of this work.

Chapter 9 evaluates the anonymization methods and the results of anonymization using the developed anonymization tool on a real-world capture.

The thesis is concluded in Chapter 10.

Possible future work is examined in Chapter 11.

2 State of the art

Computers, as well as various other devices such as cellphones and “smart” devices (devices which offer additional functionality through services available over the internet) communicate with each other by exchanging packets of data across wired and wireless links. These packets consist of addressing information necessary to deliver the packets to their destination, various fields used for the operation of the protocols the packets consist of, and data intended for the recipient. Network packet captures are recordings of these packets from the perspective of a device located somewhere along the path between their source and destination. These captures can contain not only a history of exchanged packets during a time period when they were captured, but also additional meta-information about the environment at the time of the capture, the device making the capture, and contextual information which can be interpreted from the captured packets themselves. Typically, a network packet trace is first recorded, then later examined using special software, such as Wireshark [40]. Due to the sensitivity of information contained in network captures there is a need to anonymize them, i.e. remove sensitive information such as public IP addresses from the fields of protocols contained in the capture, before releasing them to the public or sharing them with researchers. Many tools have been developed for that purpose [41, 4, 70, 69, 38, 62, 100]. With the exception of two commercial tools, anonymization tools were developed by organizations and individuals with their own needs in mind and lack the ability to adapt to the various needs of different users.

2.1 Overview of Tools

Many tools have been developed for the purpose of packet trace anonymization:

- TraceWrangler [4]
- WireEdit [70]
- SafePcap [69]
- CryptoPAn [38]
- Tcpdpriv [62]
- SCRUB-tcpdump [100]

In addition three frameworks have been developed. These frameworks differ from the tools mentioned above due to being applicable to a wider range of user needs thanks to including many of the features that will be mentioned in Chapter 3. These frameworks are:

- tcpmkpub [72]
- FLAIM [86]
- PktAnon [41]

In terms of the features and anonymization methods the tools mentioned here are limited. This is due to the tools having been developed to fit personal requirements of particular researchers and organizations. As a result they lack the ability to adapt to the various needs different users may require [86]. An exception to this is SafePcap which is a commercial tool. However, there would appear to be no research or publicly available documentation outlining this tool in more detail than the developer website. It is included here, but it seems prudent to approach their claims with a dose of scepticism. The same developer also developed WireEdit, which is not strictly an anonymization tool as its primary purpose is manual, visual editing of packets, however since the developer states removing sensitive fields as an example use case it is included as an anonymization tool.

Given the limitations of the tools mentioned above it seemed more sensible to outline their features in a table rather than discuss each individual tool in greater detail. Table 1 in Chapter 3 outlines the features supported by the anonymization tools found while researching for this thesis. Tools designed for the anonymization of other types of network logs, such as Cisco NetFlow logs which store aggregated information on traffic flows rather than individual packets, are not considered here since this falls outside of the scope of this thesis, which is packet trace anonymization.

In addition to these tools three frameworks were studied and their features used as a basis for the features of anonymization tools. These three frameworks: tcpmkpub, FLAIM and PktAnon are not present in Table 1 or discussed here since their features will be discussed in more details when defining the set of features in Chapter 3. A table of protocols supported by these tools and frameworks can be found in Chapter 4.

2.2 Methods of anonymization

Slagell et al.[86] support a large list of anonymization methods in their framework FLAIM and include a comprehensive list of anonymization methods in

their paper. Some of the methods mentioned in their list were variants of already mentioned methods and have been merged with them. The resulting list is as follows:

- **Black Marker:** The black marker[1] method replaces part of or an entire protocol field with either zeroes or a constant, leading to a loss of information that was contained in the replaced bits.
- **Truncation:** Removal of bits past a certain boundary (i.e. cutting out "user" from "user@example.com")
- **Partitioning:** Partitioning of possible values into subsets
- **Permutation:** A one-to-one mapping on a set.
- **Hash:** Using a hash function to anonymize fields.
- **Enumeration:** Assigning an initial value to an element and sequentially increasing this value

This list will be used in this thesis as it is the most comprehensive list found in the reviewed literature and provides good generalizations that future methods can be grouped into. More details about the classes are provided below.

Black Marker

The black marker method is the digital equivalent of text being censored with black markers in sensitive government documents. Slagell et al. do not just delete the field in FLAIM, however, as this could cause issues with log analysis tools. Instead values are replaced with a single value that is still valid for that field. Slagell et al. provide an example of 0.0.0.0 for IP addresses. Pang et al. name this constant substitution [71]. In their framework, tcpmkpub [72], the operation `ZERO` replaces a field's value with zeroes, and the functions labeled `const_n8`, `const_n16` and `const_n32` in their provided anonymization policy appear to allow for overwriting a protocol field with a user-provided constant. However, in [71] Pang et al. note that substituting identifiers such as IP addresses with constants is undesirable as it makes it impossible to precisely distinguish objects from one another. Gamer et al.[41] support overwriting every byte with a user-supplied constant. This would be better if the length of a value could be specified as is the case with

[1] Named after the practice of running a black marker over a body of text to conceal it.

tcpmkpub, instead of being limited to overwriting every byte. Some fields may not be valid when replaced by all zeroes or byte-sized patterns, so a black marker method should be implemented with care.

Truncation

Truncation is the removal of data past a certain boundary. Slagell et al. provide the examples of truncating an e-mail address and removing the domain, leaving only the user name and truncating 24 bits of an IP address to leave a class C subnet. They also note that truncating all bits is a special example of a black marker. This is consistent with the `ZERO` transformation used by Pang et al.

Partitioning

Partitioning is the subdividing of possible values into subsets. Slagell et al. consider black marker anonymization and truncation to be examples of partitioning. Black marker is an extreme where only one partition exists, while truncation creates many partitions depending on the amount of truncated bits. This could also be called aggregation or binning. This is similar to the idea of k-anonymity [95]. A special case of partitioning featured in FLAIM is time unit annihilation. Timestamps are broken down into year, month, day, hour, minute and second subfields which can then be removed in order to influence the time precision.

Burkhart et al. [9] argue that permutation-based IP address anonymization is not sufficient and propose the use of truncation. They evaluate the risk-utility trade-off of truncation in the case of detection of scans and denial of service attacks. Depending on the metric they use different sizes of truncation are acceptable. In the case of their example and data set, truncating up to 8 bits was acceptable in the best case for internal addresses and 20 bits for external addresses. Depending on the size of the network and other parameters, as well as the use-case, truncating more or less bits may be acceptable. Evaluating the acceptable amount of truncation is necessary before applying truncation if a good balance between risk and utility is to be achieved. One way to achieve this is ensuring that each address is indistinguishable from k-1 others, k-anonymity [95].

Permutation

Slagell et al. consider permutation the one-to-one mapping on a set. They consider block ciphers for permutations on large binary fields, and even IPv6

addresses due to their size of 128 bits, however note that there are no strong 32-bit block ciphers, making them less effective on Internet Protocol version 4 (IPv4). For this case they mention the possibility to use tables, as is done by tcpdpriv [62]. The tables used by tcpdpriv are an issue if one wants to keep their anonymization constant as tcpdpriv's table-based anonymization is not consistent between traces and depends on the order in which addresses appear. The tables are also large. For this reason the cryptography-based IP anonymization by Fan et al. [38] is a better solution in such cases. In some cases it may be necessary to preserve a relationship between values across packets. Slagell et al. provide the example of timestamps and preserving distances, stating that a simple shift can be used in this case. Finally, it may be necessarys to preserve information about the structure of a network. This requires prefix-preserving anonymization, which is unique to IP addresses. Prefix-preserving anonymization is defined by Fan et al. [38] as a one-to-one function where, if two IP addresses share the same k-bit prefix, their anonymized counterparts will share the same k-bit prefix.

Hash

Hash functions can be used to anonymize a variety of fields, however small protocol fields may require hashes to be truncated, resulting in a higher likelyhood of collisions. Slagell et al. state that dictionary attacks become very practical for fields 32 bits and smaller. This is confirmed by Brekne et al. [8] who computed Message Digest 5 (MD5) hashes for the entire IPv4 address space in "hours" on what they considered a "regular PC". The exact specifications of this Personal Computer (PC) and exact time were not revealed in their paper, however the Nvidia GTX1080Ti graphics card, launched March 10, 2017., is capable of calculating 30000 million MD5 hashes per second making it capable of calculating MD5 hashes for the whole IPv4 address space (around 4.3 billion addresses) in 0.14 seconds [45]. For the IPv6 address space this is still not feasible, but may be if an adversary is able to reduce the address space from the whole IPv6 space to a smaller space using additional information. Slagell et al. differentiate between hash and Hash-based Message Authentication Code (HMAC) in their list of supported anonymization methods, however due to their similarity it would make sense to group them into one category. HMACs are calculated using a secret key, making it impossible for an adversary to perform a dictionary attack since they would have to not only calculate all the hashes, but do so for every possible secret key. Like with hashes, collisions are also possible when using HMACs, so if collisions are not permissible neither HMACs nor hashes

should not be used.

Enumeration

Enumeration assigns an initial value to an element and then increments this value for future elements. For example, an initial timestamp could be chosen and incremented for future timestamps. This preserves information about the order of events, but removes details such as the exact time of an event or the distance between events. Pang et al. [72] refer to this method as sequential numbering.

2.3 Legal environment

When talking about the problem of anonymization of any kind, which includes packet capture anonymization, the legal environment must be considered. In this section law affecting anonymization of network captures in the EU and US is discussed.

EU

In the EU, the GDPR makes the importance of anonymization very clear. According to article 4 of the GDPR: "'personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person; ."[73]

In the case of Patrick Breyer vs Germany the European Court of Justice ruled that "with regard to the processing of personal data and on the free movement of such data [...] *a dynamic IP address* registered by an online media services provider when a person accesses a website that the provider makes accessible to the public *constitutes personal data* [...] where the latter has the legal means which enable it to identify the data subject with additional data which the internet service provider has about that person"[54]

Packet traces can contain sufficient data for IP addresses to be considered personal data, and as such these values are an obvious choice for anonymization, but following the logic of article 4 and the ruling in Breyer vs Germany there is potential for other fields to contain sufficient information to

narrow down an individual, either on their own or in combination with other gathered information, resulting in these fields becoming personal data.

US

Regarding the US, [11] identifies the ECPA as the main source of protection. The three main components of the ECPA: The Wiretap Act, Pen Register statute and SCA limit the sharing of network captures.

The SCA [92] states "a *provider* of remote computing service or *electronic communication service* to the *public* shall not *knowingly* divulge a record or other information pertaining to a subscriber to or customer of such service [...] to any governmental entity." In this example the term provider is defined in 18 United States Code (U.S.C.) §2258E [90] as an electronic communication service provider where an electronic communication service is defined in 18. U.S.C. §2510 [91] as "any service which provides to users thereof the ability to send or receive wire or electronic communications."

The author of [11] notes that "to the public" is an important phrase here as it limits the applicability of the law. A company's internal network which is accessible only by employees would not be providing services to the public. This becomes a bit more of a legal gray area when it comes to, for example, universities. The author also notes that "knowingly" is not defined in the SCA, but legal precedent in the case of Freedman v. AOL, Inc [98] defines it as an awareness of the consequences of the action. It could be argued that a faulty anonymization scheme or one circumvented by a new solution does not render the author of a packet trace guilty as they were not aware that they were divulging information. It is also important to note that the SCA prohibits divulging this information to a governmental entity. While this obviously makes it illegal to share an unanonymized trace directly with a governmental entity, it is not clear if this would also apply to a trace shared with the public in a way that makes it accessible to a governmental entity, but it would be reasonable to assume it would.

The Wiretap Act [91] and Pen Register statute [93] can be seen as two sides of the same coin. The Wiretap act prohibits the capture and disclosure of content but does not state anything regarding non-content, such as addressing information like IP addresses, while the Pen Register statute regulates the use of pen registers which are used to capture "dialing, routing, addressing, or signaling information." [94] Both the Wiretap Act and the Pen Register statute relate to real-time collection of data. While that means they are not of consequence for offline anonymization (although they could be of consequence for the data collection mechanism prior to the anonymization

process), both the Wiretap Act and Pen Register statute could potentially affect online anonymization.

2.4 Attacks on anonymization

Attacks on anonymiztion are methods by which the original data present in the capture before anonymization can be revealed or contextual information present in a capture can be used to reveal information about the original state of the network in which the capture was taken that was not intended to be shared through the capture in question.

Attacks on various anonymization schemes have been explored by many researchers. Fan et al. [38] consider the effects of attacks using cryptanalysis techniques and knowledge of compromised unanonymized-anonymized address pairs and exploiting the semantics of prefix-preserving anonymization. They call these cryptographic and semantic attacks. Pang and Paxson [71] look at inference attacks. These are attacks that can be used to infer sensitive information from traces. Their taxonomy is discussed later in this section, together with other taxonomies of attacks. Bethencourt et al. [3] demonstrate how the Internet Storm Center's internet sensors, devices which submit logs for the purpose of various reports, can be mapped by probing them with activity that will be reported and analyzing the reports. Pang et al. [72] notice their routers sometimes send Address Resolution Protocol (ARP) requests for an entire subnet in quick succession which could be used to partially deanonymize IP addresses. Coull et al. [27] demonstrate how network topology can be inferred from anonymized traces and the behavior of hosts in anonymized traces can be used to find their real-world counterparts. Slagell and Yurick [85] create a taxonomy of attacks similar to that of Pang and Paxson. This taxonomy is likewise discussed later in this section. Kohno et al. [57] demonstrate how the clock skew, small deviations in device time-measuring hardware, can be used to fingerprint a physical device. A detailed analysis of the attacks mentioned here is outside of the scope of this thesis, however knowledge of their existence and the broader class of attacks they represent is important in order to develop proper anonymization policy. The two taxonomies mentioned here, as well as a third one developed by King et al. are discussed in chronological order below.

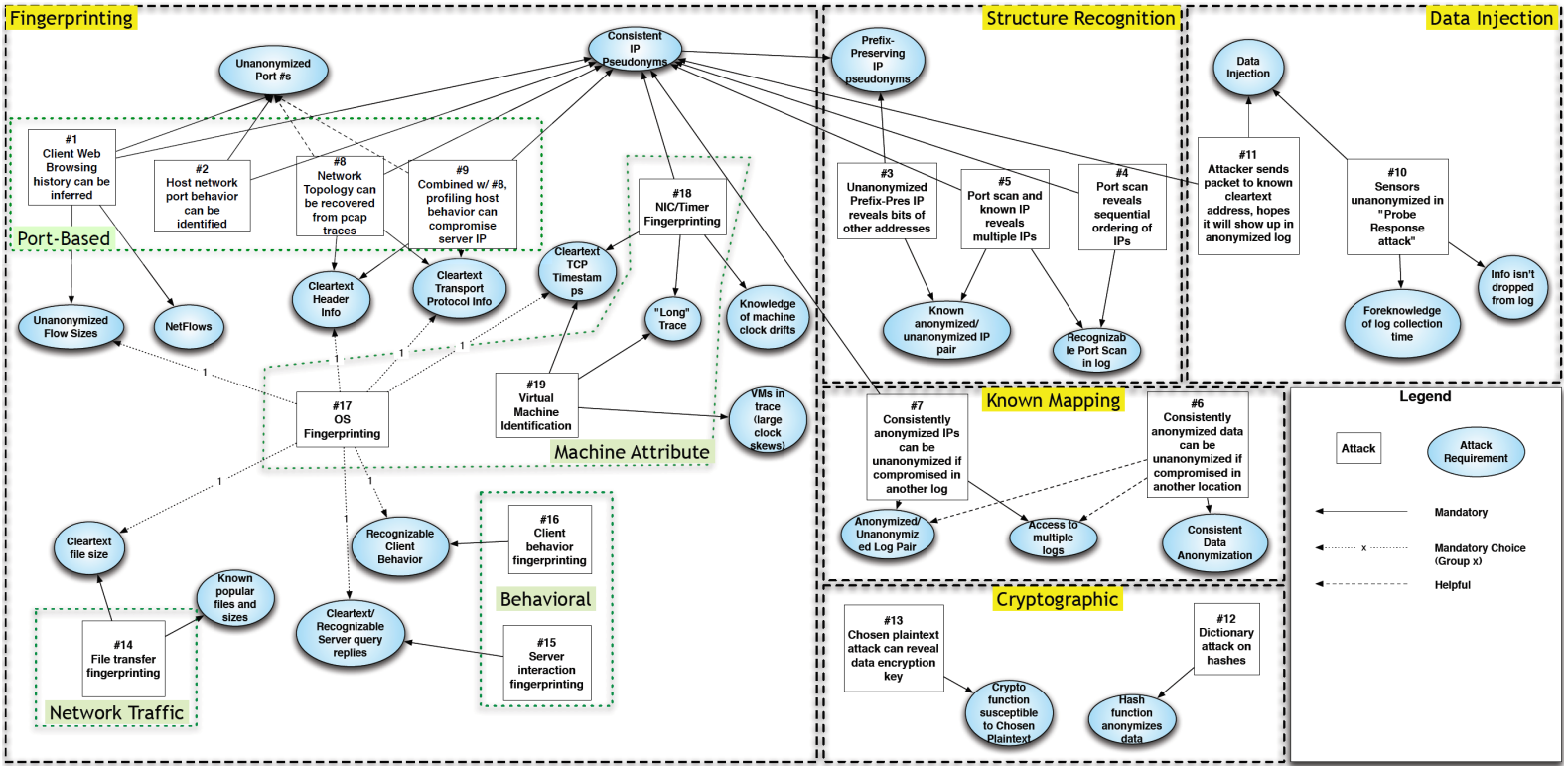

Figure 1: Grouping of anonymization attacks by King et al. [56]

A taxonomy of attacks

A taxonomy of attacks against anonymization can help develop an anonymization policy that best protects the log provider from attacks while stil maintaining the utility necessary for researchers and be used to define the strength of such an anonymization policy, which can be expressed as the amount of attacks it protects against.

In [71] Pang and Paxson look at inference techniques an adversary might use in order to strengthen their anonymization. Pang and Paxson divide attacks into the following groups:

- **Fingerprinting:** Comparing attributes of an anonymized object to those of a known object
- **Structure recognition:** Recognizing structure between objects in the anonymized trace which may aid in inferring their identities (e.g. scans)
- **Shared-Text matching:** Attributes shared between anonymized objects can be used to expose one when the other is exposed. For example in prefix-preserving anonymization deanonymizing one host also deanonymizes the shared prefix.
- **Known-Text matching:** Knowing both the anonymized and the deanonymized value can lead to deanonymizing all appearances of a value in a trace.

Pang and Paxson consider the possibility of an attacker injecting traffic of their own in order to aid with fingerprinting and known-text matching attacks, but do not classify this as an attack of its own, instead calling it an active method of the attacks specified. While they do consider scans by attackers as helpful for structure-recognition, they do not discuss the possibility of the attacker seeking to aid themselves in later structure recognition by performing the scan themselves.

In [85] Slagell and Yurick create their own taxonomy and note its similarity to the taxonomy proposed by Pang and Paxson. Slagell and Yurick propose the following taxonomy:

- Fingerprinting
- Structure Recognition
- Known Mapping Attacks
- Data Injection

- Cryptographic attacks

The definitions of fingerprinting, structure recognition and known mapping attacks used by Slagell and Yurick are identical in meaning to those used by Pang and Paxson. Additionally, they consider data injection, the attacker injecting data they can later recognize in the trace, as a separate attack. Slagell and Yurick also note that attacks can be mounted against cryptographic algorithms used in anonymization, however other attacks in their taxonomy will usually be easier to carry out.

In [56] King et al. develop a taxonomy of attacks based on nineteen attacks from other research papers with the aim of developing a complete and mutually exclusive set of classes of attacks. To do this they take into account pre-conditions, knowledge an attacker must have or be able to glean from the network log or capture. Based on these pre-conditions King et al. construct the graph in figure 1. For the sake of classifying attacks King et al. ignore consistent IP anonymization due to many attacks relying on it. The graph of attacks and prerequisite knowledge generated as a result shows a clear separation into six subgraphs. King et al. merge two of the resulting subgraphs, both of which involve cryptographic weaknesses, resulting in the five classes of attacks in their taxonomy. The resulting set of attack classes is virtually identical to the classes proposed by Slagell and Yurick. King et al. criticize the previous taxonomy of Slagell and Yurick for not being fine-grained enough to support their goal of creating a policy that maintains high levels of both security and utility. In order to support that they create subclasses of fingerprinting attacks:

- **Port-Based:** Fingerprinting devices based on well-known ports in use or statistically derived port numbers
- **Machine Attribute:** Fingerprinting based on machine characteristics such as time drift or operating system characteristics
- **Behavioral:** Fingerprinting devices based on machine or user behavior
- **Network Traffic:** Fingerprinting based on recognizing patterns in network traffic.

While their high-level taxonomy seems to hold up to the requirement of mutual exclusivity, the port-based and network traffic fingerprinting subclasses of fingerprinting attacks do not seem to do so. King et al. specifically mention ”being able to extract port numbers from statistical analysis” as part

of port-based fingerprinting, which would seem to overlap with network traffic fingerprinting as the method for statistically revealing a service (and thus its port number) is based on recognizing the unique patters in network traffic that characterize particular applications. Using traffic patterns and characteristics to identify services without the knowledge of port numbers has been done by Early et al. [36] and Collins et al. [16].

Although in Figure 1 King et al. place Operating System (OS) fingerprinting in the Machine Attribute fingerprinting subclass of the fingerprinting class, they note that it is a problem for their classification as OS fingerprinting can be done in a manner that could place it in any of the subclasses. As such they express the view that OS fingerprinting is best viewed as a group of attacks rather than a single attack.

2.5 Extended scope

Alternatives to packet traces have been proposed. These alternatives fall outside the scope of this thesis, but they are worth mentioning as they highlight some of the issues of packet trace anonymization and contribute to the discussion of packet trace anonymization in ways that can fuel improvement.

Secure queries

Mirkovic [63] argues anonymization of packet traces is a poor solution because it offers insufficient research utility and poor privacy. They look at papers published to Special Interest Group on Data Communication (SIGCOMM) and the Internet Measurement Conference in 2006 and 2007 and show that only 10 out of 144 papers used public traces while the rest used private traces. They conclude that this is likely due to lower utility of public traces.

They argue traces offer low privacy due to the utility loss required to protect from passive attacks, the inability to defend against active attacks and the inability to protect released traces against future attacks. The difficulty of defending against all passive attacks can be seen from the taxonomy of anonymization attacks by King et al. in Figure 1. A more detailed description of this taxonomy is provided in Section 2.4. It is safe to conclude, as Mirkovic does, that a trace obscuring many of the fields used by these attacks would be useless for many researchers. Mirkovic's conclusion that no sanitization can defeat active attacks is also partially corroborated by Burkhart et al. [10] who conclude that by stretching patterns in time an attacker can always manage a successful attack while staying undetected. However, the

attacker does need to know when the packets will be recorded and the trace has to be of sufficient length or the pattern sufficiently visible, both of which make this form of attack more difficult than it may seem. The third problem, the inability to protect released traces from future attacks, is evident, and can not be solved in any way when releasing packet traces publicly. Once a trace is released, it is released.

Mirkovic's system of secure queries would improve upon packet traces in many ways. It would improve privacy by providing fine-grained control over permissible queries. The system could also log queries which could then be audited to reveal misuse and help improve policy. Mirkovic states that many active attacks can be prevented by a combination of restricting possible queries, restricting results and through auditing. Additionally, the system would offer better protection against future attacks as policy could be updated to protect against new attacks, and only those users who have run queries that future policy forbids would be able to run these attacks.

There are also downsides to this approach, which Mirkovic acknowledges. The approach does not make it possible to view raw packets, which researchers may need. For this problem Mirkovic proposes exploratory research be done on private traces until a point where the research can benefit from aggregate results such as those provided by secure queries. Operations needed by some may be missing, but these can be added. Mirkovic does not address the centralization of their system as a potential security flaw. Assuming all the queries are secure and policy is kept up to date, preventing the queries themselves from being used in an attack, the system proposed by Mirkovic still has unanonymized traces stored on the system and a flaw elsewhere in the system could possibly expose the unanonymized data from their database. Additionaly, organizations may have policy preventing them from storing unanonymized data, as mentioned by Pang et al. [72]. Storing data in an unanonymized form can also lead to legal issues as personal information may be contained within. Data longevity is also a potential downside of this approach, as no incentive is suggested for data providers to store their data for any amount of time, and no incentive for hosting data at all is provided.

SC2D

The Shipping Flexible Analysis Code to the Data (SC2D) framework, developed by Mogul et al. [65], proposes to solve the problems of packet trace anonymization in a way similar to the proposal made by Mirkovic. However, instead of queries, Mogul et al. propose a framework based on modules that

support commonly used analysis methods and a high-level domain-specific interpreted language. Researchers would develop modules and send their analysis code to the trace owners who would then run it on their systems and return results. In this way the confidentiality of traces can be preserved while providing researchers with the information they need. Mogul et al. do not provide their language in their paper, so the strengths and weaknesses can not be compared to those of the secure queries proposed by Mirkovic, but such detailed comparisons would be out of the scope of this thesis anyway.

Mogul et al. propose a system of expert reviews for their framework and modules. Expert review would ensure modules and analysis code truly do what is claimed and do not contain malicious code. The expert review process does suffer from several potential issues, both technical and social in nature, which Mogul et al. acknowledge. The technical issues are the design of analysis code, modules and the framework in a way that makes reviewing them and finding issues easy. Mogul et al. also propose signing reviewed code as a way of ensuring the code delivered to a data owner is the same code that was signed off on by the expert reviewers, and automatic leakage detection as an addition to the review process that automates detecting privacy leaks in analysis code. The social issues recognized by Mogul et al. include the choice, potential payment and potential liability of experts, as well as the detection of attempted subversions of the review process and the confidentiality of a researchers work.

The system proposed by Mogul et al. suffers from the same drawbacks as secure queries proposed by Mirkovic: inability to view raw packets, centralization, the possibly short lifespan of data and lack of incentives for data providers to store data and allow the execution of code. Additionally, SC2D also makes debugging difficult as bugs might reveal themselves during runtime on provider data that could not be detected using a researcher's own data. Debugging would also be made more difficult by the lack of insight into the data that would help reveal the bug. Fixed code may also have to be resubmitted for expert review, potentially making the process time consuming. Unlike Mirkovic, who does not appear to consider the option, Mogul et al. do consider online operation. This means organizations with policies against storing data can participate, however in the case of SC2D, since analysis is performed and then the data is discarded, it also means reproducibility is an issue.

3 Features of anonymization frameworks

One possible solution to the shortcomings of existing anonymization tools [2] are frameworks such as FLAIM [86], developed by Slagell et al., tcpmkpub [72], developed by Pang et al. and PktAnon [41] developed by Gamer et al. These frameworks are designed with features that make them useful to a wider variety of users. In this chapter the following taxonomy of features is proposed, based on features discussed and mentioned by the authors of the three previously mentioned frameworks:

- **Extensibility:** The ability to extend a framework with support for new protocols or anonymization methods
- **Policy support:** Support for a detailed anonymization policy
- **Meta-data generation:** Generation of additional meta-data
- **Large file support:** Support for capture files containing gigabytes or terabytes of data
- **Consistency:** Support for consistent anonymization across multiple logs
- **Performance:** Executing anonymization tasks in a time that is reasonable for the policy employed
- **Online anonymization:** The ability to anonymize captured packets as they arrive at a network capture card or interface

In addition to these features, no authors to our knowledge discussed **recoverability**, the ability to recover from failure or run additional operations on an anonymized capture after a policy has been reviewed and found lacking. A more detailed discussion of the features mentioned can be found below.

Extensibility

Extensibility is the ability of a framework to be extended in order to support new protocols or anonymization methods.

[2] Based on the features described in this chapter, a differentiation between anonymization tools and anonymization frameworks is made. Tools are mentioned in Section 2.1. Their features according to the taxonomy in this chapter are shown in Table 1.

In FLAIM extensibility is achieved through a modular architecture and a core consisting of an anonymization engine and policy manager. Users can create modules that support different types of logs and the protocols in these logs, and create flexible policy to anonymize these logs using the anonymization engine. Users adding new anonymization primitives is not an intended feature.

Tcpmkpub allows users to specify their own C++ functions to execute over fields of the supported protocols which would allow for new anonymization methods. Tcpmkpub does not have an in-depth understanding of the protocols it anonymizes, instead relying on the anonymization policy to also specify names and lengths of fields. The way in which this is done suggests that tcpmkpub recognizes the protocol itself, despite having no deeper knowledge of its structure. This may make extending tcpmkpub to support new protocols simpler by dividing the actions of recognition and handling between the engine and the anonymization policy, but also means a user looking to add a new protocol to anonymize would have to alter the source code of tcpmkpub in order to write in recognition for their protocol.

Adding new protocol support to pktanon also requires modifying pktanon source code. Pktanon handles protocols by first passing the captured ethernet frame from a pcap capture file to a class called packet, which then uses an interface that must be supported by classes for handling different protocols. This means that inserting a new protocol requires knowledge of the interface, writing one's own protocol handler and possibly modifying the packet class to recognize the new protocol. Writing a new anonymization method for pktanon requires implementing an interface that must be supported by anonymization methods.

Policy support

Policy support is the ability of a framework to allow users to write their own policies detailing the mappings of protocol elements to supported anonymization methods.

Gamer et al. mention two important features: flexibility and configurability. They consider flexibility the ability to apply a framework to different environments and objectives, and state that it must be able to anonymize any field with any anonymization method. By configurability they mean the ability for users to define their own profiles, mappings between protocol attributes and anonymization methods.

Slagell et al. [86] state that an anonymization tool needs to be "multi-level", meaning it must support many options for each protocol field which

they consider levels of anonymization. They also state a tool must have a rich supply of anonymization algorithms. These two features are similar in spirit to the flexibility and configurability required by Gamer et al.

Pang et al. do not explicitly name this requirement, but they do state that existing tools did not satisfy their requirements and they develop their framework with the ability to accommodate a wide range of policy decisions. The ability to extend their framework with functions for anonymizing the supported protocol fields by naming C++ functions in pktanon allows for both flexibility and configurability as defined by Gamer et al.

The separation of flexibility and configurability seems somewhat superfluous as a configurable solution would automatically have to be flexible to some degree and existing solutions supporting one supported the other. Instead, we can call the union of these two features policy support.

Meta-data generation

Pang et al. state that meta-data is often crucial for understanding traces and include this additional data with their traces. The same sentiment is mirrored by Paxson [74] who argues that retaining meta-data can be beneficial as a good data-set can be revisited later for different research that may require some of that additional information. While providing meta-data is obviously a boon to researchers and can extend the usefulness of a data-set beyond its initial planned usage, it can also provide attackers with useful information that can help them refine their attacks. [28]

These features were a central part of the design of the frameworks mentioned at the start of this section, but other features that are less prominently featured but also worth mentioning are discussed below.

Large file support

It is important for a anonymization tool to be able to properly handle large capture files. This is necessary since even a 10 Mbps connection, which can be considered low-bandwidth by modern standards, can transfer 108 GB over the course of a day when saturated. Fan et al. [38] mention large trace files with an example of terabytes. Pang et al. [72] mention a trace of 48 gigabytes. Slagell at al. [86] do not explicitly mention a size but state that their framework works admirably when anonymizing large data sets and give a preliminary estimate of a gigabyte per minute. This strongly suggests that they are dealing with traces that are multiple gigabytes in size or more. Mogul and Arlitt [65] use the example of Arlitt having 5 terabytes of trace

data as one of the reasons why their proposed alternative to anonymization, shipping the analysis code to the data, may be better than sharing traces. While these 5 terabytes are not stated to be one file, it appears reasonable to believe several multi-gigabyte files may be present. A more recent example of capture sizes is available on the website of the Center for Applied Internet Data Analysis (CAIDA) [12]. While they do no explicitly mention the size of their captures, they provide a mean transmission rate, which can be used to estimate the size when combined with the duration of the capture. This results in an estimated capture size of around 2 Terabytes in the case of their most recent capture.

Consistency

Fan et al. [38] provide several examples that demonstrate a necessity for consistent anonymization of packet traces. Traces from different sites may better be anonymized simultaneously on different sites instead of being first collected and then anonymized. This would also be safer than sending unanonymized traces off-site for anonymization. A large trace may also be anonymized more quickly if it were possible to break it down into multiple sub-traces and anonymize these separately with a consistent scheme. Consistency is not without its downsides, however, as King et al. [56] note that 9 out of the 19 attacks they looked at when developing their taxonomy required some form of consistent IP pseudonyms.

TCP sequence and acknowledgement numbers present an issue when attempting to maintain consistency. In order to anonymize TCP sequence and acknowledgement numbers correctly, it is necessary to maintain a large amount of state information, or anonymize a packet trace in two passes. This also presents an issue when anonymizing sub-traces, as maintaining a consistent sequence between such traces would not be possible. Problems encountered with anonymizing TCP sequence and acknowledgement numbers during Shanon development are discussed in Section 8.3.

Performance

While performance has not been the focus of any of the developed tools or frameworks for anonymization, many authors have included some form of performance metric, usually relating to the speed of processing capture files. Slagell et al. [86] mention that their framework can handle a gigabyte per second. Fan et al. [38] express the performance of their anonymization method for IP addresses in terms of packets per second.

Their cryptography-based anonymization scheme can process 10,000 packets per second on an 800 Mhz Intel Pentium III processor, which they say is “fast enough for practical purposes”. This shows that, despite performance not being the main goal of their new scheme, they still consider it an important metric. Pang et al. [72] look at the time and peak memory usage required to anonymize their trace. Their framework, tcpmkpub, anonymized a trace containing 165 million packets, with a total size of 48 Gigabyte (GB), in 2.9 hours, having used a peak of 331 Megabyte (MB) of memory. Gamer et al. [41] express the view that performance is very important for online anonymization and evaluate the performance of various parts of their framework. They conclude that further work needs to be done to improve the speed of their framework. It seems obvious from the inclusion of performance metrics and given the possibly large size of capture files as mentioned previously, that performance should not be ignored when developing an anonymization tool or framework.

Online anonymization

Online anonymization is the ability to anonymize captured packets as they arrive at a network capture card or interface. Pang et al. [72] note that some organizations may require even internally stored traces be anonymized. They do not focus on this further as their organization does not require it and their anonymization policy requires multiple passes, preventing online anonymization. This also reveals the possible issue that online anonymization may not always be amicable with every possible policy regarding anonymization. Gamer et al. [41] mention legal requirements may make it necessary to anonymize data as soon as possible and state the opinion that that requires online anonymization.

Other features

Slagell et al. [86] state that a tool should be multi-log capable, a feature they describe as “being flexible enough to support the anonymization of most security relevant logs without major modification.” While the subject of this thesis is not the sharing of logs, but network captures, the ability to share additional logs and anonymize them consistently can aid research. Slagell and Yurick [85] expressed a belief in a need for standards in anonymization. A standard defining types of anonymization that need to be supported and ways to express the needs of different organizations would greatly improve the space and allow for different types of logs to be anonymized in a consistent, standard manner.

A feature not suggested by any author to our knowledge is **recoverability**. Pang et al. [72] took 2.9 hours to anonymize their trace using their framework, tcpmkpub, and the policy developed in their work. It is possible that an anonymization framework could crash or the process could be left unfinished due to a power outage or other issues. Being able to recover from such an issue and continue anonymization or run additional operations on an already anonymized trace could save valuable researcher time.

Cross-Comparison

Using the above features as a basis we can now provide an overview of the tools mentioned in Chapter 2. This overview can be seen in Table 1. The three frameworks used as a basis for developing this taxonomy are not present in the table both as a means of conserving space due to the size of the table and because they possess all of the listed features.

Assumptions are made in favor of the tools wherever any reasonable proof of feature support was provided, such as the presence of performance metrics for performance or a more generalized configurable anonymization scheme for policy support. The presence of configurable anonymization methods that could yield consistent anonymization across traces were counted towards consistency. Most tools do not explicitly state their support for large files, but this can be assumed for many of the tools based on context provided in the papers describing them and in their manuals. For example, any tool capable of online anonymization can just as well accept a stream of packets from a file. The same could be said for any tool based on libpcap[35] which supports large files. Any additional data that can be generated in addition to an anonymized trace and used to provide additional insight was counted as meta-data generation. This appears to only be supported by TraceWrangler. Meta-data support in TraceWrangler consists only of support for importing Snort IDS[15] alert files to mark packets detected by the IDS. Still, this meta-data can be of use to security researchers and as such has value.

Features	TraceWrangler	WireEdit	SafePcap	CryptoPAn	Tcpdpriv	SCRUB-tcpdump
Large File Support	–	Yes	Yes	Yes	Yes	Yes
Consistency	Yes	–	Yes	–	–	Yes
Online anonymization	–	–	–	–	–	Yes
Policy support	–	–	Yes	–	–	Yes
Extensibility	–	–	–	–	–	–
Meta-data generation	Yes	–	–	–	–	–
Performance	–	–	Yes	Yes	–	–
Recoverability	–	–	–	–	–	–

Table 1: Features supported by examined anonymization tools.

4 Choosing initially supported protocols

This chapter explains the choice of protocols to support in the tool developed as part of this thesis, provides an overview of the layered architecture of network protocols, and overviews of each of the chosen protocols.

The initial choice of protocols to anonymize in this work is based on the protocols anonymized by existing tools and frameworks. Table 2 paints a clear picture about the needs of organizations and researchers when it comes to protocol anonymization. All of the tools that were looked at supported anonymization of IPv4, showing a clear focus and recognized need for anonymization support. User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) are clearly also considered very important as they were only not anonymized by CryptoPAn, the focus of which is solely IPv4. The relative lack of support for Ethernet is understandable to a degree as the utility of a Media Access Control (MAC) address to a would-be attacker seems to end at the ability to consistently identify a device and its make and model. Still, it could be argued that even that is too much information to expose unnecessarily, and in light of US and EU law a MAC address could potentially be considered addressing information regulated by the Pen Register statute or personal information regulated by the GDPR. ARP and Internet Control Message Protocol (ICMP) are also somewhat neglected, but seemingly for a different reason. The utility of ARP and ICMP in diagnosing network issues and understanding the topology of a network is a double-edged sword. While it can certainly be of interest to researchers it can also be utilized by attackers to help uncover the topology of the network despite efforts to conceal it through the merging of subnets or flows, remapping of addresses or simply use of a non-prefix-preserving scheme. As an added complexity ICMP messages may contain part of a packet. This part would need to be recognized and anonymized consistently with the captured packet itself, which may require multiple passes of a file be made or in the case of online anonymization sufficient memory to store reasonably sized chunks of packets for comparisons to incoming ICMP messages for a sufficiently long time, which may be more overhead than justified by the utility of ICMP messages. That IPv6 support is low comes as no surprise as adoption is sadly still not widespread, and was far less so more than a decade ago when many of these anonymization tools were written .[44] It is also important to note that the full required set of IPv6 extension headers that must be supported by a full implementation of IPv6 according to RFC8200[31] are not supported by TraceWrangler. WireEdit and SafePcap do not explicitly state support for extension headers, but given their impres-

sive list of supported protocols it is reasonable to assume.

Protocols	Trace Wrangler	WireEdit	SafePcap	CryptoPAn	Tcpdpriv	SCRUB-tcpdump	tcpmkpub	FLAIM	PktAnon
Ethernet	Yes	Yes	Yes	–	–	Yes	Yes	Yes	Yes
ARP	Yes	Yes	Yes	–	–	–	Yes	Yes	Yes
IPv4	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IPv6	Yes	Yes	Yes	–	–	–	–	Yes	Yes
ICMP	Yes	Yes	Yes	–	–	–	Yes	Yes	Yes
UDP	Yes	Yes	Yes	–	Yes	Yes	Yes	Yes	Yes
TCP	Yes	Yes	Yes	–	Yes	Yes	Yes	Yes	Yes

Table 2: Protocols supported by at least half of the examined anonymization tools and frameworks

The protocols chosen to be supported in the initial version of Shanon, the tool created in this thesis, are:

- Ethernet
- IPv4
- IPv6
- ARP
- ICMP
- ICMPv6
- NDP
- UDP
- TCP

In additional to those supported by at least half of the examined tools, as seen in table 2, Internet Control Message Protocol for the Internet Protocol Version 6 (ICMPv6) has been added to the list as the IPv6 counterpart of ICMP. Neighbor Discovery Protocol (NDP) has also been added as it uses the same general message format as ICMPv6 and replaces ARP in IPv6 networks, along with additional functionality. Given the ubiquity of wireless communication nowadays it also seemed prudent to include 802.11 support, however upon further review dealing with 802.11 was found to be far too complicated and worthy of a thesis in and of itself. This decision was further simplified by the fact that capturing 802.11 traffic is often not possible without running a network card in promiscuous mode, a mode where all traffic the card receives can be captured, rather than only the traffic the card is meant to receive. Outside of promiscuous mode, many network cards transform 802.11 data packets into Ethernet packets before supplying them to the operating system network stack [24].

A more detailed overview of each of the supported protocols follows. The tables showing each of the protocols merge fields smaller than 1 byte and fields with lengths that are not a multiple of bytes. The reason for this alternative division of protocol fields is the way in which these fields can be retrieved from Wireshark in their raw form, as unmodified bytes of data such as they are in capture files. This is done by returning a range of bytes from the Testy Virtual Buffer (TVB), a buffer containing the bytes of a captured protocol. As such it is not possible to retrieve a field that is less than 1 byte in

size or does not end on a 1-byte boundary in this way without also retrieving parts of, or entire other fields, such as the Internet Header Length (IHL) field when retrieving the Version field. Organizing protocol fields in such a manner makes it easier to refer to these tables when actually working with the protocol data or looking at the bytes stored in capture files. In several places the lengths of fields may be marked with the letters “M” and “N”. These signify fields of variable lengths and may both be in use to clearly distinguish between fields that have variable lengths, but are not necessarily of the same length.

4.1 Layered network model

Network protocols operate in a layered architecture where each layer is responsible for providing layer-specific services to the layer above it. Lower-layer protocols encapsulate the protocols from higher layers, adding their own header before the higher-layer protocol header, and in some cases also adding a trailer after the higher-layer protocol data. The higher-layer protocol contents are the payload, or data, of the lower layer protocol.

The goal of a layered architecture is to separate different functions on different layers, each providing its services to the layer above and relying on the services of the layer below. However, in some cases the operation of protocols crosses these layer boundaries. TCP and User Datagram Protocol (UDP) pseudo-headers, described in Section 4.8, contain fields from the lower-layer protocol. In this thesis only IPv4 and IPv6 pseudo-headers are examined, but for other protocols to carry TCP or UDP traffic appropriate pseudo-headers need to exist. ICMPv6, described in Section 4.6, also uses an IPv6 pseudo-header.

Multiple models exist which describe the layered architecture of network protocols. For the purpose of this thesis, the TCP/IP 5-layer model [58] will be used. This model consists of the following layers:

- Application - Network applications and their protocols
- Transport - Transports application-layer messages between applications
- Network - Transports network-layer packets between hosts
- Link - Transports packets between nodes along a route
- Physical - Transports individual bits across a transmission medium

The protocols supported by Shanon and their place in this model can be seen in Table 3.

Layer	Protocols
Application	
Transport	TCP,UDP
Network	IPv4,IPv6,ARP,ICMP,ICMPv6,NDP
Link	Ethernet
Physical	

Table 3: The TCP/IP model and protocols supported by Shanon

4.2 Ethernet

The Ethernet frame [52], as seen in table 4, consists of 5 fields. The Destination and Source Address fields contain the MAC addresses of the destination and source of the frame. These are 6-byte unique addresses. The type/length field either carries the length of the Ethernet frame or type of the higher layer payload, depending on the value in the field. The data field is the encapsulated higher-layer packet. The Frame Check Sequence (FCS) is used to verify the integrity of the received frame.

Depending on the value in the Type/Length field, the field contains either the length of payload or the type of payload the frame is carrying. According to the 802.3 standard, a value “less than or equal to 1500 decimal” is meant to denote the length of the data field in octets. Such a frame is called a basic frame in the standard. If the value of the type/length field is “greater than or equal to 1536 decimal”, the field instead denotes the type, called the Ethertype, of the protocol encapsulated in this frame.

A size of 1504 denotes a Q-tagged frame. These are frames including a 4-byte 802.1Q header used for the logical separation of a physical network into Virtual Local Area Networks (VLANs).

A size of 1982 denotes an Envelope frame. These are frames which “allow inclusion of additional prefixes and suffixed required by higher layer encapsulation protocols.”

Q-tagged and Envelope frames will not be supported in this work.

The Ethernet FCS is a cyclic redundancy check that is calculated using all fields but the FCS. In many cases the frame check sequence may not be present at the end of a captured frame as it is not always provided to the higher layers by the network card.

Protocol	Field	Size (Bytes)
Ethernet	Destination Address	6
	Source Address	6
	Type/Length	2
	Data	46-1500, 1504 (.Q), 1982
	FCS	4

Table 4: Ethernet

4.3 IPv4

The Internet Protocol version 4 (IPv4) [78] header can be seen in table 5, and consists of 13 fields plus an optional options field which can carry a variable amount of options. As previously mentioned, fields the size of which is less than 1 byte, such as Version and Internet Header Length are merged into a single field in the table, then divided into subfields.

The IPv4 Version field is used to denote the version of the IPv4 header in use. For IPv4 this field contains the value 4 decimal.

The IHL field contains the length of the IPv4 header in 32-bit words, and has a minimum value of 5, meaning 160 bits, for an IPv4 header without any included options.

The Differentiated Services Code Point (DSCP) field and Explicit Congestion Notification (ECN) field were originally the Type of Service field.[78] Differentiated Services were defined in Request for Comments (RFC) 2474[68] and are used to classify and manage network traffic for quality of service purposes. ECN is defined in RFC 3168[80] and is used to indicate congestion in the network. A more detailed look at the mechanisms of either of these fields is outside of the scope of this thesis.

The Total Length field contains the length of the IPv4 packet in bytes. This length includes both the header and the data.

The Identification field is used to identify fragments of a fragmented packet for the purposes of reassembly. Other uses of this field are prohibited. [96]

The Flags and Fragment Offset fields are used for the purposes of fragmentation, the process of splitting a larger packet into several segments, which may be necessary if a packet can not be transported across a segment of a network in one piece due to size constraints . The Reserved flag must be zero and is currently not in use. The Do Not Fragment flag is used to indicate the fragmentation of a packet is not permitted. The More Fragments flag is set when a packet is not the last fragment of a fragmented packet. The Fragment Offset field indicates where in a fragmented packet

the fragment belongs. The unit of size for the Fragment Offset field is 8 bytes (64 bits) and the first fragment starts at 0.

The Time To Live (TTL) field is used to denote the maximum time a packet is allowed to exist. RFC 791[78] originally intended for this time to be in seconds, with processing times less than a second being rounded effectively rounded up by a minimum decrease of 1 each time a packet was processed. In practice this field now serves as a hop limit.

The Protocol field is used to identify the next layer protocol. These values are managed by The Internet Assigned Numbers Authority (IANA) and can be found online at [51].

The Header Checksum is "the 16 bit one's complement of the one's complement sum of all 16 bit words in the header."[78] For the purposes of calculating the checksum, the field is set to 0. This means that 16-bit words from the header are added together as 16-bit integers and whenever there is an overflow, the overflow bit is discarded and the sum incremented by 1. This can result in another overflow, meaning it may be necessary to perform this operation twice. This method of calculation is inefficient, as it requires 2 checks every time 2 16-bit words are added together. In practice this checksum can quickly be calculated at the price of additional memory space by adding 16 bit words from the header to a 32-bit integer, then subtracting 65535 ($2^{16} - 1$) until the 16th bit is no longer 1.

The Source and Destination IPv4 addresses are 32-bit IPv4 addresses denoting the source and intended destination of the IPv4 packet.

The IPv4 Options field is often stated to be rarely used. In their technical report [39] Fonseca et al. find that IPv4 Options are not well supported on the Internet, with their work finding that approximately half of the paths their packets took dropped packets with IPv4 options, as well as options causing a small increase in end to end latency. Due to their seemingly low usage, IPv4 Options are not reviewed in detail or supported by this work.

Protocol	Field	Size (Bytes)	Subfields	
			name	size(bits)
IPv4	Version/IHL	1	Version	4
			Internet Header Length (IHL)	4
	DSCP/ECN	1	Differentiated Services Code Point (DSCP)	6
			Explicit Congestion Notification (ECN)	2
	Total Length	2		
	Identification	2		
	Flags/Fragment Offset	2	Reserved (Flags)	1
			Don't Fragment (DF)	1
			More Fragments (MF)	1
			Fragment Offset	13
	TTL	1		
	Protocol	1		
	Header Checksum	2		
	Source IP Address	4		
	Destination IP Address	4		
	Options	N		

Table 5: IPv4

4.4 IPv6

The Internet Protocol version 6 (IPv6)[31] header can be seen in table 6. The header consists of 8 fields, with the Version, Traffic Class and Flow Label fields being merged in the table as they do not adhere to a clear 1-byte boundary.

The Version field in IPv6 serves the same purpose as in IPv4. For IPv6 this field contains the value 6 Decimal.

The Traffic Class field is currently used for DSCP and ECN the same way as the originally named Type of Service field in IPv4.

The Flow Label field is used to label packets belonging to the same flow.[1] A more detailed look is out of the scope of this thesis.

The Payload Length field in IPv6 specifies the length of the remaining payload, including any extension headers present, in bytes.

The Next Header field identifies the header following the IPv6 header, and uses the same values as the IPv4 Protocol field. These values can be found online at [51].

The Hop Limit field is used in the same way as the IPv4 TTL field, being decremented every time the packet is forwarded in the network.

The Source and Destination Addresses are 128-bit IPv6 addresses of the source and intended destination for the packet.

Unlike IPv4, IPv6 does not include options as part of the header, but rather as separate Extension Headers that can follow the IPv6 header. According to RFC 8200[31], a full implementation IPv6 includes the following headers:

- Hop-by-Hop Options
- Fragment
- Destination Options
- Routing
- Authentication
- Encapsulating Security Payload

The recommended order of these headers is as follows:

- IPv6 header
- Hop-by-Hop Options header

- Destination Options header
- Routing header
- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header
- Upper-Layer header

Each of these headers should only occur once with the exception of the Destination Options header which may occur twice. The Destination Options header before the Routing header is intended for each node along the path, while the one before the upper layer protocol header is intended for the final destination of the packet only. With the exception of the Hop-by-Hop Options header which may appear only after the IPv6 header, IPv6 nodes must accept the headers in any order, but the RFC strongly advises they appear in the recommended order. [31]

To reduce the work required to a reasonable amount, only the extension headers and parameters present in RFC 8200, the IPv6 standard, will be considered in this work. These can be seen in table 7. In addition, a No Next Header extension header exists which carries no data and signifies that no header exists after it. This header has a Next Header value of 59 and its Length field should be ignored. The Hop-by-hop and Destination Options headers can contain different options encoded in a Type-Length-Value (TLV) format, but only padding options are defined in RFC 8200. Similarly, the Routing Header has Type-specific data, but only the header format is defined in RFC 8200. [31]

Protocol	Field	Size (Bytes)	Subfields	
			name	size(bits)
IPv6	Version/Traffic Class/Flow Label	4	Version	4
			Traffic Class	8
			Flow Label	20
	Payload Length	2		
	Next Header	1		
	Hop Limit	1		
	Source Address	16		
	Destination Address	16		

Table 6: IPv6

The IPv6 adoption rate among Google users is around 30% at the time of writing (November 2020) [44].

Option	Field	Size (Bytes)
Hop-by-hop Options	Next Header	1
	Length	1
	Options	N
Routing Header	Next Header	1
	Length	1
	Routing Type	1
	Segments Left	1
	Type-specific data	N
Fragment Header	Next Header	1
	Reserved	1
	Fragment Offset/Res/M	2
	Identification	4
Destination Options	Next Header	1
	Length	1
	Options	N

Table 7: IPv6 Extension Headers

4.5 ARP

The Address Resolution Protocol (ARP) was first proposed as a means of translating a higher layer protocol's addresses into 48 bit Ethernet addresses, with generalizations made to enable the use of ARP on other hardware.

For the purpose of this thesis only the use of ARP with Ethernet and IPv4 will be examined since IPv6 uses NDP for this purpose.

The Hardware Address Space field identifies the layer 2 protocol the request is being made for. These values are managed by IANA and can be found online at [49]. For the purposes of this work only the value 1 Decimal, meaning Ethernet, is of interest.

The Protocol Address Space uses the Ethertype space, the same values used for higher layer protocol types in Ethernet.

The Hardware Address Length and Protocol Address Length are the lengths of the hardware and protocol addresses in bytes. For Ethernet the Hardware Address Length is 6 Decimal. The Protocol Address Length for IPv4 is 4 Decimal.

The Operation Code is used to denote the different types of ARP messages.

The Hardware and Protocol Addresses of Sender and Target contain the layer 2 and higher layer addresses of the sender of the ARP packet and the intended target of the ARP packet. [75]

The different kinds of ARP operations do not appear to be of significance for the work of anonymization, as they involve using ARP with different values in the Hardware and Protocol addresses and different Operation Codes. This does not appear to influence the anonymization of these addresses in any way. If the addresses are anonymized consistently with their anonymization in their respectable protocols, ARP retains its purpose. Should they not be, the ARP operations themselves can still be analyzed but the relationship between them and the devices that sent them may be lost or degraded. It seems plausible that ARP packets could still be linked to the devices that sent them even if values in the packets are inconsistently anonymized by observing the order of messages, but a deeper look into this is outside of the scope of this thesis.

Protocol	Field	Size (Bytes)
ARP	Hardware Address Space	2
	Protocol Address Space	2
	Hardware Address Length	1
	Protocol Address Length	1
	Operation Code	2
	Hardware Address of Sender	N
	Protocol Address of Sender	M
	Hardware Address of Target	N
	Protocol Address of Target	M

Table 8: ARP

4.6 ICMP and ICMPv6

Internet Control Message Protocol (ICMP) and Internet Control Message Protocol for the Internet Protocol Version 6 (ICMPv6), defined in RFC 792 and RFC 4443, are used to communicate errors and for diagnostics. For ICMP the value in the IPv4 Protocol field is 1, for ICMPv6 the value in the IPv6 Next Header field is 58. In IPv6 networks NDP replaces the functionality of ARP. NDP defines new ICMPv6 messages for this purpose and is thus also reviewed here. ICMP and ICMPv6 share a general message

format that can be seen in table 9. Despite sharing the general message format, the values for types and codes are not shared between ICMP and ICMPv6. The checksum is calculated in the same fashion, using the Internet Checksum, which is also used by IPv4, TCP and UDP, however ICMPv6 also includes a pseudo-header for IPv6 in the checksum calculation while ICMP does not do the same for IPv4. The reason behind this is that IPv6 lacks the header checksum that IPv4 has.[76, 17, 66, 31] Since there are many different ICMP and ICMPv6 messages, supporting them all in the initial development of a tool would not be feasible. Therefore an initial choice of ICMP and ICMPv6 messages to support had to be made. We chose to limit the supported ICMP and ICMPv6 messages to those present in RFC 729 (ICMP) and 4443 (ICMPv6) with the exception of those that have been deprecated.

Protocol	Field	Size (Bytes)
ICMP(v4/v6)	Type	1
	Code	1
	Checksum	2
	Body	N

Table 9: ICMP and ICMPv6

ICMP messages described in RFC 729[76] are:

- Destination Unreachable
- Time Exceeded
- Parameter Problem
- Redirect
- Echo and Echo Reply
- Timestamp and Timestamp Reply

In addition to the messages listed above Source Quench messages and Information Request and Information Reply messages are present in RFC 729 but are deprecated and therefore not part of this thesis [42, 43].

ICMPv6 messages described in RFC 4443 [17] are:

- Destination Unreachable
- Packet Too Big

- Time Exceeded
- Parameter Problem
- Echo Request and Echo Reply

These messages, in order of appearance in the ICMP standard, RFC 792, can be seen in table 10. The ICMP header field in the table represents the three fields that are always present in an ICMP and ICMPv6 message and that can be seen at the beginning of the general message format in table 9: The Type, Code and Checksum fields.

Some messages are part of one standard, but not the other. Redirect is not part of ICMPv6, but is present in NDP, an overview of which is provided later. Timestamp and Timestamp Reply messages are only present in ICMP . The Packet Too Big message would be unnecessary in ICMP as routers in an IPv4 network could fragment packets while in IPv6 networks the fragmenting of a packet is performed exclusively by the originating host, requiring a mechanism to notify the host that a segment of the network along the packet's route could not transport the packet due to the Maximum Transmission Unit (MTU) being lower than the packet's size. Thus ICMPv6 has a Packet Too Big message.

One message, the Parameter Problem message, has a difference in size of a field, the Pointer field, between the two protocols. In ICMP the pointer field is 1 Byte in size with 3 Bytes left unused, while in ICMPv6 all 4 Bytes are used for the pointer.

4.7 NDP

In IPv6 networks Neighbor Discovery Protocol (NDP) Neighbor Solicitation and Neighbor Advertisement messages are used in place of ARP. For this reason NDP is included in this thesis. It also includes the Redirect message which is not part of the ICMPv6 RFC, but is present in the ICMP RFC, as well as Router Solicitation and Advertisement which, in ICMP, is another, separate protocol that will not be supported in the initial supported protocols for this thesis as it remains a proposed standard despite existing since 1991 and to our best knowledge does not have widespread usage. NDP makes use of the same ICMP and ICMPv6 general message format seen in table 9 [32, 66].

NDP also contains options. Options are optional parts of the message encoded in a TLV format where the length field is the total length of the option (including the type and length fields) in units of 8 Bytes (64 bits). Zero

Message	Field	Size (Bytes)	
		ICMP	ICMPv6
Destination Unreachable	ICMP Header	4	4
	Unused	4	4
	Original packet data	N	N
Parameter Problem	ICMP Header	4	4
	Pointer	1	4
	Unused	3	0
	Original packet data	N	N
Redirect	ICMP Header	4	Not in ICMPv6
	Gateway Address	4	
	Original packet data	N	
Echo Echo Reply	ICMP Header	4	4
	Identifier	2	2
	Sequence Number	2	2
	Data	N	N
Timestamp Timestamp Reply	ICMP Header	4	Not in ICMPv6
	Identifier	2	
	Sequence Number	2	
	Originate Timestamp	4	
	Receive Timestamp	4	
	Transmit Timestamp	4	
Packet Too Big	ICMP Header	Not in ICMP	4
	MTU		4
	Original packet data		N

Table 10: ICMP/ICMPv6 Messages

or more options may be included at the end of a message, and some options may appear multiple times. NDP requires these options to be padded to end on 64-bit boundaries.

The following options are part of NDP:

- Source/Target Link-layer Address
- Prefix Information
- Redirected Header
- MTU

These options can be seen in table 11. As can be seen in the table, different options can be present in different messages with the Source/Target

Link-layer Address option being possible in all messages. The Prefix Information Option contains two flags, L and A. These flags are the on-link flag, L, and the autonomous address-configuration flag, A.

Option	Field	Size (Bytes)	Used in
Source/Target Link-layer Address	Type	1	All messages
	Length	1	
	Link-layer Address	N	
Prefix Information	Type	1	Router Advertisement
	Length	1	
	Prefix Length	1	
	L/A/Reserved	1	
	Valid Lifetime	4	
	Preferred Lifetime	4	
	Reserved	4	
	Prefix	16	
Redirected Header	Type	1	Redirect
	Length	1	
	Reserved	6	
	IP header + data	N	
MTU	Type	1	Router Advertisement
	Length	1	
	Reserved	2	
	MTU	4	

Table 11: NDP Options

The following messages are part of NDP:

- Router Solicitation
- Router Advertisement
- Neighbor Solicitation
- Neighbor Advertisement
- Redirect

These messages can be seen in table 12.

The Router Advertisement and Neighbor Advertisement messages both have flags. The Router Advertisement M and O flags are used to indicate that addresses (M) or other configuration information (O) is available via

Dynamic Host Configuration Protocol for IPv6 (DHCPv6), which is outside of the scope of this thesis. The Neighbor Advertisement R, S and O flags indicate that the neighbor is a router (R), whether the advertisement was a response to a solicitation message (S), and whether the advertisement should override an existing cache entry (O).

Message	**Field**	**Size (Bytes)**
Router Solicitation	ICMP Header	4
	Reserved	4
	Options	N
Router Advertisement	ICMP Header	4
	Cur Hop Limit	1
	M/O/Reserved	1
	Router Lifetime	2
	Reachable Time	4
	Retrans Time	4
	Options	N
Neighbor Solicitation	ICMP Header	4
	Reserved	4
	Target Address	16
	Options	N
Neighbor Advertisement	ICMP Header	4
	R/S/O/Reserved	1
	Reserved	3
	Target Address	16
	Options	N
Redirect	ICMP Header	4
	Reserved	4
	Target Address	16
	Destination Address	16
	Options	N

Table 12: NDP Messages

4.8 IPv4 and IPv6 pseudo-headers for TCP and UDP checksum calculation

TCP and UDP Checksums cover not only the fields of their headers, but also specific fields of the lower layer protocol and the entire higher layer protocol, or payload, they encapsulate. Fields of the lower layer protocol

are prepended to the UDP or TCP header when calculating the Checksum, and are referred to as pseudo-headers. Since pseudo-headers cross the boundary between layers, they are examined separately in this section. For the purposes of this thesis, only the IPv4 and IPv6 pseudo-headers are examined as these are the only two pseudo-headers supported by Shanon, the anonymization tool developed as part of this thesis.

TCP and UDP Checksums are calculated in a similar fashion to IPv4 Checksums, however in addition to the header itself, the Checksum calculation also includes a prepended pseudo-header, and the payload or encapsulated higher layer protocol. The pseudo-header for IPv4 is defined in RFC 768 [77] and can be seen in Table 13. The pseudo-header for IPv6 is defined in RFC 8200 [31] and can be seen in Table 14. The two pseudo-headers include the same data, but they differ in the order of fields and their sizes. The fields included in the pseudo-header are relevant for the correct delivery and integrity of the data delivered: the source and destination addresses, the protocol that is carried - the next header in the case of IPv6, the length of the upper layer protocol payload, and a padding of zeroes meant to ensure the pseudo-header length is a multiple of 16 bits, or two bytes. This is the size of the words used when calculating the checksum, as well as the size of the checksum fields in TCP and UDP.

Both the Protocol and Next Header fields used in the pseudo-headers use the same protocol numbers as IPv4 and IPv6, the assignments of which are maintained by IANA at [51].

Field	Size (Bytes)
Source Address	4
Destination Address	4
Zero	1
Protocol	1
TCP/UDP Length	2

Table 13: IPv4 Pseudo-header for TCP and UDP

4.9 UDP

The User Datagram Protocol (UDP)[77] header, seen in Table 15 is composed of 4 2-byte fields.

The Source and Destination ports serve to identify the process sending the datagram and the intended receiving process. Port numbers are divided

Field	Size (Bytes)
Source Address	16
Destination Address	16
Upper Layer Protocol Length	4
Zero	3
Next Header	1

Table 14: IPv6 Pseudo-header for TCP and UDP

into three groups. Well Known Ports are ports between 0 to 1023, Registered Ports range from 1024 to 49151 and Ephemeral Ports range from 49152 to 65535. Well Known Ports and Registered Ports are assigned specific uses by IANA. Ephemeral Ports are unassigned and can be used by various processes [26].

The Length field is the total length of the datagram, including header and payload.

The Checksum is calculated in a similar fashion to the IPv4 header checksum, but the UDP checksum covers the lower layer protocol, the UDP header, and the payload. This is done using a pseudo-header. Pseudo-headers are described in Section 4.8

Protocol	Field	Size (Bytes)
UDP	Source port	2
	Destination port	2
	Length	2
	Checksum	2

Table 15: UDP

4.10 TCP

The Transmission Control Protocol (TCP)[79] header can be seen in Table 17. It is composed of numerical fields and single-bit flags. Additional, optional fields may be present at the end of the header. These options are discussed later in this section.

The Source and Destination port serve the same purpose as in UDP and are assigned in the same fashion.

The Sequence Number field can have two meanings. When the Synchronization (SYN) flag is set, the Sequence Number is the Initial Sequence Number (ISN) and the first byte of data is ISN+1. When the SYN flag is not

set, the Sequence Number represents the order of the first byte of the TCP segment in a sequence of segments.

When the Acknowledgment field significant (ACK) flag is set, the Acknowledgment Number contains the value of the next Sequence Number that the sender of the segment is expecting, and is always sent once a connection is established.

The Data Offset field is similar in purpose to the IPv4 IHL field. It is the length of the TCP header in 32-bit words. When there are no Options included at the end of a TCP header this field contains the value 5 Decimal.

The Reserved field is reserved for future use and should be set to 0.

The Flags, also called Control Bits, are a set of 1-bit fields. The Nonce Sum (NS) flag is an experimental nonce for ECN [89]. The Congestion Window Reduced (CWR) and ECN-Echo (ECE) flags serve to communicate congestion. They also serve a secondary role which is communicating ECN capability during the TCP three-way handshake which can be seen in Figure 2 [80]. The Urgent Pointer field significant (URG) flag signals that the Urgent Pointer field is significant, signalling that bytes of the payload up to the byte marked by the Urgent Pointer are urgent. The ACK flag signals that the Acknowledgement number field contains a value of the next Sequence Number the sender of the segment is expecting. The Push Function (PSH) flag is used to signal the receiver should not wait for more data before sending the received data to the receiving process. The Reset the connection (RST) flag is used to reset the connection. RFC 793 states that: “As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection.” The SYN flag is used to synchronize Sequence numbers when initiating a connection, as can be seen in the three-way handshake in figure 2 and should otherwise not be set. The No more data from sender (FIN) flag is used to terminate a connection, as can be seen in the four-way handshake in figure 3

The Window Size field is the number of bytes the sender of the TCP segment is willing to accept, beginning with the byte indicated in the Acknowledgment Number

The Checksum field is a checksum calculated using the same method as the IPv4 and UDP checksum. Like the UDP checksum, it makes use of a pseudo-header to also cover fields of the lower layer protocol. Pseudo-headers are described in Section 4.8.

The Urgent Pointer is used in combination with the URG flag and is used to indicate which byte, relative to the received segment’s sequence number, is the last byte of urgent data.

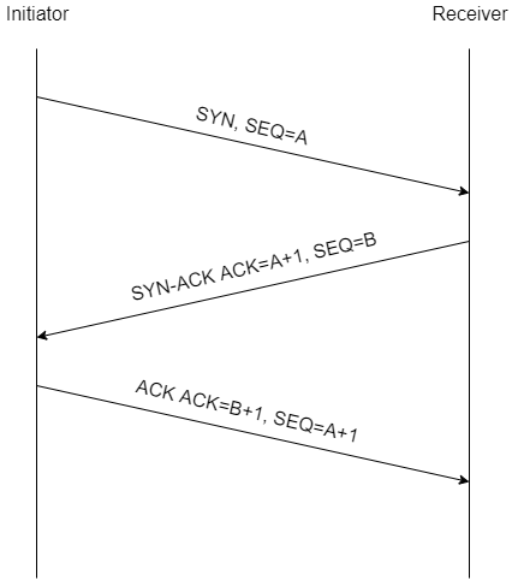

Figure 2: TCP three-way handshake

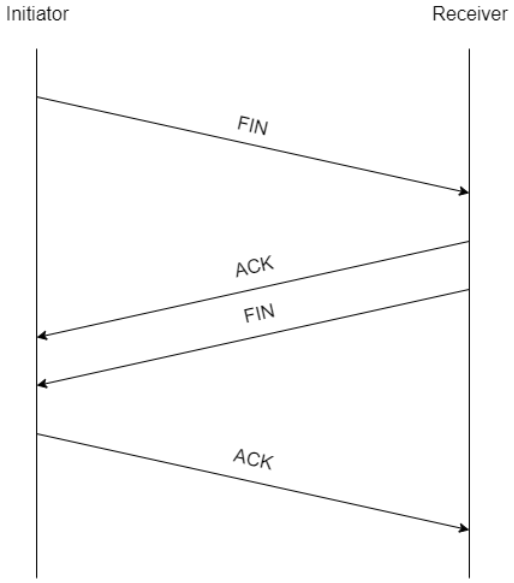

Figure 3: TCP four-way handshake

TCP Options

TCP supports up to 40 Bytes of options. :

- End of Options
- No Operation (padding)
- Maximum Segment Size
- Window Scale
- Timestamp
- Selective Acknowledgment

TCP Options can take on 2 possible forms. Either a single byte determining the kind of option with no additional data present, or a kind/length/data triplet with 1 byte determining the type of option, 1 byte determining the total length of the option (including kind, length and data fields) and a variable amount of data. The fields and values present in the TCP options listed above can be seen in table 16.

The End of Options, No Operation and Maximum Segment Size options are defined in RFC 793 [79] . End of Options is used to mark the end of the options list. No Operation can be used as padding to align options to 32-bit boundaries. If the Maximum Segment Size option is present in a TCP segment it indicates the maximum size of a TCP segment the sender is willing to receive. Maximum Segment Size must only be sent with a TCP segment that has the SYN flag set.

The Window Scale and Timestamp options are defined in RFC 7323 [5]. The Window Scale option expands the Window Size field by communicating the number of left shifts to be performed on the Window Size to determine the actual maximum size. This expands the TCP segment size from 65 kibibytes to 1 gibibytes. When using the Window Scale option TCP determines if a segment is a previously received or new segment by testing if the sequence number is within 2^{31} bytes of the left edge of the receive window. This limits the maximum size of a window to below 2^{30} bytes. Since the maximum unscaled window size is $2^{16} - 1$ this limits the magnitude of the shift to a maximum of 14 places. The Window Scale option may be present in a TCP segment with the SYN flag set to indicate the sender of the segment is willing to use Window Scaling and communicate the scaling factor to be applied to its Window Size, as well as in a segment with the SYN and ACK flags set (if the option was present in an initial SYN segment) to agree

Option	Field	Length	Value
End of Options	Kind	1	0
No Operation	Kind	1	1
Maximum Segment Size	Kind	1	2
	Length	1	4
	Maximum Segment Size	2	N
Window Scale	Kind	1	3
	Length	1	3
	Shift Count	1	N
Timestamp	Kind	1	8
	Length	1	10
	TS Value	4	N
	TS Echo Reply	4	N
Selective Acknowledgment Permitted	Kind	1	4
	Length	1	2
Selective Acknowledgment	Kind	1	5
	Length	1	N
	Left Edge of Nth Block	4	N
	Right Edge of Nth Block	4	N

Table 16: TCP Options

to use it. The Timestamp option's carries 2 fields, Timestamp Value and Timestamp Echo Reply. The Timestamp Value field contains the timestamp of the sender. The Timestamp Echo Reply is valid when the ACK bit is set and should otherwise be set to zero. The process of deciding which received timestamp to echo takes into account delayed acknowledgements, lost segments and segments filling holes left by lost segments in order to arrive at a more precise round trip time estimate. For the purposes of anonymization this exact mechanism appears to be irrelevant as reducing the precision of timestamps already interferes with its stated goals, and it is therefore not further discussed in this work. The Timestamp option may be sent with a segment with a SYN flag set to indicate willingness to use this option and in a segment with the SYN and ACK flags set (if received in a segment with a SYN flag set) to agree to use it. Once successfully negotiated (both the initial SYN segment and the SYN-ACK response contained the option) the Timestamp option must be sent in every segment that does not have the RST flag set for the negotiated connection, and segments received without it should be silently dropped without aborting the connection.

TCP Selective Acknowledgement is defined in RFC 2018 and consists of 2 options. The first option is the Selective Acknowledgment Permitted

option. The purpose of this option is to indicate support for Selective Acknowledgements, and it may be sent with a segment with the SYN flag set, but must not be sent otherwise. The choice of using 2 bytes rather than making it a 1-byte kind-only option does not seem to serve any purpose and would appear to be a design oversight. The second option is the Selective Acknowledgement option. It is composed of pairs of 32-bit integers in network byte order which mark the left and right edges of continuous blocks of received data. Thus data in TCP's own Acknowledgement field signifies the last byte of received continuous data in order, with the attached Selective Acknowledgement option identifying blocks that have also been received, but with data missing between the left edge of the 1st block and the right edge of the last received in-order block of data, as well as between the right and left edges of the blocks in the Selective Acknowledgment option.

The TCP header ends with zero padding for the purpose of aligning the beginning of data with a 32-bit boundary. This is necessary as the data offset field expresses the beginning of the data in 32-bit words.

Protocol	Field	Size (Bytes)	Subfields	
			name	size(bits)
TCP	Source port	2		
	Destination port	2		
	Sequence number	4		
	Acknowledgment number	4		
	Data offset, Reserved, Flags	2	Data offset	4
			Reserved	3
			NS	1
			CWR	1
			ECE	1
			URG	1
			ACK	1
			PSH	1
			RST	1
			SYN	1
			FIN	1
	Window Size	2		
	Checksum	2		
	Urgent pointer	2		
	Options	N		

Table 17: TCP

5 Anonymization of chosen protocols

In this chapter the importance of anonymizing the different types of fields encountered in the chosen protocols will be discussed.

Despite sharing a common function at a high level, the semantics of the different types of fields discussed here can differ significantly between protocols. Due to their commonalities it makes sense to group them for the sake of discussing anonymization, however when anonymizing the fields it is necessary to pay attention to the differences between them.

An example of these differences between functionally similar protocol fields can be seen in Section 5.1 below. Despite both IP and MAC addresses being addressing information, the addresses are semantically different and cannot be anonymized in the same fashion.

Similar situations occur with different fields in all the protocols discussed in this thesis. For this reason, while the general ideas in this chapter apply to the types of fields explored here, it is necessary to consider each field individually in the context of its operation in the protocol it is a part of when anonymizing that field.

5.1 Addresses

The importance of anonymizing addresses is high and a lot of work in anonymization has been dedicated just to methods of anonymizing IPv4 addresses and demonstrating these methods are sound and offer high levels of protection, as well as countering these methods. It therefore seems sensible to also dedicate effort to anonymizing addresses in this thesis as well. The addresses present in the protocols chosen for this thesis are MAC, IPv4 and IPv6 addresses. At the most basic level, addresses identify devices for the purpose of communicating with them in a network. However, these addresses can have different meanings and scopes.

MAC addresses

A MAC address is used to communicate on a shared medium, such as a physical cable or wireless network, or in a segment of a network where multiple devices are connected via a switch.

The first two bits of a MAC address serve specific purposes. The first bit determines whether an address is the address of an individual device(0), or a group address(1). The second bit determines whether an address is globally administered(0), or assigned locally(1). Additionally, the broadcast

address, an address used to send broadcast frames to all devices on a shared medium, is an address where all bits are set to 1. For globally administered addresses prefixes are assigned from an address space managed by the Institute of Electrical and Electronics Engineers (IEEE). According to the IEEE 802 Standard three different types of MAC address spaces are currently able to be registered. The MAC Addresses - Large (MA-L) address space has 24 IEEE assigned bits (out of 48) that can be used as a company or organization identifier and are known under the name Organizationally Unique Identifier (OUI). Many large network equipment manufacturers will be using this address space and their network equipment will thus match the commonly mentioned division of MAC addresses. The MAC Addresses - Medium (MA-M) address space begins with 28 IEEE assigned bits which cannot be used as a company or organization identifier. The MAC Addresses - Small (MA-S) address space begins with 36 IEEE assigned bits which can be used to identify a company or organization [53].

IPv4 and IPv6 addresses

IPv4 and IPv6 addresses are used for communicating within and between networks and can be used to access a device across large distances.

The IPv4 address space is managed by IANA. It is divided into blocks which are either reserved for a specific purpose or assigned to a local registry which can then assign it to individual organizations such as service providers.

The IPv4 address spaces that follow are assigned specific purposes by IANA. The addresses will be written in Classless Inter-Domain Routing (CIDR) notation, meaning an IPv4 address will be written in the form of four numbers representing the four bytes of the address, separated by dots. Following the IPv4 address will be a slash and a number which represents the number of bits, starting from the left side of the IPv4 address, which represents the network portion of the address.

The 0.0.0.0/8 address space is reserved for self-identification [6].

The 127.0.0.0/8 address space is reserved for Loopback and must not appear outside a host [6].

The 100.64.0.0/10 address space is reserved for the Shared Address Space. These addresses are used by service providers in Carrier-Grade Network Address Translation (CGN) deployments, a measure meant to extend the life of IPv4 until IPv6 is fully deployed [99].

The 169.254.0.0/16 address space is reserved for Link-Local usage, meaning communication on the same physical or logical link [14].

The 192.168.0.0/16, 172.16.0.0/12, and 10.0.0.0/8 address spaces are reserved for private use, meaning these addresses can be used by private individuals or organizations within private networks without a need to register them with a registry. These addresses are not intended to be routed on the public internet [81].

The 192.0.2.0/24, 198.51.100.0/24, and 203.0.113.0/24 address spaces are referred to as TEST-NET-1, TEST-NET-2, and TEST-NET-3, and are intended for use in documentation and examples in order to avoid conflicts and confusion. Addresses within these blocks should not appear on the public Internet [2].

The 198.18.0.0/15 address space is reserved for benchmarking network devices [7].

The 192.88.99.0/24 address space is reserved for the deprecated 6to4 anycast defined in RFC 3068 [47], a mechanism for communication between IPv6 networks over IPv4 networks. It was deprecated in by RFC 7526 [97].

Address blocks in the 224.0.0.0/8-239.0.0.0/8 range are reserved for different kinds of Multicast traffic [25].

Address blocks in the 240.0.0.0/8-255.0.0.0/8 blocks are reserved for future uses, with the exception of 255.255.255.255/32 [33]. 255.255.255.255/32 is used to broadcast to all immediate neighbours [64].

The IPv6 address space, like the IPv4 address space, is managed by IANA and divided into blocks of addresses with specific purposes or assigned to registries.

IPv6 CIDR notation differs slightly from IPv4 notation. Blocks of up to four hexadecimal characters representing two bytes each, are separated by a colon. One group of two colons may be present in the notation, indicating an uninterrupted sequence of zeroes. Like with IPv4, a slash followed by a number at the end of the sequence indicates the number of bits, starting from the left side of the IPv6 address, which represents the network portion of the address.

The IPv6 address spaces that follow are assigned specific purposes by IANA. This list of address spaces was compiled by Reseaux IP Europeens (RIPE) and is available at [67].

The ::/128 address space can only be used as by a host that is initializing before it learns its own address. This is similar to the 0.0.0.0/8 address space in IPv4.

The ::1/128 IPv6 address is used for loopback. This is similar to the function of the 127.0.0.0/8 address space in IPv4.

The ::ffff:0:0/96 address space is used to map IPv4 addresses to IPv6 addresses. The last four bytes of the IPv6 address can be written using the

same CIDR notation as the IPv4 address that is being mapped, for example ::ffff:192.0.2.47.

The fc00::/7 address space is reserved for Unique Local Addresses. These addresses serve the same purpose as IPv4 private addresses.

The fe80::/10 address space is reserved for Link-Local addresses. These addresses serve the same purpose as IPv4 Link-Local addresses.

The 2001:0000::/32 address space is used for Teredo, a mechanism that enables hosts located behind IPv4 Network Address Translations (NATs) to connect to the internet using IPv6 via a Teredo sevice [48]. The 2001:0002::/48 address space is reserved for benchmarking, similar to the 198.18.0.0/15 IPv4 address space.

The 2001:20::/28 address space is currently being used for an experiment named ORCHID2 [59].

The 2002::/16 address space is used for 6to4 unicast which is defined in RFC 3056 [13]. Unlike the IPv4 6to4 anycast transition mechanism, the unicast 6to4 mechanism is not deprecated.

The 2001:db8::/32 address space is reserved for documentation, similar to the TEST-NET-1, TEST-NET-2, and TEST-NET-3 IPv4 address spaces.

The ff00::/8 address space is used for multicast.

Anonymization considerations

When capturing network traffic in a local network, devices will have both unique IP addresses and unique MAC addresses. When communicating between networks, the MAC addresses of the communicating devices are typically not transmitted to the distant host. When recording network traffic on a transit link, between networks, captured packets will have the MAC addresses of the communicating routers, and many different IP addresses.

Both MAC and IPv4/IPv6 addresses can be used to infer information about the structure of a network. In addition to this, MAC addresses can potentially be used to uniquely identify a device, as well as to identify the device manufacturer.

IPv4 and IPv6 addresses can also potentially be used to uniquely identify a particular device. Servers with static addresses can be expected to have these addresses for long periods of time. In cases where the IPv4 or IPv6 address of a device is dynamically assigned a history of addresses can be used to identify a specific device. Additionally, IPv4 and IPv6 address spaces are assigned by and to various organizations, meaning these addresses can be used to narrow down the geographic location of a device, the organization it belongs to or the network service provider it is connected

to.

Parts of the IPv4 and IPv6 address space serve specific purposes, and do not identify a particular host within a local network or on the internet. For example, local addresses are used in many networks. On the other hand, the presence of these addresses can also reveal the presence of certain services in the network, such as the use of CGN or Teredo.

Given this information, it is evident that both MAC addresses and IPv4/IPv6 addresses need to be suitably anonymized to prevent revealing unwanted information about devices or the structure of a network. In addition to the protocols themselves, the addresses mentioned here can also be present in other protocols such as ARP and NDP.

5.2 Length

The protocols chosen for anonymization in this work and their options have various length fields. Some of these fields are of no importance to anonymization in and of themselves. For example the various options in NDP all have a length field, but the length is defined by the message type or the Link-layer in use and as such anonymizing them serves no purpose. Likewise the Hardware and Protocol Address Length in ARP will be known given the limited protocol choices of the tool. The same can be said for the length in cases where messages of known prescribed lengths are in use. However, the length of a protocol payload or a pattern of payload lengths can be used to infer the data that was carried or the upper layer protocols in use.[55] For this reason in cases where the upper layer protocol needs to be concealed or a risk exists of exposing data that has been accessed it is prudent to consider recalculating lengths and anonymizing upper layer payloads appropriately to conceal their true length.

In the case of Ethernet, the Type/Length field can serve two purposes, being either the length of the higher-layer payload, or a type field indicating the higher-layer protocol in use, depending on the value of the field. When anonymizing Ethernet, this duality needs to be taken into account in order to properly anonymize the field. The values this field can take on and their meanings are described in Section 4.2.

5.3 Type fields

In situations where concealing the activity of certain protocols is desirable, anonymizing type fields may be necessary. This only makes sense if the upper layer data is anonymized completely or removed, as it would otherwise

be simple to discover which protocol was in use. Patterns of communication, such as a TCP handshake, could also be used to infer the protocols in use.

5.4 Checksums

In an attack on anonymization checksums can be used to verify if deanonymization was successful. For this reason checksums should always be anonymized. In situations where the validity of the checksum is important, they can be recalculated and, if an incorrect checksum is required to mark packets that had incorrect checksums in the original trace, the calculated checksum can simply be incremented.

5.5 Flags

In some cases, such as the Urgent Pointer in TCP, a flag may be set or unset without necessarily breaking the operation of the protocol. Such flags could be used to introduce patterns into a capture that could aid in deanonymization. Other flags, such as the TCP SYN, ACK, FIN and RST flags, are an important part of the workings of the protocol and changing them would interfere with the operation of the protocol or subsequent interpretation of captured packets. As such, it is important to consider the value of a particular flag and whether or not to make sure the flag is cleared to avoid potential patterns.

5.6 Port numbers

TCP and UDP Port numbers are divided into three groups. For the purposes of anonymization Well Known Ports (ports ranging between 0 and 1023) and Registered Ports (between 1024 and 49151) are of interest. These ports are assigned to specific uses and as such can reveal the presence of certain types of services on a host even if upper layer payloads are anonymized and their types are removed. As such it may be necessary to anonymize port numbers in order to prevent matching a host in an anonymized capture with the actual real host based on knowledge about the organization sharing the capture.

5.7 Reserved fields

Reserved fields are fields left for future updates of protocols. Reserved fields should be set to 0 and ignored. These fields could, however, be used

to pass on secret messages or introduce patterns into an anonymized trace that could later be used to deanonymize the trace.

5.8 Payload or Data field

Network protocols operate in a layered architecture where each layer is responsible for providing layer-specific services to the layer above it. The data or payload field of a protocol instance can therefore contain an instance of a higher-layer protocol or user data, depending on the situation.

For protocols which have a type field, the type of the higher-layer protocol is specified in this field. In the case of TCP and UDP port numbers can be used providing a Well Known Port number or Registered Port number is in use, as these are linked to specific applications and use cases. These values should not be blindly trusted, however, as it is possible to misuse them.

More details on the layered network architecture are provided in Section 4.1.

5.9 Other fields of interest

In addition to the fields listed above several other fields that can potentially be abused or reveal information exist in the protocols supported by this work. In IPv4 the Identification field is intended to be used only to identify fragments of a same packet, but there does not seem to be anything preventing the field from being used to introduce patterns into a packet trace. The TTL field can potentially be used to identify the OS in use on a host since different OSes use different initial values [84]. The same applies to the IPv6 Hop Limit field which serves the same purpose as the IPv4 TTL field. Timestamps present in ICMP Timestamp messages, TCP Timestamp options, and in the pcapng file format could reveal the time a capture was taken which may help matching with known patterns that were injected or happened naturally at the time of the capture, helping deanonymization.

When anonymizing timestamps it is also important to consider the resolution of timestamps, which may differ between different timestamps, in order to achieve a consistent level of anonymization. IPv6 Hop-by-hop Options and Destination Options can contain multiple TLV encoded options which can carry different kinds of data. These fields are ripe for abuse and need to be carefully examined. In addition to their contents, the order of optional headers, such as TCP Options, may also help reveal information about the source device, such as the OS it is running [87]. It is conceivable

that the same could apply to IPv6 Extension Headers and other optional headers.

5.10 Future considerations

In addition to all the information above, new ways of abusing known fields may be discovered in the future, rendering some of the anonymization methods and considerations discussed in this work obsolete. Methods may already exist that are of yet not publicly known or have not been sufficiently discussed. As such these considerations should not be taken at face value, but used to inform decisions made when anonymizing a packet trace in the context of the environment the trace comes from, the security requirements of the organization providing the trace and the requirements of the research using the trace in order to find an acceptable trade between a reasonable level of security and research value.

6 Wireshark as a basis of a new anonymization tool

Wireshark [40] is an open source protocol analyzer with support for a wide variety of protocols across all layers. It supports both online analysis through packet capture as well as offline analysis of packets captures from a wide variety of file formats including libpcap's pcap file format and Wireshark's own pcapng format. Wireshark is available natively on many platforms, including Windows, Linux and MacOS. The remainder of this chapter discusses the features of Wireshark based on the features outlined in Chapter 3, the pcapng file format used by Wireshark, and the way in which the Lua scripting language is used in Wireshark to retrieve protocol information.

6.1 Features of Wireshark

Wireshark is a good choice as a basis for a new anonymization tool due to its support for the many desirable features outlined in Chapter 3. Large files are supported both by libpcap's pcap file format as well as Wireshark's own pcapng file format. Consistency can in part be achieved through the pcapng file format's support for concatenating multiple captures, however support for consistent online anonymization and consistent anonymization at multiple locations would have to come from the tool itself as well as be possible for the particular policy developed by a prospective user. Wireshark's capability to analyze both online and packets captured from files allows for the development of an anonymization tool supporting online as well as offline anonymization. Wireshark itself is extensible through the writing of protocol dissectors and taps. Dissectors enable the dissection (analysis) of new protocols through the development of new dissectors. Taps allow tapping into (accessing programmatically) the information made available by these dissectors. This extensibility and support for a wide variety of protocols can also be utilized by an anonymization tool by allowing users to write their own anonymizers for protocols of their choosing. However, some protocol features may require more than just an existing dissector and filter (e.g. generating checksums post-anonymization). Performance does not currently seem to be a strong feature in Wireshark, as it is only single-threaded at the time and may perform slowly with large captures.[23]

6.2 The pcapng file format

The pcapng[61] file format is a file format for storing network packet traces used by Wireshark. It was developed to address the lack of functionality of libpcap's pcap file format. The file format consists of a set of blocks defining various aspects of the capture. These blocks are:

- **Section Header Block (SHB):** Section Header Blocks define the beginning of a section in a pcapng file, which can contain multiple sections. Since a pcapng file must always begin with a Section Header Block (SHB) it is possible to merge multiple pcapng files into a valid pcapng file by simply concatenating them.
- **Interface Description Block (IDB):** An Interface Description Block is used to store information related to an interface that was used to capture packets. Interface Description Blocks (IDBs) can be used to store additional information about a capture device such as the interface's name, description, addresses and other. Other blocks within a section can reference these IDBs.
- **Interface Statistics Block (ISB):** An Interface Statistics Block is used to store capture statistics for a particular interface which is specified by an interface Identifier (ID) in the block. Statistics are recorded as options added to the block. These can be the capture start time, end time, received and dropped packets, and others.
- **Enhanced Packet Block (EPB):** An Enhanced Packet Block stores a captured packet as well as additional information regarding this packet such as a 64-bit timestamp, the original length and the actually captured length, and an interface ID that references an existing IDB in the same section.
- **Simple Packet Block (SPB):** A Simple Packet Block is a simpler block for storing packets. It only stores the original length and the captured portion of the packet, omitting the extra information present in an Enhanced Packet Block (EPB).
- **Name Resolution Block (NRB):** A Name Resolution Block is used to store pairs of IPv4 and IPv6 addresses and their records as retrieved from a Domain Name System (DNS) server.
- **Custom Block (CB):** A Custom Block can be used to store custom data.

Each block may also have additional options. These are optional fields that add additional information to the particular block. Many of these options are block-dependant, and thus will not be listed here. An end-of-options option marks the end of options for each block type. A comment option containing UTF-8 text as well as a custom option the contents of which are up to the implementer can be added to any block.

In addition to the blocks mentioned above several experimental blocks exist. Due to being experimental, they are not mentioned in this thesis.

Wireshark's pcapng file format could be used to attach meta-data directly to captures. Custom blocks and options can be used by an application to attach meta-data to particular packets, sections marking groups of multiple captured packets or to the entire file. This feature can also be used to support recoverability by marking particular packets or sections of a file (or entire files) as having experienced a certain error in a way the application can parse and possibly correct. The configuration used to anonymize the file could also be stored in the file in a machine-readable way that would allow the application to execute additional anonymization operations while leaving already sufficiently anonymized parts untouched. It is important to note that a Private Enterprise Number (PEN) is needed in order to generate a valid Custom Block. This is to avoid number space collisions between organizations looking to implement their own custom blocks in the file format. A PEN can be freely registered with IANA by filling out a form.[50]

The tool developed as part of this thesis makes use of the SHB, IDB and EPB, as well as the comment, end of options and interface timestamp resolution options. A more detailed look at the structures of these three blocks and the attached options is provided in Section 7.2 where the functions for writing a pcapng filesystem are discussed.

6.3 Lua in Wireshark

Lua, meaning "Moon" in Portuguese, is a free, open-source scripting language developed and maintained by the Pontificial Catholic University of Rio de Janeiro in Brazil [34]. A more in-depth analysis of Lua itself and the many functions available in Wireshark is out of scope for this thesis. Instead, this section will focus on functions used in this thesis.

In Wireshark Lua can be used to write dissectors, post-dissectors and taps. [21]

Dissectors are used to analyze a part of the data of a captured packet. They can be written in Lua, which can help prototype dissectors, but all Wireshark dissectors are written in C for performance reasons.

Post-dissectors are similar to dissectors, but run after all other dissectors have run.

Taps are used to retrieve information after a protocol has been analyzed by Wireshark's dissectors. This means taps have read access to all fields of the protocol and Wireshark's interpretation of these fields, as well as any additional data Wireshark generates about them, such as TCP state information or the validity of checksums. For the purposes of this thesis Wireshark taps were used so the main focus of this section will be taps.

Taps

The official Wireshark documentation is too short and unclear in many cases. The description of taps is one such case. For example, in the wiki article for Lua taps [22] taps are described as a mechanism to fetch data from every frame, however an example script using this mechanism is also referred to as a tap. In the Wireshark documentation a function that is called for every packet matching a certain filter or having a certain tap is referred to as a listener [20]. In this context the word tap seems to refer to an internal mechanism by which data from dissectors and post-dissectors is made available. In the following text the function that is called when a particular protocol has been dissected will be referred to as a listener.

To create a listener, the function Listener.new is used. The function can accept three optional parameters. The first parameter is referred to as the tap. The exact meaning of tap is, again, unclear in this context, however listing available listener names using the Listener.list function reveals that these tap names correspond to the names of protocols as used when using filters in Wireshark. The second parameter is the filter. This would seem to be any filter which could be applied in Wireshark. The third parameter is named allfields and is described as a boolean determining if all fields should be geneated. The exact meaning of this is not demonstrated or further clarified in the documentation. The result of calling Listener.new is a table containing a set of callback functions which Wireshark itself will execute. The function of interest for this thesis is the listener.packet function which is called for every packet Wireshark processes that matches the provided tap and filter. When called by Wireshark, the listener.packet function is given three parameters. A Packet Information (pinfo) object containing information about the captured packet, a TVB object representing the buffer containing the bytes of the captured packet, and a Tap Information (tapinfo) table that contains information based on the type of listener, or nil. The documentation currently lacks further information about the tapinfo table [20]. .

Retrieving protocol data

The various fields of the protocols analyzed by Wireshark can be retrieved using a field extractor. Field extractors are creates using the Field.new function with a field name as a parameter. Field names correspond to the filters that can be used in Wireshark. For example “ip.src” would correspond to the source address of an IPv4 packet. Field extractors need to be created outside the callback functions previously mentioned in this section. Once created, a field extractor can be called within a callback function and will return an array of fieldinfo objects. The objects in this array will be ordered by order of appearance in the analyzed portion of the captured frame. This behaviour is not explained in the documentation, and the call is treated as retrieving a single value of a field in several examples, which can be confusing. If an array of values is not expected and the returned value is stored in a variable instead of a table, only the first value will be retrieved. The lack of documentation for this scenario can cause issues in situations where multiple instances of the same protocol are expected to be in a captured packet, such as can be the case with IP in IP encapsulation or ICMP and ICMPv6 containing a portion of the packet that caused the ICMP or ICMPv6 message.

The fieldinfo objects created by calling a field extractor inside a callback function contain several functions and properties for retrieving information about a field, however they lack a method to get the binary data of a field. For most intended purposes this is not an issue, however for our use case we require the binary data. To do this the offset and length of the field can be retrieved from the fieldinfo object, then a tvbrange object containing the bytes in this part of the buffer can be retrieved from the TVB buffer using the tvb:range function with the offset and length of the field as parameters. One important thing to note about this is that the retrieved offset of a protocol field will correspond to the first byte of the buffer containing the first bit of this field. This is important when retrieving the bytes of fields which do not begin or end on byte boundaries, such as the version field in IPv4 and IPv6 headers which begins on a byte boundary, but is only four bits long. Likewise, the length of a field is expressed in bytes and rounded up. As a result the aforementioned IPv4 version field begins at the first byte of the IPv4 header and has a length of one byte. This means that an IPv4 version field retrieved from Wireshark’s buffer in this fashion contains both the version field and the IHL field. The values retrieved from the buffer in this fashion are encoded in the form of hexadecimal characters by default. In order to retrieve the actual bytes the tvbrange:raw function needs to be called, which will return the actual (raw) bytes contained in the retrieved part of the TVB buffer in the

form of a string of bytes. For this reason bytes retrieved from the TVB buffer in this way may be referred to as raw bytes in parts of this thesis [19, 18]

7 libAnonLua

libAnonLua is a Lua library written in C which contains constants, functions for writing a pcapng files, anonymization functions, functions for recalculating correct checksums, and helper functions. It is the core functional part of Shanon. Its place in the tool can be seen in Figure 5 in Chapter 8. libAnonLua source code can be found online at [29].

In this chapter an overview of the constants and functions provided by libAnonLua is provided.

7.1 Constants

libAnonLua makes several constants available for use in code using the library. These are intended to make using the functions of the library easier. The pcapng file format has several link types that can be present in IDBs and which determine what layer 2 protocol the data in the EPBs or Simple Packet Blocks (SPBs) is treated as. For ease of use these are loaded from a Comma Separated Values (CSV) file named linktypes.csv that needs to be located in the same folder as the library. The libAnonLua source files come with this file included. Loading these values at library initialization makes it possible to add additional linktypes in the future without reworking the whole library. The link types are then made available under easy to remember names prefixed with LINKTYPE_. For example, for Ethernet the link type would be LINKTYPE_ETHERNET. Since the full table of link types is long, it is not included here.

Since the black_marker anonymization function can be applied starting at the Least Significant Bit (LSB) or Most Significant Bit (MSB), effectively either removing a prefix or suffix from an address, two constants for the direction are also present to make using this function more intuitive. These are black_marker_LSB and black_marker_MSB.

Finally, a version constant is made available in order to ensure that the appropriate version of libAnonLua is being used. This serves to prevent situations where an outdated library would cause compatibility issues due to changes in library functions.

7.2 Writing pcapng files

libAnonLua has three functions for writing pcapng files. These functions are:

- create_filesystem(string path)

- `add_interface(string path, LINKTYPE type)`
- `write_packet(string path, string_raw packet_bytes, integer IDB ID, [double timestamp], [string comment])`

The create_filesystem function

The `create_filesystem` function accepts a single parameter: a path, including filename and extension, and creates a pcapng compatible file with that name by writing a minimal SHB to that file. This minimal SHB can be seen in table 18.

In case of success the `create_filesystem` function returns a value of 1, otherwise it returns -1.

Protocol	Field	Size (Bits)	Value
SHB	Block Type	32	0x0A0D0D0A
	Block Length	32	28
	Byte Order Magic	32	0x1A2B3C4D
	Major Version	16	1
	Minor Version	16	0
	Section Length	64	0
	Block Length	32	28

Table 18: A minimal SHB block

The add_interface function

The `add_interface` function accepts two parameters: a path of the same kind as the `create_filesystem` function and a `LINKTYPE` parameter, which is an integer constant determining the type of layer 2 interface the IDB represents. Link types are explained in Section 7.1.

A IDB as written by `libAnonLua` can be seen in table 19. The minimal length of a IDB without any additional options is 20 bytes, however an additional 12 bytes are added to the IDB produced by `libAnonLua` as an option (Interface Timestamp Resolution, `if_tsresol`) is added to each IDB to set the time resolution to nanoseconds instead of the default, microseconds, which are used for compatibility with libpcap pcap files. The timestamp resolution is added for compatibility with both Wireshark and the method used by `libAnonLua`, internally, to get precise system time. Wireshark's Lua Application Programming Interface (API) returns the absolute time a packet was captured at in seconds as a Lua number, which is in a double-precision

floating-point format. Thus a high precision in nanoseconds can be reached. Using the default microsecond time resolution would possibly destroy the precision preserved in this timestamp. Likewise, the method used to get system time internally in `libAnonLua` returns the time in nanoseconds. The exact precision of this time depends on the time resolution of the system clock. Since options need to align to a 32-bit boundary, 24 bits of pad are added following the option, then an endofopt option is added which marks the end of options.

In case of success the `add_interface` function returns the IDB ID, an identifier for the created IDB that is used when writing packets to determine which interface the packets originated from. In case of failure the `add_interface` function returns a value of -1.

Protocol	Field	Size (Bits)	Value
IDB	Block Type	32	0x00000001
	Block Length	32	32
	Link Type	16	`LINKTYPE`
	Reserved	16	0
	SnapLen	32	0
	Option `if_tsresol` Option code	16	0x09
	Option `if_tsresol` Length	16	0x01
	Option `if_tsresol` Value	8	0x09
	Pad	24	0
	Option `opt_endofopt` Option code	16	0
	Option `opt_endofopt` Length	16	0
	Block Length	32	32

Table 19: A IDB as written by LibAnonLua

The write_packet function

The `write_packet` function accepts up to five parameters: the path to the file to write into, same as the `create_filesystem` and `add_interface` functions, a string containing the frame bytes to write, the IDB ID of the interface this frame was captured on, an optional timestamp, in seconds, and an optional comment. The string containing the packet bytes is referred to as a raw string in reference to the Wireshark Lua API where an array of bytes taken from a packet will be a string of hex characters, and the method raw is used to get the actual binary values of these bytes. The IDB ID refers to the ID returned by the add_interface function and marks the generated

EPB as having come from that interface and using the layer 2 protocol of that interface. If the optional timestamp is omitted or a non-number value is used the system time at the time `libAnonLua` writes this packet is used for the time. This results in the order of packets remaining the same as during the capture or in the original capture file as packets are processed one after the other in sequence, but destroys the original times between packets as these now depend solely on how quickly the packets were processed before being written. If the optional comment is included, it is attached to the created EPB as a comment option and will show up as a comment on this packet when viewing the generated capture in Wireshark. Due to the varying possible comment lengths zero padding may or may not be present to align the comment option to a 32-bit boundary. The fields of an EPB as used by `libAnonLua` can be seen in table 20.

Protocol	Field	Size (Bits)	Value
EPB	Block Type	32	0x00000006
	Block Length	32	Variable
	Interface ID	32	1
	Timestamp High	32	Variable
	Timestamp Low	32	Variable
	Captured Packet Length	32	0x09
	Original Packet Length	32	0x01
	Packet data	Variable	Variable
	Option `opt_comment` Option code	16	0x01
	Option `opt_comment` Length	16	Variable
	Option `opt_comment` comment	Variable	Variable
	Pad	Variable	0
	Option `opt_endofopt` Option code	16	0
	Option `opt_endofopt` Length	16	0
	Block Length	32	Variable

Table 20: The fields of an EPB

7.3 Anonymization functions

`libAnonLua` contains the following anonymization functions:

- `black_marker(string_raw bytes, int mask_length, int direction)`
- `apply_mask(string_raw bytes, string_raw mask)`

- `get_port_range(string_raw port_number)`
- `HMAC(string_raw bytes, string salt, int iterations)`
- CryptoPAN functions:
  - `init_cryptoPAN(string filename)`
  - `cryptoPAN_anonymize_ipv4(string_raw IPv4_address)`
  - `cryptoPAN_anonymize_ipv6(string_raw IPv6_address)`

black_marker

The `black_marker` function accepts three parameters: a string of bytes to anonymize, the length of bytes to mask, and the direction in which to anonymize in the form of an integer. For ease of use this direction is available as two constants: `black_marker_LSB` to start anonymizing from the Least Significant Bit (LSB) and `black_marker_MSB` to start from the Most Significant Bit (MSB).

The return value of the `black_marker` function is the anonymized string of bytes.

apply_mask

The `apply_mask` function accepts two parameters: a string of bytes to anonymize, and a string of bytes representing a mask that will be applied to the bytes that are to be anonymized in the same order.

The `apply_mask` function returns a string of bytes with the mask applied to them.

get_port_range

The `get_port_range` function accepts a TCP or UDP port number in the form of a raw string of bytes and returns a value indicative of the range the port comes from.

For well-known ports a value of 0 is returned, for registered ports a value of 1024, and for ephemeral ports a value of 49152.

HMAC

The `HMAC` function uses OpenSSL's `libcrypto` library to calculate a Password Based Key Derivation Function 2 (PBKDF2) HMAC of the provided

bytes using 256-bit Secure Hash Algorithm 2 (SHA-2). This algorithm is typically employed to protect passwords stored in databases, in which case the bytes provided would be the password, the salt would be a randomly generated value used to protect against pre-computed hashes, and the number of iterations determines the number of times the function is executed over the provided password. In the `HMAC` function instead of a password the provided bytes are hashed, and the salt and iterations serve as a key of sorts. For the same salt and number of iterations the `HMAC` function will return the same result consistently for the same input bytes.

On failure the `HMAC` function throws a Lua error which appears in the command line of the script being executed. On success a string of bytes of equal length as the input is provided as a result.

CryptoPAN functions

The CryptoPAN functions consist of three functions:

- `init_cryptoPAN(string filename)`
- `cryptoPAN_anonymize_ipv4(string_raw IPv4_address)`
- `cryptoPAN_anonymize_ipv6(string_raw IPv6_address)`

The `init_cryptoPAN` function is used to generate or read bytes for use by 256-bit Advanced Encryption Standard (AES). 64 bytes are either read from an existing file located at the path specified by the parameter filename, or alternatively generated by reading 64 bytes from `/dev/urandom`. These 64 bytes are used as the Key, IV and pad for AES.

The `cryptoPAN_anonymize_ipv4` function is a modification of the original cryptoPAN algorithm written by Jinliang Fan et. al. A direct source for the original CryptoPAN could not be found at the time of writing, but the PktAnon framework uses the original CryptoPAN and had it as part of its source code. This was retrieved from [83].

The original code was modified to use OpenSSL's libcrypto library instead of a version of AES that was originally included with CryptoPAN.

the `cryptoPAN_anonymize_ipv6` function is a modification of the IPv4 version of CryptoPAN for longer IPv6 addresses.

All three functions return a status (-1 on failure, 1 on success) which can be used to determine if the operation failed and take appropriate action. In addition to this status, `cryptoPAN_anonymize_ipv4` and `cryptoPAN_anonymize_ipv6` return the anonymized IPv4 or IPv6 address on success, or an empty string on failure.

7.4 Recalculating checksums

libAnonLua supports calculating checksums for Ethernet, TCP, UDP, IPv4, ICMP and ICMPv6 using the following functions:

- calculate_eth_fcs(string_raw frame)
- calculate_ipv4_checksum(string_raw header)
- calculate_tcp_udp_checksum(string_raw IP_packet)
- calculate_icmp_checksum(string_raw ICMP_packet)
- calculate_icmpv6_checksum(string_raw IPv6_packet)

calculate_eth_fcs

The calculate_eth_fcs function calculates the FCS of an Ethernet frame. The FCS of an Ethernet frame is a 32-bit Cycling Redundancy Check (CRC) of all fields of the entire Ethernet frame, without the trailing FCS field. The input frame is expected not to have the trailing 4-byte FCS field, as it is not possible to detect if this is part of the higher layer data or a FCS field. The function returns the resulting FCS and the entire frame with the FCS appended to the end.

calculate_ipv4_checksum

The calculate_ipv4_checksum function calculates the IPv4 header checksum. The naive version of this is to set the Header Checksum field to 0 and then add 16-bit words from the header together, checking if there was an overflow every time, and if so adding 1 to the result. This can be imagined as the overflowing bit being added at the back. This can result in a second overflow that needs to be checked for again. calculate_ipv4_checksum does this in an optimized fashion by adding all 16-bit words in the header into a 32-bit integer, then subtracting 65535 until the result is less than 65536. This reduces the number of checks for the overflow necessary to a minimum. The operation of removing the 16th bit and adding 1 is also merged into a single subtraction of 65535. The result of this is then inverted to receive the final checksum. The function returns the calculated checksum as well as the provided header with the correct checksum inserted.

calculate_tcp_udp_checksum

The `calculate_tcp_udp_checksum` function accepts an entire IPv6 or IPv4 packet and calculates the TCP or UDP checksum based on this packet. The entire packet is necessary in order to populate the fields of the pseudo-headers used for TCP and UDP, which can be seen in tables 13 and 14. The checksum itself is calculated using the same method as the IPv4 checksum, but is calculated over the whole UDP datagram or TCP segment with the pseudo-header prepended. The function returns the calculated checksum, the TCP/UDP header with the correct checksum inserted, and the whole TCP segment/UDP datagram.

calculate_icmp_checksum

The `calculate_icmp_checksum` function accepts the ICMP packet in its entirety and calculates a checksum for the provided ICMP packet. This checksum is calculated over all fields of the ICMP header as well as included data with the ICMP checksum set to 0 for the purposes of the calculation. The calculation method is the same as for the IPv4, TCP and UDP checksums. The function returns the calculated checksum as well as the provided ICMP packet with the correct checksum inserted in the checksum field.

calculate_icmpv6_checksum

The `calculate_icmpv6_checksum` function accepts the entire IPv6 packet. Unlike IPv4, IPv6 does not include a checksum in the header. For this reason, unlike ICMP which merely includes a checksum of its own header and data, ICMPv6 includes a checksum for the entire IPv6 packet. This is done by prepending a pseudo-header to the ICMPv6 packet. The pseudo-header is the same as for TCP and UDP and can be seen in table 14. The checksum is calculated in the same way as the IPv4, TCP, UDP and ICMP checksums. The function returns the calculated checksum and the provided IPv6 packet with the correct ICMPv6 checksum inserted.

7.5 Helper functions

`libAnonLua` includes two helper functions:

- `ntop(string_raw address)`
- `ip_in_subnet(string_raw address, string CIDR_notation)`

These two functions are meant both as a means to ease debugging and as a means of simplifying use of the library.

The `ntop` function is a wrapper for the `inet_ntop` function from the `arpa/inet.h` C header and returns the provided IPv4 or IPv6 address in a human-readable form.

The `ip_in_subnet` function accepts an IPv4 or IPv6 address in the form of a string containing the raw bytes of the IP address and an address in human-readable CIDR notation (e.g. 192.168.1.0/24) and checks to see if the provided address is in the provided subnet. The function returns a boolean indicating the address is in the subnet (TRUE) or not (FALSE).

```
mislav@Master:~/Masterarbeit/Shanon/Shanon$ tshark -Xlua_script:shanon.lua -e frame.number -T fields -r Tests/MassiveCapture.pcapng
Validated tcp
Validated ipv4
Validated ipv6
Validated arp
Validated icmpv6
Validated icmp
Validated udp
Validated ethernet
1
2
3
4
5
6
7
8
9
```

Figure 4: Running the Shanon packet trace anonymization tool

8 Shanon

The Shanon packet trace anonymization tool is a tool written in Lua as part of this thesis. The name is a combination of Wire**sha**rk and **anon**ymizer.

In this chapter an overview of the functionality of the tool will be provided using the features provided in Chapter 3 as a basis. Due to the large amount of code written a detailed exploration of the various functions used in the tool will not be provided. A general overview of the structure of Shanon can be found in Section 8.1, and an overview of the configuration file in Section 8.4. Those interested in a more detailed look can look at the commented source code at [30].

An example of running Shanon and the generated terminal output can be seen in Figure 4. The anonymized packets in this example are outputted to a file defined in Shanon's cofiguration file. Since tshark will always generate output for each processed packet, Shanon itself does not generate additional output. The `-e frame.number` and `-T fields` parameters are supplied to limit the output of tshark to frame numbers only. These parameters are not necessary to run Shanon, but they reduce visual clutter and make it easier to follow anonymization progress.

8.1 Shanon structure

The Shanon anonymization tool is divided into several files and folders. These can be seen in Figure 5. The main body of the tool is contained in the `shanon.lua` file. Additional lua files containing functions used by Shanon and the individual protocol anonymizers are contained in the `shanonHelpers.lua` and `shanonPolicyValidators.lua` files. The config folder contains the configuration file, `config.lua`, which contains parameters for the anonymization of captured packets. The protocols folder contains the individual anonymiz-

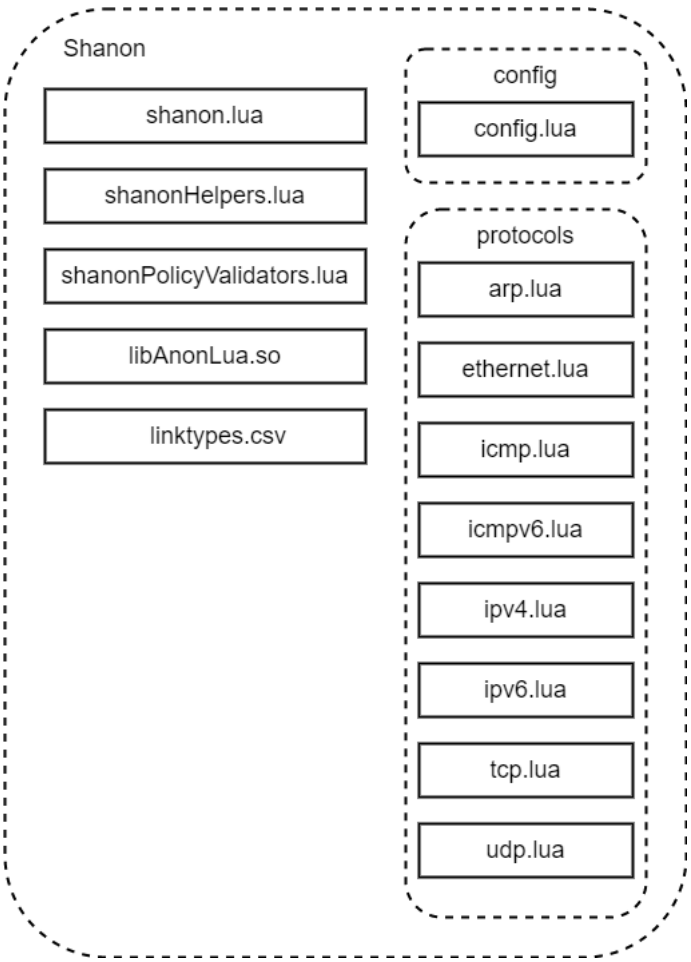

Figure 5: Structure of the Shanon packet trace anonymization tool

ers for the supported protocols: `arp.lua`, `ethernet.lua`, `icmp.lua`, `icmpv6.lua`, `ipv4.lua`, `ipv6.lua`, `tcp.lua` and `udp.lua`. Finally, the `libAnonLua` library and its `linktypes.csv` file must be present for Shanon to operate correctly.

Shanon is built with extensibility and online anonymization in mind. For this reason the main application is built as a loop that iterates over protocols present in a particular captured frame and calls the necessary individual anonymizers to handle the details of these protocols. This functions similarly to how the network stack naturally functions. Higher-layer protocols are anonymized first, then this data is handed over to lower layer anonymizers when they are called to anonymize the lower layer protocols. Any unrecognized protocol that breaks this chain is logged if the appropriate configuration option is set and the data including this unrecognized protocol and above is treated as data by the protocol below. Depending on the settings

of the anonymizer for that protocol this data can either be kept in the same form or it can be replaced depending on the parameters of the particular anonymizer.

This method of anonymization allows new protocol anonymizers to be created and added to the anonymization chain without interfering with existing protocol anonymizers, making Shanon easy to extend with support for new protocols.

Shanon is a single-pass anonymization tool, meaning it only looks at each packet once, anonymizes it based on the information available up to that point, then moves on to the next packet. This enables online anonymization which would be prohibitively memory-intensive if a copy of each packet captured so far had to be kept in memory, but makes anonymizing TCP sequence and acknowledgement numbers difficult and inaccurate.

8.2 Shanon features

Shanon was created with the features described in Chapter 3 in mind. In this section the support for each of these features in Shanon will be discussed.

Extensibility

Shanon is built with extensibility in mind. Potocols are anonymized by individual anonymizers rather than together as part of a monolithic application. Individual anonymizers are called to run on a captured frame in order from the highest layer protocol contained in that frame to the lowest layer, based on the names of protocols used by Wireshark.

Adding a new protocol to Shanon is relatively simple. To do so, a new anonymizer needs to be created and added to the protocols folder. Anonymizers take on the form of Lua modules, meaning they are Lua files which return a table containing their variables and methods. Each protocol anonymizer must contain the following values and methods:

- **filterName** - The name of the protocol this anonymizer is intended for as used by Wireshark. For example “eth” for Ethernet.
- **fauxProtocols** - Certain tags, such as for example ethertype, can appear in the protocol list as a way of presenting information about a certain protocol. These are called faux protocols, as they are not actual protocols. This function accepts a protocol name and returns whether this is a faux protocol related to this protocol or not. For example the presence of an ethertype protocol in the list of protocols contained in

a captured frame means that the captured ethernet frame has a type field and not a length field.

- **relativeStackPosition** - Multiple protocols of the same type can be present in a captured frame. When this occurs, Wireshark will return a table of values when retrieving fields of a particular protocol in order of their appearance from the lowest layer to the highest. The `relativeStackPosition` variable is used to indicate which instance of a protocol in the frame is being processed.

- **anonymize** - This function accepts the buffer containing protocol fields, the list of protocols, the current position in this list, the anonymized frame containing the results of anonymization of upper layers and the configuration and returns a new anonymized frame with the results of the current anonymization added to it, and optionally comments that will be added to the comments in this frame along with comments from other anonymizers, if present.

- **validatePolicy** - This function accepts the Shanon config file and validates the configuration for this specific policy. Should any part of the policy fail to validate the application is expected to crash with an appropriate error message and log. The `validatePolicy` function can validate a policy in any way deemed appropriate, however helper functions are made available in the `shanonPolicyValidators.lua` script that can ease validating policies.

Once an anonymizer has been written and placed in the protocols folder, it can be added to Shanon by adding it to the protocols table in the `shanon.lua` file.

A simple example of an anonymizer which can provide a good overview of how anonymizers are intended to be written is the Ethernet anonymizer, `ethernet.lua`. It contains a small number of fields and options compared to other anonymizers, thus making it easier to understand. The IPv6 anonymizer, `ipv6.lua`, is a good example of a more complex anonymizer since anonymizing IPv6 involves validating a policy which includes a table of subnets that can have different policies applied to them, as well as parsing and anonymizing protocol options.

Policy support

Shanon features a large and detailed policy with many options. Special attention was given to the anonymization of IPv4 and IPv6. Different anonymiza-

tion methods can be set for packets with source or destination addresses in different subnets, and addresses belonging to different subnets can be anonymized in different ways. Using multiple methods of anonymization to anonymize a single address is also supported, such as running a black marker method, erasing a selected amount of bits from the beginning or end, followed by CryptoPAN.

Other protocols also feature a support for various options, with each field having two or more ways in which it can be treated.

One downside of the modular way in which Shanon has been written is the separation of network layers that occurs even during anonymization. With the exception of the IPv4 and IPv6 anonymizers handling checksums for TCP, UDP and ICMPv6 due to pseudo-headers being necessary when calculating the checksums for these protocols, there is no interaction between different anonymizers. This also extends to policy. It may, in some situations, be desirable to anonymize TCP port numbers, for example, differently depending on the subnet the traffic comes from. However, the TCP anonymizer only handles TCP and is unaware of other protocols in the chain. As such, this form of policy is not supported.

A more thorough analysis of each individual option of each protocol would be far too extensive to include here, however the config.lua file included with Shanon is thoroughly commented and can be used to further inform oneself on the supported options.

Meta-data generation

The current version of Shanon generates a very limited amount of meta-data. Currently it is possible to add a comment to each anonymized packet with the version of Shanon and `libAnonLua` which were used to anonymize it. Additionally, the TCP and UDP anonymizers can generate a comment containing the original stream index Wireshark generated for that UDP flow or TCP conversation. Due to the way certain anonymization options change TCP and UDP, Wireshark can become unable to recognize which segments or messages belonged to a particular exchange. To aid in this, adding the original stream index as a comment can help reconstruct the exchange.

The ability to add comments to packets when writing them enables future anonymizers to generate their own meta-data, as well as additional meta-data to be added to existing anonymizers in the future.

Large file support

Wireshark supports large capture files, as does the pcapng filesystem used by Wireshark and by Shanon. Thus, Shanon itself also supports large capture files without any necessary additional actions taken by the user. The file size Shanon is able to process in a real-world scenario is limited by limitations present within Wireshark itself. These are discussed in Section 9.3 as part of the evaluation of Shanon's performance.

Consistency

Every method of anonymization in Shanon is intended to produce consistent results, meaning for the same configuration and packet capture the same anonymized capture should be produced. It is, however, possible to create some inconsistency within the capture file itself due to how IP address anonymization works. The IPv4 and IPv6 configurations enable the creation of entirely different policies for different subnets. If present, these policies will be used if an address matches a subnet instead of the default. If a policy contains two subnets which exchange packets, the policy matching the source IPv6 address will be prioritized over the destination address in protocols where a source and destination address are available. In situations where there is only one address, such as the ICMP Redirect gateway address, the subnet matching that address will be selected. This can lead to a situation where the same address is anonymized in different ways depending on the source of the packet. For this reason policies requiring different anonymization methods for multiple subnets should be carefully crafted to avoid such conflicts.

Performance

A test of Shanon's performance on an example real-world capture can be seen in Section 9.3.

Online anonymization

Just like Wireshark, shanon can be used for both offline and online anonymization, meaning both the anonymization of previously captured packets in the form of a packet capture or a live capture of packets from an interface. This is an inherent property arising from the usage of Wireshark as a basis for the tool, as well as the one-pass nature of how the tool processes captured packets.

Recoverability

Recoverability was not considered during the development of Shanon. Due to the sensitive nature of anonymization, inconsistencies or failures to anonymize can lead to unwanted data leaks. As such, any automated recovery from a failure, or any failure at all, can not be trusted. In addition to the issue of trust, recovery from failure is also a complicated technical issue. The reasons for failure are versatile, and depending on how the failure happened an inconsistent state could be achieved. Finally, even without this feature Shanon is already a large and complicated tool which took significant time to develop so additional development time was not dedicated to this feature. However, recoverability could be achieved in the future by storing metadata related to anonymization either in comments or as additional optional data attached to captured packets using optional blocks in the pcapng filesystem.

8.3 Limitations

During the development of Shanon several issues were encountered. The encountered issues, employed solutions, and limitations imposed by these issues and solutions are documented here.

Partial protocols and partial captures

Shanon does not support dealing with partial protocol data. This means any protocol where fields that should be present but are not can not be processed. This can occur in two cases.

The first case is when a frame was partially captured, either due to a limitation on the number of bytes captured per frame or due to some other reason. In this situation the captured length of the frame will differ from the actual length of the frame. Shanon will crash with a warning message informing the user about this situation and the frame in which it occurred if this is the case.

The second case is when an ICMP or ICMPv6 message contains partial protocol data, such as part of the packet that caused a redirect or destination unavailable message included in the payload. To avoid this from happening, Shanon does not process any protocol layers above the first instance of ICMP or ICMPv6 found in a particular frame. Any higher-layer data above the first instance of these two protocols is instead treated as data by their anonymizers and replaced with an equal-length payload of zeroes.

The decision to implement this restriction rather than account for possibly missing protocol fields was made due to time limitations and the already

large amount of work done when writing this tool. Processing partial protocol data would require the retrieval of each protocol field to take into account the possibility of the data not being present and anonymization to account for only some fields possibly being present, which would require a significant rewriting of protocol field retrieval and anonymization to account for.

Retrieval of optional fields and multiple protocol instances

Some protocols, such as ICMP, have a different structure depending on the type of message that is being sent. Some fields may not be present for every instance of this protocol. Other protocols, such as NDP and IPv6 have additional options or headers which may or may not be present when certain messages are sent or, in the case of IPv6 extension headers, may even be present multiple times. It is also possible for multiple instances of the same protocol to be present in the same capture, for example IP in IP encapsulation.

When retrieving a particular protocol field using methods supplied by Wireshark, the result is a Lua table containing a list of tables with information for all fields of this kind. In a situation where a protocol appears only once and a field is always present for that protocol, simply retrieving the only result of retrieving the field is the correct value for this field. However, in situations where multiple instances of the same protocol are present, the first value will always be the lowest instance of that protocol in the stack of protocols. To overcome this, Shanon counts the number of occurrences of a particular protocol and keeps track of the specific occurrence of a protocol which is being processed.

The same problem is present for optional fields and extension headers, however the problem is more pronounced. Since an optional field or extension header can possibly appear multiple times, or not appear at all, for any particular instance of a protocol, the method applied to individual protocols does not apply to options or extension headers. Instead, Shanon sets boundaries for where in the buffer of protocol data these options or extension headers can be. These boundaries are the beginning of the protocol being processed and either the beginning of the higher layer instance of this protocol, the end of the data this protocol carries, or end of the capture depending on the specific situation. When retrieving the fields of a particular instance of a protocol, Shanon retrieves all options or extension headers which are within the appropriate boundaries for that instance of the protocol in question.

The IPv6 next header value

The structure of IPv6 extension headers presented a unique issue for anonymization. Since Shanon does not support all possible IPv6 extension headers, but only the subset present in RFC 8200 and the details mentioned there, it is necessary to reconstruct the sequence of extension headers according to what is supported and anonymized. An additional difficulty is the way Wireshark makes information on extension headers available through their API. Known extension headers have their own names and sets of fields available, while unrecognized extension headers are grouped together into a single category. Without writing a parser for IPv6 next header values which would be capable of traversing the set of extension headers until an upper layer protocol was reached correctly parsing and assigning the next header value for the upper layer protocol is not possible in all situations.

The solution to this problem employed by Shanon is using the list of protocols contained in the anonymized capture and setting the next header value for recognized protocols when these are present. In a majority of cases the upper layer protocol will be TCP, UDP or ICMPv6, all of which will be correctly recognized and properly set. It is also worth noting that this issue is only present when IPv6 extension headers are processed, as the correct next header value is otherwise present in the IPv6 header itself. In cases where the next layer protocol is not recognized by Shanon, protocol number 59 is used, which signifies no next header is present. Upper layer data provided by anonymizers is retained, however Wireshark does not recognize the upper layer protocol.

TCP sequence and acknowledgement numbers

Anonymizing TCP sequence and acknowledgement numbers is a more difficult problem than was initially anticipated. Two different approaches were attempted, both of which have downsides. The second approach was selected to be the one employed in Shanon.

The first attempt at renumbering involved creating a table for each TCP conversation containing sequence and acknowledgement numbers for each side. Each time a TCP segment was seen, the appropriate sequence and acknowledgement numbers would be incremented so that the conversation flowed from message to message. This presented a problem, however. Information about messages being retransmitted was completely lost and sequentially recorded messages received sequential sequence numbers regardless of their actual order in the sequence.

The second approach was to keep track of the original and expected

sequence and acknowledgement numbers and only increment when segments are encountered that fit into this. For segments encountered that are out of order, a decremented sequence number is set so that the segment appears as out of order in the capture. This approach, however, presents a new issue. Should a series of segments from both sides be missing, the incrementing will stop and all following segments will appear out of order.

In addition to the issues encountered when implementing the two approaches, an interesting observation was made with keep-alive segments. Shanon preserves the information that a TCP segment was sent without data and in situations where such a segment is encountered a minimum payload is not attached when configured to do so. In cases where TCP keep-alive segments do not have data attached to them this preserves the semantics of the TCP keep-alive. However, if a keep-alive is sent with a single octet of data, which may be the case, the semantics are lost.

Lack of IPv6 tools

The lack of tools with support for IPv6 severely limited the extent to which ICMPv6, NDP and IPv6 could be tested during development. For example, nping [60], a packet generator and ping utility that comes with nmap, was able to generate ICMP messages of various types for the purposes of testing, however failed when it came to ICMPv6. DPKT [88], a Python module for packet generation, has very little useful documentation and few examples, all of which are for IPv4. It was possible to generate ICMPv6 packet too big, echo, parameter problem and destination unreachable messages using DPKT, however it required trial and error and looking at sources to write a few lines of code to actually produce these packets, and the packets themselves were of limited utility. The thc-ipv6 IPv6 attack toolkit [46] was also used to generate NDP router advertisements.

8.4 Configuring Shanon

The configuation file for Shanon is located in the config folder and is named `config.lua`. It is a Lua file which contains configuration options for Shanon in the form of a Lua table. The table is further subdivided into a set of general options: the output file, the key file for the CryptoPAN algorithm, logging settings, meta-data settings, and an anonymization policy which is further subdivided into tables containing the anonymization settings for each protocol Shanon supports. A more detailed look into all the configurable options in this policy would be too extensive to include here. Instead, those

interested should look at the detailed comments in the policy file itself which describe the available options for each protocol in detail. The source code for Shanon is available at [30].

8.5 Running the tool

In order for Shanon to run, all files and folders included with the tool must be present in the location the main script, `shanon.lua`, will be run from. Additionally, a compiled version of `libAnonLua` [29] and the `linktypes.csv` file used by `libAnonLua` must be present in the same directory and must match the expected `libAnonLua` version for the version of Shanon in use.

Shanon is intended to be run with tshark, the terminal-based version of Wireshark. In order to run Shanon when processing a packet capture with tshark, tshark needs to be started from the folder containing Shanon with the following additional parameters:

```
-X lua_script:shanon.lua
```

9 Evaluation

In this chapter the anonymization methods and results of the Shanon packet capture anonymization tool will be evaluated.

9.1 Anonymization methods

With the exception of the CryptoPAN algorithm described in [38], the methods used by Shanon rely on the principle of K-anonymity [95]. The result of these actions is the removal of an amount of information that serves to distinguish a particular packet, sequence of packets, or source or destination host from others in the capture and in the real world. When a certain packet or host is indistinguishable from k-1 others, it can be said that it is k-anonymous.

Methods such as the black marker, named after the practice of censoring bits of text by going over them with an opaque black marker, or setting a field to zero, reduce the amount of information present in a field, rendering that field indistinguishable from a number of other fields with the same changes made depending on the amount of bits erased. Setting field values to particular values ensures devices cannot be identified based on unique values that may be in use for particular settings, such as lifetime values in router advertisements, or time to live and hop count values in IPv4 and IPv6 headers which have different default values on different OSes. Recalculating checksums eliminates the possibility of comparing anonymized data to real data to deanonymize part of it or using checksums to validate attempted deanonymization. Replacing TCP and UDP payloads with minimal payloads of zeroes and recalculating TCP sequence and acknowledgement numbers guards against attacks that use the size and order of messages to fingerprint a source or destination such as network traffic fingerprinting described in [56]. Shanon also sets any field that is reserved for future use in a protocol to zero, and fields such as the IPv4 and IPv6 version fields which should have constant specific values to their standard values, eliminating the possibility of using these fields to insert patterns that can be recognized at a later date. This protects against attacks that rely on injecting and later recognizing patterns in network traffic using these fields. While the individual anonymization methods used in Shanon are sound, as demonstrated above, the quality of the anonymization of a packet capture depends on properties of the environment the capture was taken in and the combination of anonymization methods used. Due to this a set of sound methods is not sufficient to guarantee protection, but instead the details of the capture it-

self must be considered and a policy created that is suitable to the level of protection desired and the capture being anonymized.

9.2 The possibility of leaks

Shanon utilizes defensive transformation [41]. The retrieved original values of fields for each protocol anonymized by Shanon are stored in separate variables from the anonymized fields. When anonymizing a protocol, the anonymized values are concatenated to create a new, anonymized instance of a protocol. The anonymized values are derived from the original values using the anonymization methods chosen by the user in the configuration file. As a result of this no original values can accidentally pass through anonymization and end up in the anonymized capture. This also removes the possibility of leaks from higher-layer encapsulated data, as this data can only be included in the output if it was processed by Shanon. Shanon also constructs its own pcapng capture files which contain the minimum set of necessary headers and options for the correct functioning of the files and Shanon itself, thus avoiding potential information leaks from within the structure of the capture files themselves.

9.3 Performance

Shanon was tested on a 568 MB capture containing 638993 packets on a VirtualBox VM running 64-bit Ubuntu 20.04 LTS with 2 cores from a Core i7-6700K CPU running at 4.0 GHz with 4.2 GHz turbo enabled. The virtual machine was run from an Crucial P1 1 TB NVMe solid state drive. A detailed structure of the capture can be seen in Section 9.4.

Running on this configuration, Shanon achieved a packet processing rate of 2075 packets per second having processed all packets in 5 minutes, 7.868 seconds as reported by the utility program `time` which was used to time the execution. Based on the observed read/write intensity it does not seem likely that the SSD played a crucial role in enabling this speed, as the reported CPU utilization was near to or at 100% at all times.

We consider this processing speed sufficient to satisfy users in both offline and online anonymization scenarios. Compared to tcpmkpub, developed by Pang et al. Shanon falls short, however. In their test [72] Pang et al. processed 165 million packets in 2.9 hours. Translated to packets per second, this results in 15804 packets per second. It is also worth noting that their test was conducted on older hardware.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	638993	100.0	574908405	575 k	0	0	0
Ethernet	100.0	638993	1.6	8945902	8.953	0	0	0
Logical-Link Control	0.0	3	0.0	18	0	0	0	0
Logical-Link Control Basic Format XID	0.0	3	0.0	9	0	3	9	0
Link Layer Discovery Protocol	0.0	8	0.0	352	0	8	352	0
Internet Protocol Version 6	0.0	39	0.0	1560	1	0	0	0
Internet Control Message Protocol v6	0.0	39	0.0	1868	1	39	1868	1
Internet Protocol Version 4	99.8	637984	2.2	12759796	12 k	0	0	0
User Datagram Protocol	24.0	153620	0.2	1228960	1.229	0	0	0
Simple Service Discovery Protocol	0.1	356	0.0	53508	53	356	53508	53
Network Time Protocol	0.0	12	0.0	576	0	12	576	0
NetBIOS Name Service	0.0	305	0.0	15466	15	305	15466	15
NetBIOS Datagram Service	0.0	23	0.0	4596	4	0	0	0
SMB (Server Message Block Protocol)	0.0	23	0.0	2710	2	0	0	0
SMB MailSlot Protocol	0.0	23	0.0	575	0	0	0	0
Microsoft Windows Browser Protocol	0.0	23	0.0	732	0	23	732	0
Multicast Domain Name System	0.0	173	0.0	5276	5	173	5276	5
Link-local Multicast Name Resolution	0.0	200	0.0	4732	4	200	4732	4
ETSI Distribution & Communication Protocol (for DRM)	0.0	30	0.0	32607	32	14	14949	14
Malformed Packet	0.0	16	0.0	0	0	16	0	0
Dynamic Host Configuration Protocol	0.0	18	0.0	5224	5	18	5224	5
Domain Name System	0.3	2072	0.0	181677	181	2072	181677	181
Data	23.5	150431	29.3	168463956	168 k	150431	168463956	168 k
Transmission Control Protocol	75.8	484068	66.6	383051408	383 k	352490	326278628	326 k
Transport Layer Security	20.6	131643	63.8	366803539	367 k	129522	360253217	360 k
SSH Protocol	0.0	71	0.0	62724	62	71	62724	62
Malformed Packet	0.0	81	0.0	0	0	81	0	0
Hypertext Transfer Protocol	0.1	410	1.0	5734059	5.738	248	88698	88
Portable Network Graphics	0.0	41	0.4	2419212	2.421	41	2428256	2.430
Media Type	0.0	41	0.1	413677	414	41	345171	345
Line-based text data	0.0	26	0.1	443336	443	26	448777	449
JPEG File Interchange Format	0.0	42	0.1	834390	835	42	855402	856
Compuserve GIF	0.0	3	0.0	7080	7	3	7895	7
Domain Name System	0.0	6	0.0	3566	3	6	3566	3
Data	0.2	1497	0.3	1552473	1.553	1497	1556723	1.558
Internet Group Management Protocol	0.0	29	0.0	472	0	29	472	0
Internet Control Message Protocol	0.0	267	0.0	50455	50	267	50455	50
Address Resolution Protocol	0.2	959	0.0	32306	32	959	32306	32

Figure 6: Protocol hierarchy of original capture

Due to the data that needs to be stored when processing a capture file, Wireshark necessarily has a large memory footprint when processing large captures. For a test file 568 MB in size containing 638993 packets the memory footprint exceeded 2 GB. Large captures containing fewer packets or not containing protocols for which Wireshark needs to maintain a lot of state information, such as TCP, may fare better, however this was not tested. This is a limitation inherent to Wireshark itself and thus one Shanon can not overcome.

9.4 Anonymization results

The following results were observed when anonymizing a capture of private computer usage over the span of 2 hours, 13 minutes and 13 seconds. The capture contains a total of 638993 packets with a total size of 596 MB. The protocol hierarchy of the original and anonymized captures can be seen in Figures 6 and 7.

The percentages are provided by Wireshark and are rounded up to a single decimal point of precision. Where percentages are not included the number was too small to display and Wireshark displayed 0.0 instead. One issue with using this capture is the low amount of IPv6 traffic recorded. However, given the lack of tool support as described in Section 8.3 and the low level of IPv6 adoption [44] this was deemed acceptable. During development each

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	638993	100.0	47149649	47 k	0	0	0
Ethernet	100.0	638993	19.0	8945902	8.953	0	0	0
Logical-Link Control	0.0	3	0.0	138	0	0	0	0
Data	0.0	3	0.0	126	0	3	126	0
Link Layer Discovery Protocol	0.0	8	0.0	400	0	8	400	0
Internet Protocol Version 6	0.0	39	0.0	1560	1	0	0	0
Internet Control Message Protocol v6	0.0	39	0.0	1868	1	39	1868	1
Internet Protocol Version 4	99.8	637984	27.1	12759680	12 k	0	0	0
User Datagram Protocol	24.0	153620	2.6	1228960	1.229	0	0	0
Data	24.0	153620	6.5	3072400	3.074	153620	3072400	3.074
Transmission Control Protocol	75.8	484068	37.7	17793464	17 k	483570	17773544	17 k
Data	0.1	498	0.0	9960	9	498	9960	9
Internet Group Management Protocol	0.0	29	0.0	588	0	29	588	0
Internet Control Message Protocol	0.0	267	0.1	50455	50	267	50455	50
Address Resolution Protocol	0.2	959	0.1	47950	47	959	47950	47

Figure 7: Protocol hierarchy of anonymized capture

individual protocol anonymizer was tested on a small scale with generated or captured traffic, depending on the availability of such traffic. For the purposes of this evaluation, however, using a larger real-world capture is more suitable as it demonstrates the functionality of the tool in a realistic scenario. Using generated traffic equivalent to the generated traffic used to test during development would not demonstrate that the tool functions properly, but instead demonstrate that the tool functions when encountering the exact traffic it was written to function on. The results demonstrated below are based on statistics generated by using the various options in the statistics menu in Wireshark. Testing each possible configuration of the anonymization tool and producing statistics for each would be too extensive, so instead the focus is put on a subset of results that were deemed significant.

Protocol hierarchy of the anonymized capture

The protocol hierarchy in the anonymized capture remained unchanged for all supported protocols. Unsupported protocols are treated as data by Shanon, meaning they are replaced by zero payloads of minimal or equal length depending on the protocol anonymizer and configuration. The result of this is that some unsupported protocols disappeared from the hierarchy in the anonymized capture. Protocols that can be identified by their protocol number in Ethernet frames and IPv6 and IPv4 packets remained in the list, despite their contents being replaced with zeroes, however when preserving only the range of TCP and UDP ports all protocols carried by TCP and UDP showed up as data in the protocol hierarchy. Preserving port numbers preserved more of the original hierarchy, but there were still losses in information. One example of lost information occurs with traffic on TCP port 443. In the original capture, Transport Layer Security (TLS) is appropriately identified based on the carried data. With the higher-layer protocol data automatically stripped by Shanon, the packets are instead marked as Secure

Sockets Layer (SSL) or TCP. Since none of the protocols encapsulated in TCP and UDP are anonymized by Shanon a more in-depth look at the protocols lost due to anonymization and the causes of this loss is outside of the scope of this work. Instead, what is important to note is that for some protocols the semantics of the protocol contents are relevant to the correct identification of those protocols by Wireshark and as a result the removal of protocol payloads by Shanon results in failure to identify these protocols.

Metadata contained in the original capture

The original capture contained information on the hardware and software the capture was taken on. This consisted of the CPU manufacturer and model, the OS version and build and the version of Wireshark used to make the capture. There was also metadata about the interface the capture was made from, containing the original name of the interface, number of dropped packets, type and the packet size limit for that interface. Such additional information can be included in a capture file by using the appropriate options defined in the pcapng standard. [61] These options are not used by Shanon nor is this data retrieved when anonymizing a capture, and thus this additional information is lost. In the anonymized capture these values showed up as unknown or 0.

Shanon can either preserve the original timestamps stored in the pcapng file or use relative timestamps the arrival times of which are relative to the time of the capture being recorded. When using relative timestamps the capture start time shows up as 1970-01-01 01:00:00 in Wireshark statistics. The duration of the capture and time between individual packets is preserved.

An example of some of the information associated with a frame in a Wireshark packet capture can be seen in Figure 8. The captured frame itself has a timestamp associated with it, while the interface information is stored in a separate block in the pcapng filesystem called an Interface Description Block (IDB). The frame references this block through the Interface id property. In the case of the anonymized frame, which can be seen in Figure 9, the timestamp is relative to the beginning of the capture, and since no additional metadata about the interface is stored in the output file, there is no additional information about the capture device present in Wireshark's display of the anonymized frame.

```
▾ Frame 22: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{D802B43A-1C22-4DF8-8764-9FA83E6C0BDE}, id 0
  ▸ Interface id: 0 (\Device\NPF_{D802B43A-1C22-4DF8-8764-9FA83E6C0BDE})
    Encapsulation type: Ethernet (1)
    Arrival Time: Aug  1, 2020 07:21:36.149173000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1596259296.149173000 seconds
    [Time delta from previous captured frame: 0.021020000 seconds]
    [Time delta from previous displayed frame: 0.021020000 seconds]
    [Time since reference or first frame: 2.685641000 seconds]
    Frame Number: 22
    Frame Length: 66 bytes (528 bits)
    Capture Length: 66 bytes (528 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: TCP SYN/FIN]
    [Coloring Rule String: tcp.flags & 0x02 || tcp.flags.fin == 1]
```

Figure 8: Frame information from Frame 22 of the original capture

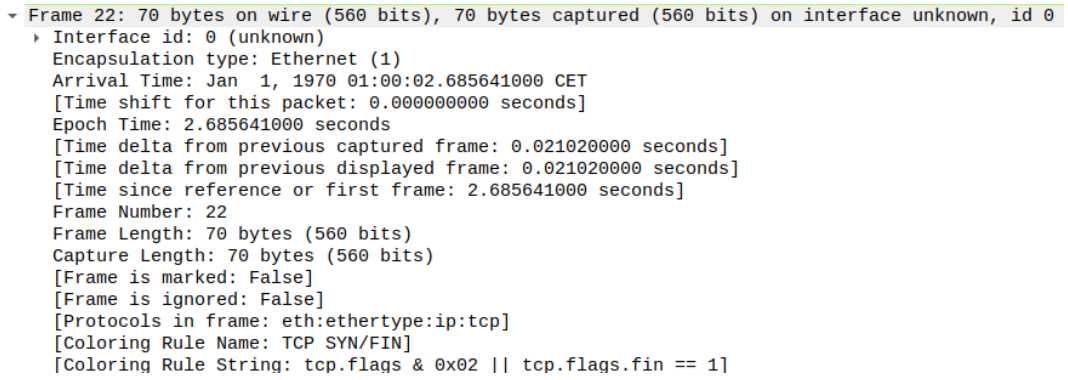

Figure 9: Frame information from Frame 22 of the anonymized capture

Effects on capture size

Shanon does not keep information it can not process, such as IPv6 extension headers, IPv4, TCP and ICMPv6 options other than the ones implemented. In the cases of Ethernet, IPv4 and IPv6 if no data is provided by a previous anonymizer Shanon uses a minimum payload. For TCP and UDP Shanon provides a choice of 5 options: keeping the payload, using a 20 byte payload of zeroes, using a payload of zeroes of the same length, and two options which keep an anonymized payload if present or one of the two zero payload options if not. As a result of this, the capture size can change compared to the original. In a test using the “Anonymized1” option, which preserves an anonymized payload if present, and provides a 20 byte payload of zeroes if not, the anonymized capture file was reduced in size from the 596 MB original down to 125 MB. This property can be used, for example, in combination with preserving TCP sequence numbers, to retain information about the original sizes of TCP segments while also reducing the size of a packet capture to one that is easier to transfer and store.

Effects of storing metadata as packet comments

When generating capture file properties, Wireshark also lists all packet comments in the generated statistics. Comments generated on a per-packet basis by Shanon resulted in list of 638993 comments. While the exact time it took Wireshark to generate statistics was not measured, it took significantly longer and Wireshark even became unresponsive at one moment. Using the same anonymization settings without storing any comments resulted in a significantly faster generation time for this statistic.

Wireshark allows for packet filtering based on the contents of comments.

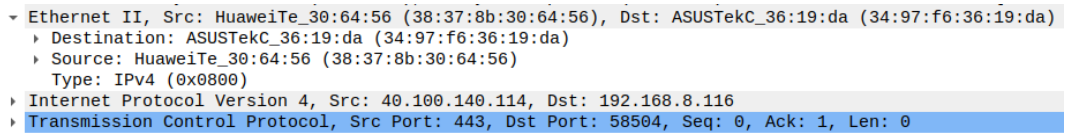

Figure 10: Ethernet header from Frame 22 of the original capture

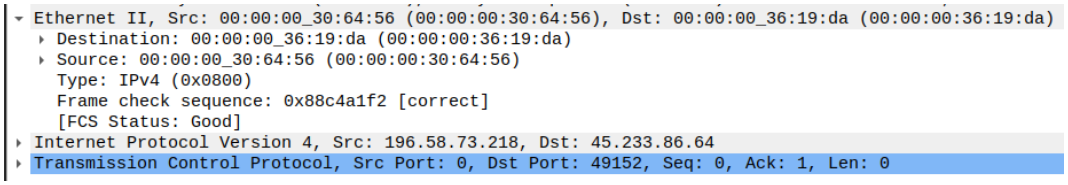

Figure 11: Ethernet header from Frame 22 of the anonymized capture

In combination with the generated metadata this allows original TCP and UDP streams, as identified by Wireshark in the original captures, to be preserved and filtered out. The level of detail still present in these streams depends on other anonymization options chosen.

MAC addresses

When using a black marker method on the first 24 bits of MAC addresses, the number of endpoints was preserved between the original and anonymized capture, however none of the addresses in the anonymized capture showed the manufacturer name in Wireshark. This achieves a possibly desired goal of anonymization which is concealing the hardware used when making a capture. An example of this effect can be seen in Figures 10 and 11. In the original capture in Figure 10 Wireshark displays a short manufacturer name. In the anonymized capture in Figure 11 this name is no longer present as it was removed by removing the first 24 bits. The multicast bits in the MAC address are also lost in the process, and the broadcast MAC address is also not labelled as such by Wireshark. Depending on other anonymization settings some of this information may be preserved through context, but some will inevitably be lost.

IPv4 and IPv6 addresses

To show the total number of different IPv4 and IPv6 addresses as well as the packets sent and received endpoint statistics were used. When using CryptoPAN to anonymize IPv4 and IPv6 addresses the observed number of addresses and packets sent and received did not change. The addresses

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
All Addresses	637984				0,0798	100%	10,5500	7397,086
192.168.8.118	423716				0,0530	66,41%	4,8600	7854,661
185.117.88.9	423235				0,0529	66,34%	4,8600	7854,661
192.168.8.116	214253				0,0268	33,58%	10,5400	7397,086
74.125.13.168	47063				0,0059	7,38%	7,8000	6430,450
173.194.160.136	20637				0,0026	3,23%	5,1200	6690,775
74.125.108.8	19064				0,0024	2,99%	8,4300	6215,560
74.125.99.44	10994				0,0014	1,72%	7,8300	6594,610
172.217.23.54	10880				0,0014	1,71%	5,7600	7160,091

Figure 12: Partial list of IPv4 addresses included in the original capture

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
All Addresses	637984				0,0798	100%	10,5500	7397,086
45.233.86.67	423716				0,0530	66,41%	4,8600	7854,661
91.56.152.211	423235				0,0529	66,34%	4,8600	7854,661
45.233.86.64	214253				0,0268	33,58%	10,5400	7397,086
190.139.96.171	47063				0,0059	7,38%	7,8000	6430,450
65.118.125.215	20637				0,0026	3,23%	5,1200	6690,775
190.139.27.51	19064				0,0024	2,99%	8,4300	6215,560
190.139.17.237	10994				0,0014	1,72%	7,8300	6594,610
64.187.189.76	10880				0,0014	1,71%	5,7600	7160,091

Figure 13: Partial list of anonymized IPv4 addresses in the anonymized capture

in the anonymized capture and original capture were different, but a 1 to 1 mapping and the preservation of prefix relationships could be seen. A partial list of original and anonymized IPv4 addresses from the example capture used in this evaluation can be seen in Figures 12 and 13. The relationships between the addresses demonstrate the prefix-preserving properties of the CryptoPAN anonymiziation algorithm.

Truncating parts of the IPv4 addresses predictably resulted in a decrease of observed addresses. In this capture truncating the 16 least significant bits produced far fewer addresses than truncating the most significant 16 bits. This makes sense for the capture in question since multiple blocks of addresses sharing the same 16-bit prefix were present. When truncating this prefix the different addresses would still remain different, however truncating 16 bits after this prefix would result in blocks of addresses sharing the same 16-bit prefix to merge into one address. One example of such a block of addresses is 172.217.0.0/16 owned by Google.

TCP and UDP

When analyzing TCP and UDP using Wireshark endpoint statistics an endpoint is a combination of IPv4 or IPv6 address and TCP or UDP port.

In the original trace the number of endpoints for TCP was 1040, and for UDP 1065. In the anonymized trace TCP and UDP port ranges were preserved by mapping values to the first port of each range. This resulted in the number of endpoints dropping to 239 for TCP and 54 for UDP. This result makes sense since the anonymization method results in ports from the same range all becoming the same port. As a result, multiple different endpoints in the same port range for any given IPv4 or IPv6 address would merge into one endpoint.

Keeping the original port numbers while also using CryptoPAN to ensure IPv4 and IPv6 addresses are transformed 1 to 1 results in the same number of TCP and UDP endpoints as the original capture: 1040 for TCP and 1065 for UDP.

Examples of original and anonymized TCP headers can be seen in Figures 14 and 15. The mapping of port numbers to the first port of their respective ranges can be seen. The original source port 443 has been mapped to 0, and the destination port 58504 has been mapped to 49152. Sequence numbers have been remapped, and due to this the original sequence numbers which begin at a random initially chosen value begin at 0. The relative sequence numbers are unaffected by this change. Flags have been preserved. Several TCP options are present. As mentioned in Section 5.9, the order of Options may help reveal information about the source of the TCP frames. Shanon implements its own order of TCP Options, followed by No-Operation Options used as padding, with an End of Option List Option at the end. This prevents the order of TCP Options from being used to fingerprint the source device.

9.5 Comparison of original and anonymized captures in the Wireshark window

In this section a side-by-side comparison of a portion of the original and anonymized captures will be made based on the first 53 captured packets. The original capture can be seen in Figure 16. The anonymized capture can be seen in Figure 17.

The order of the packets is preserved by Shanon, as each packet is read, anonymized, then written to the output file in a single-threaded process before the next one is read. As a result the 53 packets visible in the

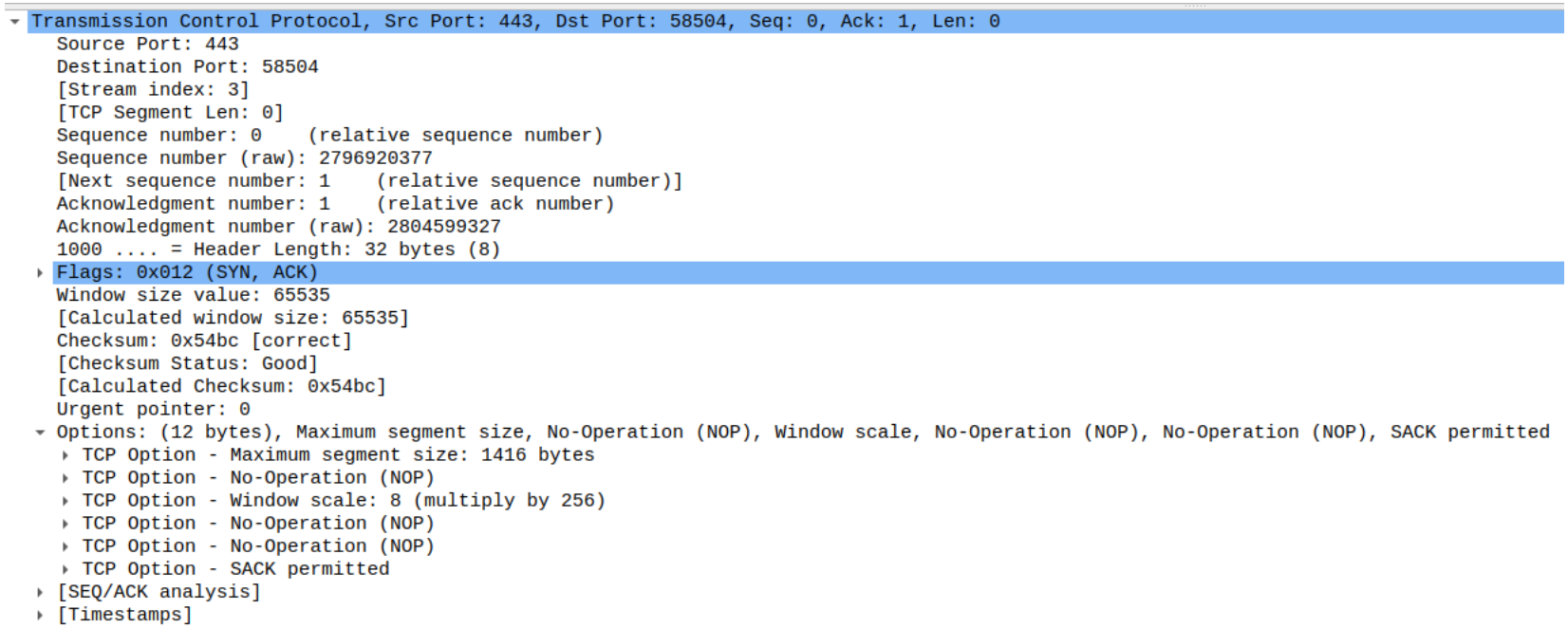

Figure 14: TCP header from Frame 22 in the original capture with TCP Options

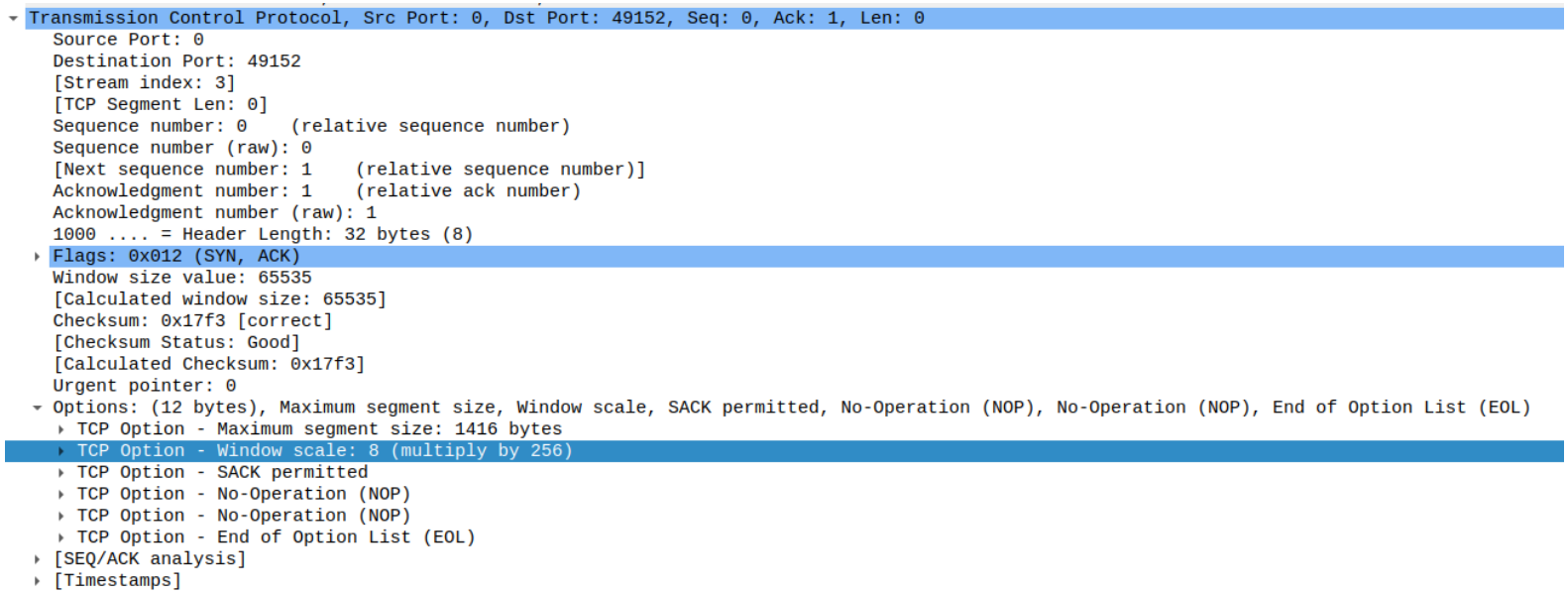

Figure 15: TCP header from Frame 22 in the anonymized capture with TCP Options

original and anonymized captures are the same 53 packets before and after anonymization through Shanon.

The second column in the capture window shows the relative time of a packet's arrival compared to the beginning of the capture. With the exception of a few packets the relative arrival times of the packets match. The few packets where this is not the case would appear to suffer from a floating point rounding error. An example of that is packet 11 in the anonymized capture.

The source and destination addresses (columns 3 and 4) have been anonymized using the CryptoPAN algorithm. The consistency of prefixes can be observed when looking at packets from different devices in the local network, such as packets 15 and 17. Just like in the original capture, these packets share the first three octets of their source IP addresses in the anonymized capture.

The protocol column (column 5) demonstrates the loss of information that occurs due to the TCP and UDP source and destination ports being remapped. As a result of this, Wireshark is not able to identify the upper layer protocol in the anonymized capture. In the original capture Wireshark is able to determine the upper layer protocol. It is worth noting that the exact version of TLS in use is determined from the application layer payload and without it Wireshark would display SSL in the protocol field.

In the info column (column 6) of the original capture we can see many packets marked as having incorrect checksums. Due to checksums always being recalculated in the anonymized capture, this information is lost and all checksums appear to be correct. This results in the packets at the start of the capture being highlighted in black. Many of the later packets in the anonymized capture are marked as Out-Of-Order, which also results in a black highlight. TCP sequence and acknowledgement numbers are difficult to anonymize in a consistent manner without maintaining a large amount of state information or using a two-pass approach. Neither of these approaches would be suitable for online anonymization. As a result, the anonymization of TCP sequence and acknowledgement numbers does not always result in a sequence mirroring the order of sequence and acknowledgement numbers in the original. A detailed explanation can be found in Section 8.3.

The last three columns are not present in Wireshark's window by default, and were added for debugging purposes during development, but also serve to demonstrate the effects of anonymization on the TCP payload sizes and sequence and acknowledgement numbers. In this anonymization run Shanon was set to replace TCP payloads with a fixed minimum-size pay-

load of 20 bytes. As a result any TCP frame which had a payload has had it replaced with a 20-byte payload. When combined with recalculating TCP sequence and acknowledgement numbers, this conceals the actual sizes of the exchanged packets and prevents fingerprinting attacks based on sequences of packet sizes.

No.	Time	Source	Destination	Protocol	Length	Info	Payload	SEQ	ACK
1	0.000000	192.168.8.116	162.159.133.234	TCP	54	58180 → 443 [ACK] Seq=1 Ack=1 Win=1026 [TCP CHECKSUM INCORRECT] Le…	0	1	1
2	0.113076	192.168.8.116	192.168.8.255	UDP	305	54915 → 54915 Len=263			
3	0.946636	162.159.133.234	192.168.8.116	TLSv1.2	136	Application Data	82	1	1
4	0.987711	192.168.8.116	162.159.133.234	TCP	54	58180 → 443 [ACK] Seq=1 Ack=83 Win=1026 [TCP CHECKSUM INCORRECT] L…	0	1	83
5	1.117359	192.168.8.116	192.168.8.255	UDP	305	54915 → 54915 Len=263			
6	1.878697	52.159.49.199	192.168.8.116	TLSv1.2	415	Application Data	361	1	1
7	1.881396	192.168.8.116	52.159.49.199	TLSv1.2	2628	Application Data	2574	1	362
8	1.979684	52.159.49.199	192.168.8.116	TLSv1.2	415	[TCP Spurious Retransmission] , Application Data	361	1	1
9	1.979704	192.168.8.116	52.159.49.199	TCP	66	[TCP Dup ACK 7#1] 58072 → 443 [ACK] Seq=2575 Ack=362 Win=1028 [TCP…	0	2575	362
10	2.067644	52.159.49.199	192.168.8.116	TCP	60	443 → 58072 [ACK] Seq=362 Ack=2575 Win=2052 Len=0	0	362	2575
11	2.106988	192.168.8.116	192.168.8.255	UDP	305	54915 → 54915 Len=263			
12	2.337913	192.168.8.116	192.168.8.1	DNS	114	Standard query 0x1e71 A fc783b562e938ba3309e43b877cf1ad5.fp.measur…			
13	2.368769	192.168.8.116	192.168.8.1	DNS	114	Standard query 0x1e71 A fc783b562e938ba3309e43b877cf1ad5.fp.measur…			
14	2.496643	192.168.8.116	52.202.187.250	TLSv1.2	317	Application Data	263	1	1
15	2.540294	192.168.8.1	192.168.8.116	DNS	312	Standard query response 0x1e71 A fc783b562e938ba3309e43b877cf1ad5.…			
16	2.540368	192.168.8.1	192.168.8.116	DNS	312	Standard query response 0x1e71 A fc783b562e938ba3309e43b877cf1ad5.…			
17	2.540713	192.168.8.116	40.100.140.114	TCP	66	58504 → 443 [SYN] Seq=0 Win=64240 [TCP CHECKSUM INCORRECT] Len=0 M…	0	0	0
18	2.588525	192.168.8.116	40.100.140.114	TCP	66	58505 → 443 [SYN] Seq=0 Win=64240 [TCP CHECKSUM INCORRECT] Len=0 M…	0	0	0
19	2.663645	52.202.187.250	192.168.8.116	TLSv1.2	87	Application Data	33	1	264
20	2.664607	52.202.187.250	192.168.8.116	TLSv1.2	301	Application Data	247	34	264
21	2.664621	192.168.8.116	52.202.187.250	TCP	54	57838 → 443 [ACK] Seq=264 Ack=281 Win=1026 [TCP CHECKSUM INCORRECT…	0	264	281
22	2.685641	40.100.140.114	192.168.8.116	TCP	66	443 → 58504 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1416 WS=256…	0	0	1
23	2.685681	192.168.8.116	40.100.140.114	TCP	54	58504 → 443 [ACK] Seq=1 Ack=1 Win=263168 [TCP CHECKSUM INCORRECT] …	0	1	1
24	2.685927	192.168.8.116	40.100.140.114	TLSv1.2	571	Client Hello	517	1	1
25	2.727791	40.100.140.114	192.168.8.116	TCP	66	443 → 58505 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1416 WS=256…	0	0	1
26	2.727836	192.168.8.116	40.100.140.114	TCP	54	58505 → 443 [ACK] Seq=1 Ack=1 Win=263168 [TCP CHECKSUM INCORRECT] …	0	1	1
27	2.728093	192.168.8.116	40.100.140.114	TLSv1.2	571	Client Hello	517	1	1
28	2.838733	40.100.140.114	192.168.8.116	TCP	60	443 → 58504 [ACK] Seq=1 Ack=518 Win=524800 Len=0	0	1	518
29	2.840781	40.100.140.114	192.168.8.116	TCP	1470	443 → 58504 [ACK] Seq=1 Ack=518 Win=524800 Len=1416 [TCP segment o…	1416	1	518
30	2.840804	40.100.140.114	192.168.8.116	TCP	1470	443 → 58504 [ACK] Seq=1417 Ack=518 Win=524800 Len=1416 [TCP segmen…	1416	1417	518
31	2.840818	192.168.8.116	40.100.140.114	TCP	54	58504 → 443 [ACK] Seq=518 Ack=2833 Win=263168 [TCP CHECKSUM INCORR…	0	518	2833
32	2.841755	40.100.140.114	192.168.8.116	TCP	1470	443 → 58504 [ACK] Seq=2833 Ack=518 Win=524800 Len=1416 [TCP segmen…	1416	2833	518
33	2.841755	40.100.140.114	192.168.8.116	TLSv1.2	243	Server Hello, Certificate, Certificate Status, Server Key Exchange…	189	4249	518
34	2.841772	192.168.8.116	40.100.140.114	TCP	54	58504 → 443 [ACK] Seq=518 Ack=4438 Win=263168 [TCP CHECKSUM INCORR…	0	518	4438
35	2.848728	192.168.8.116	40.100.140.114	TLSv1.2	212	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Messa…	158	518	4438
36	2.848862	192.168.8.116	40.100.140.114	TLSv1.2	147	Application Data	93	676	4438
37	2.848992	192.168.8.116	40.100.140.114	TLSv1.2	352	Application Data	298	769	4438
38	2.888800	40.100.140.114	192.168.8.116	TCP	1470	443 → 58505 [ACK] Seq=1 Ack=518 Win=524800 Len=1416 [TCP segment o…	1416	1	518
39	2.888801	40.100.140.114	192.168.8.116	TCP	1470	443 → 58505 [ACK] Seq=1417 Ack=518 Win=524800 Len=1416 [TCP segmen…	1416	1417	518
40	2.888831	192.168.8.116	40.100.140.114	TCP	54	58505 → 443 [ACK] Seq=518 Ack=2833 Win=263168 [TCP CHECKSUM INCORR…	0	518	2833
41	2.889694	40.100.140.114	192.168.8.116	TCP	1470	443 → 58505 [ACK] Seq=2833 Ack=518 Win=524800 Len=1416 [TCP segmen…	1416	2833	518
42	2.889694	40.100.140.114	192.168.8.116	TLSv1.2	243	Server Hello, Certificate, Certificate Status, Server Key Exchange…	189	4249	518
43	2.889710	192.168.8.116	40.100.140.114	TCP	54	58505 → 443 [ACK] Seq=518 Ack=4438 Win=263168 [TCP CHECKSUM INCORR…	0	518	4438
44	2.896532	192.168.8.116	40.100.140.114	TLSv1.2	212	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Messa…	158	518	4438
45	2.992729	40.100.140.114	192.168.8.116	TLSv1.2	380	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	326	4438	676
46	2.992730	40.100.140.114	192.168.8.116	TLSv1.2	123	Application Data	69	4764	676
47	2.992767	192.168.8.116	40.100.140.114	TCP	54	58504 → 443 [ACK] Seq=1067 Ack=4833 Win=262912 [TCP CHECKSUM INCOR…	0	1067	4833
48	2.992891	192.168.8.116	40.100.140.114	TLSv1.2	92	Application Data	38	1067	4833
49	3.015602	40.100.140.114	192.168.8.116	TLSv1.2	92	Application Data	38	4833	1067
50	3.016656	40.100.140.114	192.168.8.116	TLSv1.2	445	Application Data	391	4871	1067
51	3.016675	192.168.8.116	40.100.140.114	TCP	54	58504 → 443 [ACK] Seq=1105 Ack=5262 Win=262400 [TCP CHECKSUM INCOR…	0	1105	5262
52	3.018283	192.168.8.116	40.100.140.114	TLSv1.2	147	Application Data	93	1105	5262
53	3.043671	40.100.140.114	192.168.8.116	TLSv1.2	380	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message	326	4438	676

Figure 16: First 53 packets of Original Wireshark Capture

No.	Time	Source	Destination	Protocol	Length	Info	Payload	SEQ	ACK
1	0.000000000	45.233.86.64	78.246.224.199	TCP	64	[TCP Window Update] 49152 → 0 [ACK] Seq=1 Ack=1 Win=1026 Len=0	0	1	1
2	0.113076000	45.233.86.64	45.233.86.148	UDP	66	49152 → 49152 Len=20			
3	0.946636000	78.246.224.199	45.233.86.64	TCP	78	0 → 49152 [PSH, ACK] Seq=1 Ack=1 Win=95 Len=20	20	1	1
4	0.987711000	45.233.86.64	78.246.224.199	TCP	64	49152 → 0 [ACK] Seq=1 Ack=2 Win=1026 Len=0	0	1	2
5	1.117359000	45.233.86.64	45.233.86.148	UDP	66	49152 → 49152 Len=20			
6	1.878696999	220.17.198.119	45.233.86.64	TCP	78	0 → 49152 [PSH, ACK] Seq=1 Ack=1 Win=525312 Len=20	20	1	1
7	1.881396000	45.233.86.64	220.17.198.119	TCP	78	49152 → 0 [PSH, ACK] Seq=1 Ack=2 Win=1028 Len=20	20	1	2
8	1.979684000	220.17.198.119	45.233.86.64	TCP	78	[TCP Retransmission] 0 → 49152 [PSH, ACK] Seq=1 Ack=1 Win=525312 L…	20	1	1
9	1.979704000	45.233.86.64	220.17.198.119	TCP	70	49152 → 0 [ACK] Seq=2 Ack=2 Win=1028 Len=0 SLE=1 SRE=1	0	2	2
10	2.067644000	220.17.198.119	45.233.86.64	TCP	64	0 → 49152 [ACK] Seq=2 Ack=2 Win=525312 Len=0	0	2	2
11	2.106987999	45.233.86.64	45.233.86.148	UDP	66	49152 → 49152 Len=20			
12	2.337913000	45.233.86.64	45.233.86.58	UDP	66	49152 → 0 Len=20			
13	2.368769000	45.233.86.64	45.233.86.58	UDP	66	49152 → 0 Len=20			
14	2.496643000	45.233.86.64	220.85.242.130	TCP	78	49152 → 0 [PSH, ACK] Seq=1 Ack=1 Win=1027 Len=20	20	1	1
15	2.540294000	45.233.86.58	45.233.86.64	UDP	66	0 → 49152 Len=20			
16	2.540368000	45.233.86.58	45.233.86.64	UDP	66	0 → 49152 Len=20			
17	2.540713000	45.233.86.64	196.58.73.218	TCP	70	49152 → 0 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1	0	0	0
18	2.588524999	45.233.86.64	196.58.73.218	TCP	70	[TCP Retransmission] 49152 → 0 [SYN] Seq=0 Win=64240 Len=0 MSS=146…	0	0	0
19	2.663645000	220.85.242.130	45.233.86.64	TCP	78	0 → 49152 [PSH, ACK] Seq=1 Ack=2 Win=422 Len=20	20	1	2
20	2.664607000	220.85.242.130	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=2 Ack=2 Win=422 Len=20	20	2	2
21	2.664621000	45.233.86.64	220.85.242.130	TCP	64	49152 → 0 [ACK] Seq=2 Ack=3 Win=1026 Len=0	0	2	3
22	2.685641000	196.58.73.218	45.233.86.64	TCP	70	0 → 49152 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1416 WS=256 S…	0	0	1
23	2.685681000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=1 Ack=1 Win=263168 Len=0	0	1	1
24	2.685927000	45.233.86.64	196.58.73.218	TCP	78	49152 → 0 [PSH, ACK] Seq=1 Ack=1 Win=263168 Len=20	20	1	1
25	2.727791000	196.58.73.218	45.233.86.64	TCP	70	[TCP Out-Of-Order] 0 → 49152 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=…	0	0	1
26	2.727836000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=1 Ack=1 Win=263168 Len=0	0	1	1
27	2.728093000	45.233.86.64	196.58.73.218	TCP	78	[TCP Retransmission] 49152 → 0 [PSH, ACK] Seq=1 Ack=1 Win=263168 L…	20	1	1
28	2.838733000	196.58.73.218	45.233.86.64	TCP	64	0 → 49152 [ACK] Seq=1 Ack=2 Win=524800 Len=0	0	1	2
29	2.840781000	196.58.73.218	45.233.86.64	TCP	78	0 → 49152 [ACK] Seq=1 Ack=2 Win=524800 Len=20	20	1	2
30	2.840804000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [ACK] Seq=2 Ack=2 Win=524800 Len=20	20	2	2
31	2.840818000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=2 Ack=3 Win=263168 Len=0	0	2	3
32	2.841755000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [ACK] Seq=3 Ack=2 Win=524800 Len=20	20	3	2
33	2.841755000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=4 Ack=2 Win=524800 Len…	20	4	2
34	2.841771999	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=2 Ack=5 Win=263168 Len=0	0	2	5
35	2.848728000	45.233.86.64	196.58.73.218	TCP	78	[TCP Retransmission] 49152 → 0 [PSH, ACK] Seq=2 Ack=5 Win=263168 L…	20	2	5
36	2.848862000	45.233.86.64	196.58.73.218	TCP	78	[TCP Out-Of-Order] 49152 → 0 [PSH, ACK] Seq=3 Ack=5 Win=263168 Len…	20	3	5
37	2.848992000	45.233.86.64	196.58.73.218	TCP	78	[TCP Out-Of-Order] 49152 → 0 [PSH, ACK] Seq=4 Ack=5 Win=263168 Len…	20	4	5
38	2.888800000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [ACK] Seq=1 Ack=2 Win=524800 Len=20	20	1	2
39	2.888801000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [ACK] Seq=2 Ack=2 Win=524800 Len=20	20	2	2
40	2.888831000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=2 Ack=3 Win=263168 Len=0	0	2	3
41	2.889694000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [ACK] Seq=3 Ack=2 Win=524800 Len=20	20	3	2
42	2.889694000	196.58.73.218	45.233.86.64	TCP	78	[TCP Retransmission] 0 → 49152 [PSH, ACK] Seq=4 Ack=2 Win=524800 L…	20	4	2
43	2.889710000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=2 Ack=5 Win=263168 Len=0	0	2	5
44	2.896532000	45.233.86.64	196.58.73.218	TCP	78	[TCP Out-Of-Order] 49152 → 0 [PSH, ACK] Seq=2 Ack=5 Win=263168 Len…	20	2	5
45	2.992728999	196.58.73.218	45.233.86.64	TCP	78	[TCP Retransmission] 0 → 49152 [PSH, ACK] Seq=5 Ack=5 Win=524544 L…	20	5	5
46	2.992730000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=6 Ack=5 Win=524544 Len…	20	6	5
47	2.992766999	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=5 Ack=5 Win=262912 Len=0	0	5	5
48	2.992891000	45.233.86.64	196.58.73.218	TCP	78	[TCP Retransmission] 49152 → 0 [PSH, ACK] Seq=5 Ack=5 Win=262912 L…	20	5	5
49	3.015602000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=7 Ack=5 Win=524032 Len…	20	7	5
50	3.016656000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=8 Ack=5 Win=524032 Len…	20	8	5
51	3.016675000	45.233.86.64	196.58.73.218	TCP	64	49152 → 0 [ACK] Seq=6 Ack=9 Win=262400 Len=0	0	6	9
52	3.018283000	45.233.86.64	196.58.73.218	TCP	78	[TCP Out-Of-Order] 49152 → 0 [PSH, ACK] Seq=6 Ack=9 Win=262400 Len…	20	6	9
53	3.043671000	196.58.73.218	45.233.86.64	TCP	78	[TCP Out-Of-Order] 0 → 49152 [PSH, ACK] Seq=5 Ack=3 Win=524544 Len…	20	5	3

Figure 17: First 53 packets of Anonymized Wireshark Capture

10 Conclusion

Real network data is vital for network research, education, advancing network design, maintaining secure and reliable networks, and the evaluation and development of security mechanisms [71, 56, 82, 37]. This data can be contained in various logs, such as Intrusion Detection System (IDS) logs, or packet captures or traces which contain packets captured from an interface within the network. Packet traces often contain information about individuals, enterprises and the networks themselves that should not be disclosed. In order to properly protect sensitive information in these traces while also maintaining a reasonable amount of utility it is important to be able to anonymize various aspects of the traces.

The aim of this thesis was to develop a network packet capture anonymization tool. In order to achieve this, the problem was divided into four tasks:

1. Identifying features the tool needs to support
2. Identifying protocols the tool needs to support
3. Developing an anonymization tool - Shanon
4. Evaluating the developed tool

In the process of completing these tasks this thesis makes multiple contributions to the field of network packet trace anonymization.

The first contribution is a review of literature in the field of network packet trace anonymization which includes existing tools and frameworks, standards, and relevant US and EU law: the ECPA and GDPR.

Shanon's features and protocols are based on features and protocols identified as part of the study of existing work. Identifying these features led to the creation of a taxonomy of features anonymization tools should possess in order to adapt to the needs of a wider range of users. The taxonomy is based on features present in three frameworks: FLAIM [86], developed by Slagell et al., tcpmkpub [72], developed by Pang et al. and PktAnon [41] developed by Gamer et al. It can be found in Chapter 3, and is the second contribution this thesis makes to its field. The protocols Shanon supports are based on protocols supported by existing tools and frameworks. These can be seen in Table 2. Additionally ICMPv6 and NDP are included as the IPv6 counterparts of ICMP and ARP.

The final two contributions are Shanon, a flexible, policy-based anonymization tool written in Lua that plugs into the Wireshark network protocol analyzer, and `libAnonLua`, a Lua library containing reusable network proto-

col anonymization primitives, as well as additional utility functions and constants.

Anonymization is a difficult problem to tackle. It requires attention to details and consideration of the information present not just in a protocol field, but in a combination of protocol fields, even after some or all of them have been anonymized, as well as how this information can be used to defeat it. The attack surface is broad, as can be seen in Section 2.4. Anonymizing in a way that protects from these threats also inevitably leads to a significant loss of information. Some of the effects of this can be seen in Chapter 9.

As a result of this, some have argued that anonymization is not sufficient or the right approach [63, 65]. Somewhat reluctantly, considering the large amount of work that went into this thesis, we must acknowledge they have a point. Sufficiently strong anonymization causes such data losses that the utility of a capture after this anonymization has been executed is questionable. One example of this is the difficulty of anonymizing TCP sequence and acknowledgement numbers in a consistent way without maintaining a prohibitively large amount of state information, as discussed in Section 8.3.

Using Wireshark as a basis for Shanon offers a solution to part of the anonymization problem. Wireshark processes capture files and analyzes protocols, allowing Shanon to focus on retrieving the protocol fields and anonymizing them. However, Wireshark is not without its own issues. Wireshark documentation does not offer sufficient information for new users to fully understand how to work with the provided Lua API without spending a large amount of time on trial and error. The documentation seems to be more suited as a reminder to those already familiar with the subject matter than an entry point for new users. This creates an entry barrier for new users, and also results in a loss of time when first developing scripts intended to run in Wireshark. Lua as a language is also not the best choice for a tool such as this, however the alternative, writing the tool in C, would require compiling Wireshark with the tool in order for the tool to work, which would have made the tool less portable and debugging during development slower. It would potentially result in an increase in performance, however, and is therefore worth considering for future work.

The number of chosen protocols for this work was too large. Looking back, it would have been more reasonable to choose a smaller set of protocols as an initial choice. A significant portion of work was dedicated to retrieving all the fields of the anonymized protocols. The amount of work was further increased due to difficulties with retrieving optional fields described in Section 8.3. The result of the large number of chosen protocols was a large amount of work and time necessary to complete Shanon and

this thesis.

Despite the difficulty of the problem, the aim of this thesis has been achieved. With the exception of recoverability, which is difficult to implement as well as a potential security risk, Shanon possesses all the features defined in Chapter 3 as shown in Chapter 8. Chapter 9 demonstrates some of Shanon's capabilities on a real-world capture, showing that anonymization was achieved for the protocols present in that capture.

11 Future Work

In the future, Shanon could be improved in multiple ways. When it comes to performance, Shanon achieved only 2075 packets per second, which is roughly 13% of what tcpmkpub can do. This perfomance could be improved by coding a future version in C instead of Lua, but would potentially come at the cost of portability.

Metadata could be improved as well. Currently only the information that anonymization was carried out by Shanon, and original TCP and UDP stream numbers, can be preserved. This is done using packet comments which can result in Wireshark becoming unresponsive when viewing certain statistics for a capture with many packets. In the future, metadata could be expanded to include more information. Instead of packet comments, a custom block in the pcapng filesystem could be used, or a custom option. These would require registering a PEN with IANA, however.

Shanon currently does not support exchanging information between anonymizers for individual protocols. Each anonymizer is run independently. While this makes extending Shanon with new protocols simple as there is no need to handle the complexity of different possible protocols at different layers in each anonymizer, it also limits the flexibility of anonymization policies. It is currently impossible to anonymize protocols differently depending on external context. For example, anonymizing TCP differently for different subnets. The TCP anonymizer is completely unaware of any protocol other than TCP. Improving this would yield the ability to create more intricate policies which better thread the fine line between retaining information for research utility and removing information which should not make it into a capture.

Finally, new anonymizers for protocols which are currently not supported by Shanon could be written, expanding its capabilities.

Bibliography

[1] S. Amante et al. *IPv6 Flow Label Specification*. RFC 6437. RFC Editor, 2011-11. URL: https://www.rfc-editor.org/rfc/rfc6437.html.

[2] J. Arkko, M. Cotton, and L. Vegoda. *IPv4 Address Blocks Reserved for Documentation*. RFC 5737. RFC Editor, 2010-01. URL: https://www.rfc-editor.org/rfc/rfc5737.html.

[3] John Bethencourt, Jason Franklin, and Mary K Vernon. "Mapping Internet Sensors with Probe Response Attacks." In: *USENIX Security Symposium*. 2005, pp. 193–208.

[4] Jasper Bongertz. *TraceWrangler Manual*. 2016.

[5] D. Borman et al. *TCP Extensions for High Performance*. RFC 7323. RFC Editor, 2014-09. URL: https://www.rfc-editor.org/rfc/rfc7323.html.

[6] Robert Braden. *Requirements for Internet Hosts - Communication Layers*. STD 3. RFC Editor, 1989-10. URL: http://www.rfc-editor.org/rfc/rfc1122.txt.

[7] Scott Bradner and Jim McQuaid. *Benchmarking Methodology for Network Interconnect Devices*. RFC 2544. RFC Editor, 1999-03. URL: http://www.rfc-editor.org/rfc/rfc2544.txt.

[8] Tønnes Brekne and André Årnes. "Circumventing IP-address pseudonymization." In: *Communications and Computer Networks*. 2005, pp. 43–48.

[9] Martin Burkhart et al. "The Risk-Utility Tradeoff for IP Address Truncation". In: *Proceedings of the 1st ACM Workshop on Network Data Anonymization*. NDA '08. Alexandria, Virginia, USA: Association for Computing Machinery, 2008, pp. 23–30. ISBN: 9781605583013. DOI: 10.1145/1456441.1456452. URL: https://doi.org/10.1145/1456441.1456452.

[10] Martin Burkhart et al. “The role of network trace anonymization under attack”. eng. In: *ACM SIGCOMM Computer Communication Review* 40.1 (2010), pp. 5–11. ISSN: 0146-4833.

[11] Aaron J Burstein. “An Uneasy Relationship: Cyber Security Information Sharing, Communications Privacy, and the Boundaries of the Firm.” In: *WEIS*. 2007.

[12] Caida. *The CAIDA UCSD Trace Statistics for CAIDA Passive OC48 and OC192 Traces website*. URL: https://www.caida.org/data/passive/trace_stats/.

[13] B. Carpenter and K. Moore. *Connection of IPv6 Domains via IPv4 Clouds*. RFC 3056. RFC Editor, 2001-02. URL: https://www.rfc-editor.org/rfc/rfc3056.html.

[14] S. Cheshire, B. Aboba, and E. Guttman. *Dynamic Configuration of IPv4 Link-Local Addresses*. RFC 3927. RFC Editor, 2005-05. URL: https://www.rfc-editor.org/rfc/rfc3927.html.

[15] Cisco. *Snort IDS*. 2019. URL: https://www.snort.org/ (visited on 2019-04-08).

[16] M.P. Collins and M.K. Reiter. “Finding peer-to-peer file-sharing using coarse network behaviors”. In: vol. 4189. Springer Verlag, 2006, pp. 1–17. ISBN: 354044601X.

[17] A. Conta, S. Deering, and M. Gupta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443. RFC Editor, 2006-03. URL: http://www.rfc-editor.org/rfc/rfc4443.txt.

[18] Wireshark Contributors. *Wireshark Documentation: Functions For Handling Packet Data*. 2020. URL: https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Tvb.html (visited on 2020-08-13).

[19] Wireshark Contributors. *Wireshark Documentation: Obtaining Dissection Data*. 2020. URL: https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Field.html (visited on 2020-08-13).

[20] Wireshark Contributors. *Wireshark Documentation: Post-Dissection Packet Analysis*. 2020. URL: https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Listener.html (visited on 2020-08-11).

[21] Wireshark Contributors. *Wireshark Wiki Lua Page*. 2020. URL: https://gitlab.com/wireshark/wireshark/-/wikis/Lua/ (visited on 2020-08-13).

[22] Wireshark Contributors. *Wireshark Wiki Lua/Taps Page*. 2012. URL: https://gitlab.com/wireshark/wireshark/-/wikis/Lua/Taps (visited on 2020-08-13).

[23] Wireshark Contributors. *Wireshark Wiki Performance Page*. 2013. URL: https://wiki.wireshark.org/Performance (visited on 2019-04-08).

[24] Wireshark Contributors. *Wireshark Wiki WLAN (IEEE 802.11) Capture Setup Page*. 2020. URL: https://wiki.wireshark.org/CaptureSetup/WLAN (visited on 2020-08-19).

[25] M. Cotton, L. Vegoda, and D. Meyer. *IANA Guidelines for IPv4 Multicast Address Assignments*. BCP 51. RFC Editor, 2010-03. URL: https://www.rfc-editor.org/rfc/rfc5771.html.

[26] M. Cotton et al. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. BCP 165. RFC Editor, 2011-08. URL: https://www.rfc-editor.org/rfc/rfc6335.html.

[27] Scott E. Coull et al. "Playing Devil's Advocate: Inferring Sensitive Information from Anonymized Network Traces". In: *in Proceedings of the Network and Distributed System Security Symposium*. 2007, pp. 35–47.

[28] Scott E Coull et al. "Taming the devil: Techniques for evaluating anonymized network data". In: (2008).

[29] M. Culig. *libAnonLua*. URL: https://github.com/mculig/libAnonLua.

[30] M. Culig. *Shanon*. URL: https://github.com/mculig/Shanon.

[31] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. STD 86. RFC Editor, 2017-07. URL: https://www.rfc-editor.org/rfc/rfc8200.html.

[32] Stephen E. Deering. *ICMP Router Discovery Messages*. RFC 1256. RFC Editor, 1991-09. URL: http://www.rfc-editor.org/rfc/rfc1256.txt.

[33] Steve Deering. *Host extensions for IP multicasting*. STD 5. RFC Editor, 1989-08. URL: http://www.rfc-editor.org/rfc/rfc1112.txt.

[34] Lua Developers. *Lua About Page*. URL: https://www.lua.org/about.html (visited on 2020-08-19).

[35] Tcpdump/libpcap devs. *Tcpdump/libpcap*. 2019. URL: https://www.tcpdump.org/ (visited on 2019-04-08).

[36] J.P Early, C.E Brodley, and C Rosenberg. “Behavioral authentication of server flows”. eng. In: *19th Annual Computer Security Applications Conference, 2003. Proceedings*. Vol. 2003-. IEEE, 2003, pp. 46–55. ISBN: 0769520413.

[37] Sonia Fahmy and Christine Tan. *Balancing Privacy and Fidelity in Packet Traces for Security Evaluation*. Tech. rep. Tech. Rep. CSD-04-034, Purdue Univ, 2004.

[38] Jinliang Fan et al. “Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme”. eng. In: *Computer Networks* 46.2 (2004), pp. 253–272. ISSN: 1389-1286.

[39] Rodrigo Fonseca et al. *IP Options are not an option*. Tech. rep. UCB/EECS-2005-24. EECS Department, University of California, Berkeley, 2005-12. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-24.html.

[40] Wireshark Foundation. *Wireshark*. 2019. URL: https://www.wireshark.org/ (visited on 2019-04-08).

[41] Th. Gamer, Chr. Mayer, and M. Schöller. “PktAnon - A Generic Framework for Profile-based Traffic Anonymization”. In: *PIK - Praxis der Informationsverarbeitung und Kommunikation* 31.2 (2008), pp. 76–81. ISSN: 0930-5157.

[42] F. Gont. *Deprecation of ICMP Source Quench Messages*. RFC 6633. RFC Editor, 2012-05. URL: http://www.rfc-editor.org/rfc/rfc6633.txt.

[43] F. Gont and C. Pignataro. *Formally Deprecating Some ICMPv4 Message Types*. RFC 6918. RFC Editor, 2013-04. URL: http://www.rfc-editor.org/rfc/rfc6918.txt.

[44] Google. *Google IPv6 adoption statistics*. 2019. URL: https://www.google.com/intl/en/ipv6/statistics.html (visited on 2019-04-08).

[45] Jeremi M Gosney. *Nvidia GTX 1080 Ti Hashcat Benchmarks*. 2017. URL: https://gist.github.com/epixoip/973da7352f4cc005746c627527e4d073 (visited on 2019-05-10).

[46] Marc Heuse. *thc-ipv6*. URL: https://github.com/vanhauser-thc/thc-ipv6.

[47] C. Huitema. *An Anycast Prefix for 6to4 Relay Routers*. RFC 3068. RFC Editor, 2001-06. URL: https://www.rfc-editor.org/rfc/rfc3068.html.

[48] C. Huitema. *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)*. RFC 4380. RFC Editor, 2006-02. URL: https://www.rfc-editor.org/rfc/rfc4380.html.

[49] IANA. *IANA ARP Parameters Registry*. URL: https://www.iana.org/assignments/arp-parameters/arp-parameters.xml (visited on 2020-08-19).

[50] IANA. *IANA PEN Application Forms*. 2019. URL: https://pen.iana.org/pen/ (visited on 2019-04-08).

[51] IANA. *IANA Protocol Numbers Registry*. URL: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml (visited on 2020-08-19).

[52] *IEEE Standard for Ethernet (802.3-2018)*. eng. New York, USA: IEEE, 2018, pp. 1–5600. ISBN: 9781504450904.

[53] "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture". In: *IEEE Std 802-2014 (Revision to IEEE Std 802-2001)* (2014), pp. 1–74.

[54] European Court of Justice. *Patrick Breyer vs Bundesrepublik Deutschland*. 2016. URL: http://curia.europa.eu/juris/document/document.jsf?docid=184668&doclang=EN (visited on 2019-03-04).

[55] T Karagiannis, K Papagiannaki, and M Faloutsos. "BLINC: Multilevel traffic classification in the dark". English. In: *Acm Sigcomm Computer Communication Review* 35.4 (2005), pp. 229–240. ISSN: 0146-4833.

[56] Justin King, Kiran Lakkaraju, and Adam Slagell. "A taxonomy and adversarial model for attacks against network log anonymization". eng. In: *Proceedings of the 2009 ACM symposium on applied computing*. SAC '09. ACM, 2009, pp. 1286–1293. ISBN: 9781605581668.

[57] T Kohno, A Broido, and K.C Claffy. "Remote physical device fingerprinting". eng. In: *IEEE Transactions on Dependable and Secure Computing* 2.2 (2005), pp. 93–108. ISSN: 1545-5971.

[58] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. 6th. Pearson, 2012. ISBN: 0132856204.

[59] J. Laganier and F. Dupont. *An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers Version 2 (ORCHIDv2)*. RFC 7343. RFC Editor, 2014-09. URL: https://www.rfc-editor.org/rfc/rfc7343.html.

[60] Gordon Lyon. *Nping - Network packet generation tool / ping utiliy*. URL: https://nmap.org/nping/.

[61] Ed. M. Tuexen et al. *PCAP Next Generation (pcapng) Capture File Format*. 2019. URL: https://github.com/pcapng/pcapng (visited on 2019-04-07).

[62] Greg Minshall. *Tcpdpriv BSD Man Page*. 1996. URL: http://fly.isti.cnr.it/software/tcpdpriv/tcpdpriv.0.txt (visited on 2019-03-02).

[63] J. Mirkovic. "Privacy-safe network trace sharing via secure queries". In: 2008, pp. 3–10. ISBN: 9781605583013.

[64] Jeffrey Mogul. *Broadcasting Internet Datagrams*. STD 5. RFC Editor, 1984-10. URL: http://www.rfc-editor.org/rfc/rfc919.txt.

[65] Jeffrey Mogul and Martin Arlitt. "SC2D: an alternative to trace anonymization". eng. In: *Proceedings of the 2006 SIGCOMM workshop on mining network data*. Vol. 2006. MineNet '06. ACM, 2006, pp. 323–328. ISBN: 159593569X.

[66] T. Narten et al. *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861. RFC Editor, 2007-09. URL: http://www.rfc-editor.org/rfc/rfc4861.txt.

[67] RIPE NCC. *IPv6 Address Types*. 2019. URL: https://www.ripe.net/ipv6-address-types (visited on 2021-01-24).

[68] Kathleen Nichols et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474. RFC Editor, 1998-12. URL: http://www.rfc-editor.org/rfc/rfc2474.txt.

[69] Omnipacket. *SafePcap Website*. 2016. URL: https://omnipacket.com/safepcap (visited on 2019-02-27).

[70] Omnipacket. *WireEdit website*. 2014. URL: https://wireedit.com/ (visited on 2019-02-27).

[71] RM Pang and V Paxson. "A high-level programming environment for packet trace anonymization and transformation". English. In: *Acm Sigcomm Computer Communication Review* 33.4 (2003), pp. 339–351. ISSN: 0146-4833.

[72] Ruoming Pang et al. “The devil and packet trace anonymization”. eng. In: *ACM SIGCOMM Computer Communication Review* 36.1 (2006), pp. 29–38. ISSN: 0146-4833.

[73] European Parliament. *General Data Protection Regulation*. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (visited on 2019-03-04).

[74] V. Paxson. “Strategies for sound internet measurement”. In: *Proceedings of the 2004 ACM SIGCOMM Internet Measurement Conference, IMC 2004*. 2004, pp. 263–271. ISBN: 1581138210.

[75] David C. Plummer. *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. STD 37. RFC Editor, 1982-11. URL: http://www.rfc-editor.org/rfc/rfc826.txt.

[76] J. Postel. *Internet Control Message Protocol*. STD 5. RFC Editor, 1981-09. URL: http://www.rfc-editor.org/rfc/rfc792.txt.

[77] J. Postel. *User Datagram Protocol*. STD 6. RFC Editor, 1980-08. URL: http://www.rfc-editor.org/rfc/rfc768.txt.

[78] Jon Postel. *Internet Protocol*. STD 5. RFC Editor, 1981-09. URL: http://www.rfc-editor.org/rfc/rfc791.txt.

[79] Jon Postel. *Transmission Control Protocol*. STD 7. RFC Editor, 1981-09. URL: http://www.rfc-editor.org/rfc/rfc793.txt.

[80] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. RFC Editor, 2001-09. URL: http://www.rfc-editor.org/rfc/rfc3168.txt.

[81] Yakov Rekhter et al. *Address Allocation for Private Internets*. BCP 5. RFC Editor, 1996-02. URL: http://www.rfc-editor.org/rfc/rfc1918.txt.

[82] Bruno Ribeiro et al. “Analyzing Privacy in Enterprise Packet Trace Anonymization”. In: *In Proceedings of the 15 th Network and Distributed Systems Security Symposium*. 2008.

[83] Thomas Gamer Roland Bless Christoph P. Mayer. *PktAnon v1.4.0-dev Source Code*. URL: https://www.tm.uka.de/software/pktanon/download/index.html.

[84] Stavros Shiaeles and Maria Papadaki. “FHSD: An improved IP spoof detection method for web DDoS attacks”. In: *The Computer Journal* 58 (2014-02). DOI: 10.1093/comjnl/bxu007.

[85] Adam J. Slagell and William Yurcik. “Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization”. In: *CoRR* cs.CR/0409005 (2004). URL: http://arxiv.org/abs/cs.CR/0409005.

[86] Adam Slagell, Kiran Lakkaraju, and Katherine Luo. “FLAIM: A Multilevel Anonymization Framework for Computer and Network Logs”. In: *Computing Research Repository - CORR* (2006-01), pp. 63–77.

[87] Matthew Smart, G Robert Malan, and Farnam Jahanian. “Defeating TCP/IP Stack Fingerprinting.” In: *Usenix Security Symposium*. 2000.

[88] Dug Song and Contributors. *dpkt*. URL: https://dpkt.readthedocs.io/en/latest/index.html.

[89] N. Spring, D. Wetherall, and D. Ely. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*. RFC 3540. RFC Editor, 2003-06. URL: https://www.rfc-editor.org/rfc/rfc3540.html.

[90] United States. *18 U.S.C. §2258E*. 2019. URL: https://www.law.cornell.edu/uscode/text/18/2258E#6 (visited on 2019-04-07).

[91] United States. *18 U.S.C. §2510*. 2019. URL: https://www.law.cornell.edu/uscode/text/18/2510 (visited on 2019-04-07).

[92] United States. *18 U.S.C. §2702*. 2019. URL: https://www.law.cornell.edu/uscode/text/18/2702 (visited on 2019-04-07).

[93] United States. *18 U.S.C. §3121*. 2019. URL: https://www.law.cornell.edu/uscode/text/18/3121 (visited on 2019-04-07).

[94] United States. *18 U.S.C. §3127*. 2019. URL: https://www.law.cornell.edu/uscode/text/18/3127 (visited on 2019-04-07).

[95] Latanya Sweeney. “k-anonymity: A model for protecting privacy”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[96] J. Touch. *Updated Specification of the IPv4 ID Field*. RFC 6864. RFC Editor, 2013-02. URL: http://www.rfc-editor.org/rfc/rfc6864.txt.

[97] O. Troan and B. Carpenter. *Deprecating the Anycast Prefix for 6to4 Relay Routers*. BCP 196. RFC Editor, 2015-05. URL: https://www.rfc-editor.org/rfc/rfc7526.html.

[98] Alexandria Division. United States District Court E.D. Virginia. *Freedman v. America Online, Inc., 329 F. Supp. 2d 745 (E.D. Va. 2004)*. 2004. URL: https://law.justia.com/cases/federal/district-courts/FSupp2/329/745/2409876/ (visited on 2019-04-07).

[99] J. Weil et al. *IANA-Reserved IPv4 Prefix for Shared Address Space*. BCP 153. RFC Editor, 2012-04. URL: http://www.rfc-editor.org/rfc/rfc6598.txt.

[100] William Yurcik et al. “SCRUB-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs”. eng. In: *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007*. IEEE, 2007, pp. 49–56. ISBN: 9781424409747.