Fall 12-7-2018

The Nature of Ephemeral Secrets in Reverse Engineering Tasks

Antonio Miguel Espinoza
University of New Mexico

Antonio M. Espinoza

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Jedidiah R. Crandall , Chairperson

Stephanie Forrest

Abdullah A. Mueen

Marie J. Vasek

The Nature of Ephemeral Secrets in Reverse Engineering Tasks

by

Antonio M. Espinoza

B.S., Computer Science, University of New Mexico, 2011
M.S., Computer Science, University of New Mexico, 2015

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

Spring 2019

Acknowledgments

I would like to thank my friend and advisor Professor Jedidiah R. Crandall for getting me started in research as an undergrad and showing me early on how interesting and fun research is. I would also like to thank my dissertation committee, Professors Stephanie Forrest, Abdullah A. Mueen, and Marie J. Vasek for their invaluable advice and feedback on my dissertation.

I would like to thank my loving and patient wife, who has been with me throughout my journey towards my Ph.D, for all of her support.

Finally I would like to thank my mother for showing me, by example, how valuable education is.

The Nature of Ephemeral Secrets in Reverse Engineering Tasks

by

Antonio M. Espinoza

B.S., Computer Science, University of New Mexico, 2011
M.S., Computer Science, University of New Mexico, 2015

Ph.D, Computer Science, University of New Mexico, 2019

Abstract

Reverse engineering is typically carried out on static binary objects, such as files or compiled programs. Often the goal of reverse engineering is to extract a secret that is ephemeral and only exists while the system is running. **My thesis:** Automation and dynamic analysis enable reverse engineers to extract *ephemeral secrets* from *dynamic systems*, obviating the need for analyzing static artifacts such as executable binaries.

I support this thesis through four automated reverse engineering efforts: (1) named entity extraction to track Chinese Internet censorship based on keywords; (2) dynamic information flow tracking to locate secret keys in memory for a live program; (3) man-in-the-middle to emulate server behavior for extracting cryptographic secrets; and, (4) large-scale measurement and data mining of TCP/IP handshake behaviors to reveal machines on the Internet vulnerable to TCP/IP hijacking and other attacks.

In each of these cases, automation enables the extraction of ephemeral secrets,

often in situations where there is no accessible static binary object containing the secret. Furthermore, each project was contingent on building an automated system that interacted with the dynamic system in order to extract the secret(s). This general approach provides a new perspective, increasing the types of systems that can be reverse engineered and provides a promising direction for the future of reverse engineering.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reverse engineering is a dual-use tool that has many applications and is not inherently what one might consider 'good' or 'bad'. Malicious uses of reverse engineering include: reverse engineering a competitor's software to uncover trade secrets, or discovering a flaw in a server to create an exploit to extract private user data. Conversely, reverse engineering can be used non-maliciously. For example, one may independently verify cryptography in a program to ensure that the network communication of the program is safe, or even aid in preserving the rights of an end user by discovering client side censorship and surveillance in a chat program. Reverse engineering is an essential tool in assessing censorship and privacy issues in a system. Currently the extent of censorship on the Internet is difficult to measure and without reverse engineering it would be impossible to uncover the extent of censorship or any other privacy issue.

The traditional view of reverse engineering is that of an engineer meticulously inspecting a program's executable binary (or some other binary object) in order to learn a specific secret for which the engineer is searching. However, many of today's systems require obtaining *ephemeral secrets* that may only exist while the system is

running, or from binary code that the engineer has no access to. Often the ephemeral secret is located on a remote machine a reverse engineer has no access to. In order to deal with these types of dynamic systems, automation is key.

By automating interactions with dynamic portions of a system, I show how to reverse engineer different types of systems and move beyond reverse engineering static binaries. This method allows me to reverse engineer different types of ephemeral secrets, such as the words that are censored by the Great Firewall of China (GFC), or the ephemeral key required to perform attacks on live chat applications which require an attacker to have the perspective of the application's server. Both of these tasks are interesting in that I do not have everything a traditional reverse engineer is accustomed to having. When reverse engineering words filtered by the GFC, I had no access to any of the code, but instead interacted with the system automatically to find new words and test the validity of old words. In attacking LINE (a mobile chat application) I had only the binary code for the client. Despite this limitation, I developed a method for performing an attack that requires the perspective and privilege of the LINE server, without actually having the perspective and privilege of the LINE server or learning all of LINE's client to server communication protocol.

There are two types of program analysis performed to inspect a code's behavior: static and dynamic. Static analysis is performed only on a program's binary, and can be insufficient for discovering specific secrets. Dynamic analysis still requires the program's binary, however the program is executed which provides a more complete picture of how the code works. Dynamic analysis is helpful for discovering parts of a computation that may only exist when the program is running, what I refer to as *ephemeral secrets* (a term commonly used in cryptography). For example, a traditional reverse engineer might acquire a malware sample and attempt to extract a symmetric key that the program uses to encrypt all traffic before it is sent to a command server. Once the key is obtained, all communication can be decrypted to

and from the malware. In contrast, if that malware were to communicate with the server and determine an ephemeral key to use for communication of that session, traditional static analysis would fail. In this scenario dynamic analysis is required to obtain the ephemeral key each time it is calculated. If I wanted to decrypt all traffic between the malware and its control server, I would need to automate key retrieval in order to obtain each ephemeral key. In this and many other situations, dynamic analysis and automation are fundamental tools for reverse engineering systems where ephemeral secrets exist and obscure the nature of the system's behavior.

Each chapter of this dissertation consists of work that pushes the boundaries of what is traditionally considered *reverse engineering* (Chapters 3, 6) or what is possible through traditional reverse engineering methods (Chapters 4, 5). Table 1.1 describes in detail the boundaries each work pushed to uncover its targeted ephemeral secret. Each Chapter required the utilization of automation and dynamic analysis to extract an ephemeral secret. Chapters 4 and 5 both had ephemeral secrets that were cryptographic keys located in memory. The former detected keys used to encrypt text in programs utilizing standard open source libraries, by applying a process called Dynamic Information Flow Tracking. The later was an ephemeral key (derived during program execution) used by LINE for client-to-server encryption, extracted by means of an ephemeral key dump. Chapters 3 and 6 focus on ephemeral secrets in the domain of Internet measurement. Chapter 3 focuses on a list of censored keywords as the ephemeral secret, discovered by creating a system to automatically discover potentially censored keywords in news articles. Chapter 6 focuses on Initial Sequence Number (ISN) creation algorithms as ephemeral secrets. In the ISN work, although the ephemeral secret is an ephemeral process, to learn about the process I observed and studied an ephemeral object (the ISN) created by the process.

Ch.	Name of Chapter	Contribution
3	Automated Named Entity Extraction for Tracking Censorship of Current Events.	Created a system to automatically extract named entites (people places and orginizations) from news sources and test for censorship events. First to automatically generate and dynamically test potentially censored keywords. **Published in FOCI 2011 [43]**
4	Vector-Based Dynamic Information Flow Tracking.	Created a DIFT system that utilizes vectors as taint marks. First DIFT work to handle address and control dependencies in a general and meaningful way. **Published in ARES 2016 [44]**
5	Analysis of End-to-End Encryption in the LINE Messaging Application.	Developed a new method for reverse engineering, that is resilent to the use of certificate pinning, to view the packets from the perspective of a remote server . **Published in FOCI 2017 [45]**
6	Internet-scale Study of TCP Initial Sequence Numbers.	Scanned and evaluted the algorithmic complexity of initial sequence numbers of IPv4 TCP/IP connections. First Internet-scale study of initial sequence numbers.

Table 1.1: **Table of works.**

1.1 Dissertation Overview

Chapter 2 presents related work, highlighting how my methods for discovering ephemeral secrets extend the state of the art in their respective areas.

Chapter 3 describes a censorship study focused on China. In this chapter I describe two different types of censorship implemented in China, as well as a system to find and track the censorship in real time. The dynamic system is the censorship infrastructure in China, colloquially called the Great Firewall of China (GFC). The ephemeral secret is the list of unknown keywords censored by the GFC. I identifed words that were likely to trigger one of the GFC's censorship mechanisms. I auto-

mated the process [43] using Named Entity Extraction to automically select keywords (people, places, and organizations) from news articles which were then used to probe the GFC for censorship. This work was, to my knowledge, the first dynamic system to *actively* probe a country's network for censorship.

Chapter 4 describes a system that can automatically detect, in a program, regions of memory likely to contain a cryptographic key using a technique called Dynamic Information Flow Tracking (DIFT). In Chapter 4 the dynamic system I reverse engineer is the program analyzed at runtime. The ephemeral secret is the key located in memory. The automated system is the DIFT system I constructed that propagates information with vectors (V-DIFT) [44]. Using the system described in Chapter 4, I was able to automatically detect regions of memory likely to contain cryptographic keys in 24 out of 27 test programs. In this work I was the first to account for address and control dependencies (an open problem in DIFT) in a general way to extract ephemeral information from a program using DIFT.

Chapter 5, focuses on reverse engineering the cryptographic security of LINE, a mobile phone chat application. With knowledge of the system gained through reverse engineering, I describe the implementation of two attacks on LINE's end-to-end cryptography. In Chapter 5 the system being reverse engineered is LINE's end-to-end cryptographic implementation. The ephemeral secret is the symmetric key the client and server derive to communicate with each other. The automated process is the environment implemented to perform the replay attack and attack on the lack of forward secrecy.

Chapter 6 describes the first Internet-scale study of the algorithmic complexity of IPv4 TCP/IP ISNs. In this work the ephemeral secret is the algorithmic complexity of ISNs for hosts on the Internet. The dynamic system is the Internet as a whole, and the automation lies in my scans and analysis. Through this study I was able to identify portions of the Internet susceptible to various blind attacks that take advan-

tage of ISNs that are predictable or easy to guess due to their reduced algorithmic complexity.

Finally Chapter 7 presents my conclusion and explores the kind of work possible when researchers, reverse engineers, and educators consider ephemeral secrets in their works.

Chapter 2

Related Work

the original definition of reverse engineering as defined by Rekoff [109], is the process by which a hardware item is analyzed and documented in order to "create a clone or a surrogate." In the domain of computer science the definition of reverse engineering is extended to include: the process by which a program is analyzed in order to extract a secret (often static) from a binary object. This is the definition of reverse engineering used throughout this dissertation. A secret can be a *method* or an *object*, and is either *static* or *ephemeral*. Figure 2.1 shows the reverse engineering landscape (as it relates to static and ephemeral methods and objects) and the space this dissertation covers in that landscape. As seen in Figure 2.1, traditional reverse engineering techniques are able to uncover static *methods* and *objects*, such as an encryption function or hard-coded key. Ephemeral *methods* are more difficult for traditional reverse engineering techniques, and work is still being done to explore how to deal with them. An example of a typical ephemeral method is one where the code builds a function in memory that is used to decode and run other areas of itself. Reverse engineering techniques that can extract *ephemeral objects* are relatively unexplored in the research community and are the focus of this dissertation.

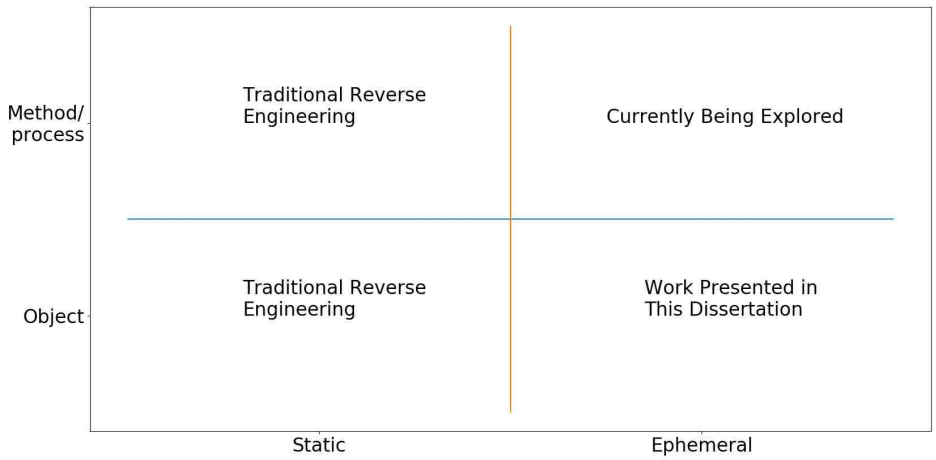


Figure 2.1: **The landscape of reverse engineering as it relates to static and ephemeral secrets of methods and objects.**

Two methods of analysis are used by reverse engineers: static and dynamic analysis. Static analysis is the first type of analysis a reverse engineer typically uses to analyze a binary object, with the aid of a decompiler or disassembler. There are many techniques that fall under the classification of static analysis (searching for specific strings, monitoring what libraries the object links to, *etc.*), however these techniques only help to discover static secrets since the binary object is never executed. In the event that static analysis is insufficient, dynamic analysis is performed. Dynamic analysis requires the binary object to run in whole, or in part, usually with the aid of a debugger. Traditional dynamic analysis is tedious and requires a reverse engineer to discover exactly where the secret is located (requiring direct access to the binary object). This usually requires setting a breakpoint and stepping through a program binary in the same manner a programmer would hunt for a bug in their code until the secret is discovered. Dynamic analysis can be used to uncover both static and ephemeral secrets, however many times it is used to find static secrets. Unlike static secrets, ephemeral secrets can only be discovered through dynamic analysis.

Although analysis is complex, reverse engineering often has a simple goal (to retrieve a secret) which requires the malware or sample to behave the way the reverse engineer requires just once. Once the goal is achieved the malware does not need to be inspected further, as the secret has already been revealed. For example, Knockel *et al.* [74] used this approach to reverse engineer TOM Skype's censorship and surveillance lists for multiple client versions by analyzing its decryption algorithm, and locating the hard coded symmetric key. This work is an example of a larger body of work that focuses around new applications of old reverse engineering techniques (*e.g.*, [117, 67, 118]). Occasionally tools are created to solve classic issues reverse engineers commonly face. For example, PANDA (developed by Dolan-Gavitt *et al.* [39]) is a recent tool that focus on the principle of reproducibility. As stated before, reverse engineers often need the code to behave the right way just once, however the steps followed to achieve the desired behavior are often not reproducible because many programs are non-deterministic. PANDA solves the reproducibility problem created by non-determinism, allowing reverse engineers to share traces, and analyze the same code as it was originally executed. In contrast to recent and traditional work that focuses on extracting one or more static secrets, my work introduces a new paradigm focused on reverse engineering *ephemeral secrets* by applying automation and dynamic analysis.

Extracting ephemeral secrets often yields the same result as classic reverse engineering techniques. However, as developers use increasingly sophisticated anti-reverse-engineering methods, classic reverse engineering techniques become ineffective. For example, in Garman *et al.*'s work [53], the researchers faced a problem similar to the challenge I faced with reverse engineering LINE in Chapter 5. They had the same threat model as mine requiring that the attack be performed from the perspective of iMessage's server without actually having access to iMessage's server. Garman *et al.* approached the problem by forging a TLS certificate. This is a common technique employed by readily available off-the-shelf software such as

Man in the Middle Proxy [92]. This method is dynamic and automated, and it removes the need to find the ephemeral secret (in this case the symmetric key). I also implemented this technique for reverse engineering LINE, but it was ineffective due to LINE's implementation specifics (namely, LINE does not use the OS'es certificate trust authority for all authentication). Instead of TLS certificate forging, an Ephemeral Key Dump (EKD) was performed. In this technique that I developed as part of this dissertation, once the client and server agree on a shared symmetric key, the key is automatically dumped to a location the reverse engineer can access. Because the key is symmetric, the key the client obtains is the same as the server's and all traffic can be decrypted to appear as it would on the server, yielding the same result as TLS certificate forging.

Reverse engineering ephemeral secrets has many applications, one of the most exciting is its use in studying Internet censorship [89, 69, 46]. Studying censorship is particularly difficult, because there is often no binary available to the reverse engineer and the censorship mechanism must be tested by measuring other indicators. In addition, the ephemeral secret is often remote, which is problematic for traditional reverse engineering. The work from Chapter 3 laid the groundwork for systems that dynamically measure Internet censorship in China by treating the blacklist as an ephemeral secret. Recent work has begun to automatically find or generate blocklists by treating censorship like an ephemeral secret. Work by Darer *et al.* [35, 36] and later Hounsel *et al.* [64] are examples of work based on the same technique for measuring censorship pioneered in Chapter 3. Work concurrent with Chapter 3, such as Sfakianakis *et al.* [119], relied on user input to generate a group of websites to track and lacked the dynamic and automatic elements required to discover new instances of censorship.

Beyond censorship measurement, the discovery of ephemeral secrets is beneficial to other areas of Internet measurement, such as studying ISNs. ISNs are important

to TCP connections because they offer a layer of security to protect against attacks: *e.g.,* blind reset, blind injection, or blind spoofing attacks. Many implementations of these attacks require the attacker to have access to the ephemeral ISN but typically rely on the victim running malicious code [101, 102, 57, 20]. These attacks have very specific threat models, therefore it is not easy to study the number of machines on the Internet susceptible to their respective attacks. Cao *et al.* [15, 16] were able to exploit a side channel in RFC 5961. The RFC was implemented in the Linux kernel [107] (the kernel was quickly patched) and they were able to do a longitudinal study [104] of machines on the Internet that were vulnerable to their attack over time. My Internet-scale study of ISNs does not rely on a side channel or malicious code, instead it measures the number of machines that lack algorithmic complexity in their ISN generation code across the IPv4 address space on port 80.

My research adds to a body of similar work that demonstrates my thesis. Each one contributes novel ideas or approaches that push the bounds and capabilities of reverse engineering. The work presented in this dissertation is a mixture of both: old methods that exist for other purposes that have been combined together in new ways (Chapters 3 and 5), and new methods for extracting ephemeral secrets that existing methods can not be used to uncover (Chapters 4 and 6).

Chapter 3

Automated Named Entity Extraction for Tracking Censorship of Current Events

Tracking Internet censorship is challenging because what content the censors target can change daily, even hourly, with current events. The process must be automated because of the large amount of data that needs to be processed. My focus in this chapter is on automated probing of keyword-based Internet censorship, where natural language processing techniques are used to generate keywords with which to probe for censorship. In this chapter I present a named entity extraction framework that can extract the names of people, places, and organizations from text such as a news article. Previous efforts to automate the study of keyword-based Internet censorship where based on semantic analysis of existing bodies of text, such as Wikipedia, and so could not extract meaningful keywords from the news with which to probe.

I used a maximum entropy approach for named entity extraction, because of its flexibility. My results suggest that this approach gives good results with only a

rudimentary understanding of the target language. This means that the approach is very flexible, and while my implementation was for Chinese, I anticipate that extending the framework to other languages such as Arabic, Farsi, and Spanish will be straightforward because of the maximum entropy approach. In this chapter I present some testing results as well as results from probing China's GET request censorship and search engine filtering using this framework.

There are many open questions about Internet censorship, including how effective it is, what makes it effective, what kinds of targeted activities it is effective (or is not effective) at stopping, and so forth. A first step toward answering any of these questions is to collect enough data to understand how censorship is applied and what kinds of activities are targeted by the censors. This implies automated probing that is broad and carried out over a long period of time, because censorship within a single country can vary from province to province, company to company, and technology to technology and what content is targeted can change daily, even hourly.

3.0.1 Related Work

My focus in this chapter is on keyword-based Internet censorship, and for the results I present I am interested specifically in China, as it is known to be the "most advanced keyword-based Internet censorship mechanism." [30] Keyword-based Internet censorship in China has been studied by several groups of researchers, but is not well understood. An anonymous government official writing as "Mr. Tao", in a report published by Reporters Without Borders [93], described three types of keywords: masked words (replaced by asterisks), sensitive words (in need of a moderator's approval), and taboo words (words that cannot be used). According to Mr. Tao, a keyword list is produced and updated by the Information Office of the State Council. He adds, "each site adds key-words to its own filters in order not to run the risk of being criticised, punished or, worse still, closed down."

One of the more thoroughly studied forms of keyword-based Internet censorship is GET request filtering at the router level, where GET request packets containing blacklisted keywords cause routers in the backbone of China's Internet to forge reset packets and attempt to reset the TCP connection between the offending client and server. In contrast to HTML response filtering, which appears to have not been effective and may have been discontinued [98], GET request filtering is very effective in terms of the ratio of offending connections that are reset to connections that contain offending text and is still pervasive on China's Internet today. The methods of China's HTTP keyword filtering were first published by the Global Internet Freedom Consortium [54]. Clayton *et al.* [24] published a more detailed study of this mechanism. The ConceptDoppler project [30] found that HTTP keyword filtering in China is not peremptory and is not strictly implemented at the border of the Chinese Internet, with a significant amount of filtering occurring in the backbone. The ConcpetDoppler project also used latent semantic analysis [76] to cluster words from the Chinese-language version of Wikipedia around sensitive concepts and then probe with these potentially sensitive words to see if they are censored by the GET request router-based mechanism. ConceptDoppler initially produced a list of 122 words, and has produced two more lists since.

Software that runs on servers in China, such as blogging software, also implements keyword-based censorship. One snapshot of a blacklist from a blog site is available in a Human Rights Watch Report [65] from 2006, for example. Client-side programs such as chat clients also implement keyword censorship. A blacklist for QQChat is available in the same report [65], and Villeneuve [126] gives a high-level analysis of topics censored by the chat client that is part of TOM-Skype.

Note that all of these lists are one-time-only snapshots. Some of the lists contain different sets of words, suggesting they come from different sources. Furthermore, my results indicate that the HTTP GET request blacklists that are used by routers in

the backbone of China's Internet do not change on a daily, weekly, or even monthly basis. Existing systems that can continuously probe, such as ConceptDoppler, are based on document summary techniques that cluster words based on concept and therefore are not suitable for finding the named entities that are relevant to current events. Such document summary techniques can only compare documents and terms to an existing corpus of text based on the semantics that are latent in term and document frequencies. In contrast, named entity extraction gives additional semantic information about what a document is about based on its use of named entities. Because of the lack of data about Internet censorship and appropriate methods for gathering the data broadly and over a long period of time I have developed a named entity extraction framework, which I present in this chapter.

3.0.2 Structure of the Rest of the Chapter

I discuss the implementation of my framework in Section 3.1. Then I explain my experimental methodology for my results in Section 3.2 followed by the results in Section 3.3 and some concluding remarks.

3.1 Implementation

I implemented a *named entity extraction* (NEE) framework by means of *maximum entropy* (ME) machine learning. Chieu *et al.* [21] define ME as a "frame work [that] estimates probabilities based on the principle of making as few assumptions as possible." Borthwick *et al.* [12] demonstrated that an ME approach to NEE allows for flexibility in the choice of features to train on, since the interactions among features are not as important as they would be in other approaches such as Hidden Markov Models or Maximum Likelihood. I focused on three types of named entities: names

of people, names of places, and names of organizations.

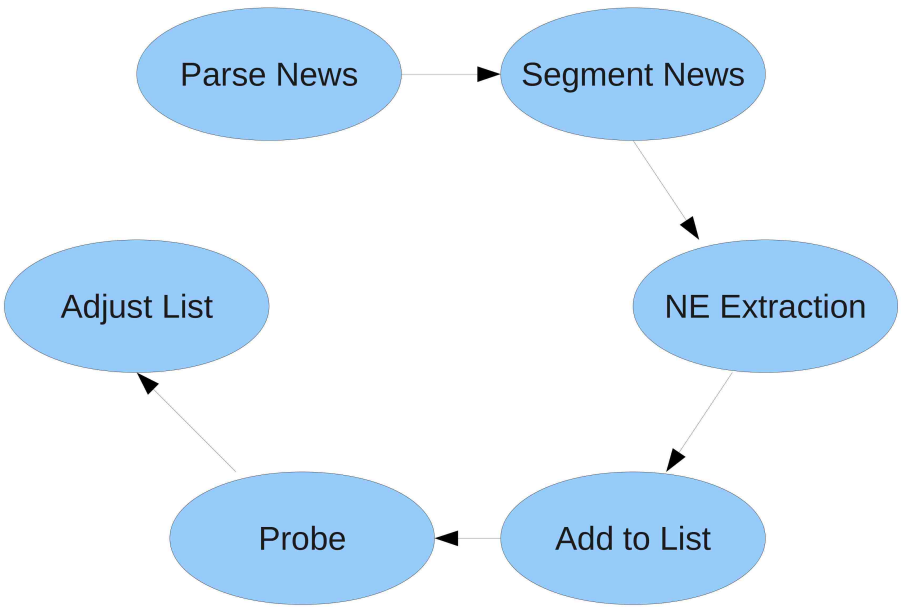


Figure 3.1: **The high-level workflow of my implementation.**

My NEE framework requires a training corpus that has existing *labels*. That is, every word in the training corpus should be labeled with one of four labels: as a name of a person, name of a place, name of an organization, or not any of these types of named entities. The first three labels are then subdivided into complete, beginning, middle or end of the type of label (*e.g.* beginning_person, complete_place *etc.*, are valid labels). This is done so that it is possible for a named entity to span multiple segments after segmentation. I used the Chinese-language version of Wikipedia as my training corpus (192GB), under the assumption that it was a well labeled data set. For example, when people, places, or organizations appear in Wikipedia, the reference is often a link to that person, place, or organization. In addition to the labeled data the ME framework also requires a feature vector for each word in order to build a model of *features* that correlate with particular labels. A feature is a property of the labeled word. One example feature is whether the word contains any

Feature	**Test**
Is place?	Does the word translate to a known place?
Has a name character?	Does the word contain a common name character?
Has punctuation?	Does the word contain any punctuation?
What punctuation (if any)?	What punctuation does the word contain?
Is month?	Does the word contain the character 月?
Has capital letters?	Does the translated word contain roman characters that are capitalized?
Has number?	Does the word consist of only roman numerals?
Has a Chinese number character?	Does the word contain a Chinese number character (一, 二, 三...十).
Has de?	Does the word contain the symbol 的?
Is a dictionary term?	Is the word in a Chinese dictionary?
Parts of Speech	All the parts of speech the translated word has.
Number characters	The number of characters in the word.

Table 3.1: **Features used.**

characters that are common Chinese surnames. Another example feature is if the word is followed by a possessive such as Chinese 的 (de). Table 3.1 lists the features used.

I used several heuristics to treat the Wikipedia corpus as a labeled data set. I assumed the link was a label for a name of a person if the document that was linked to had 年出生 (year of birth), 年逝世 (year of death), or 年逝世人物列表 (year of death person list) among its categories. I assumed the link was a label for a name of a place if the document that was linked to had GPS coordinates associated with it or contained one of the following infoboxes: country, city, cncity (cn=Chinese), prc provence (prc=Peoples Republic of China), or university. Lastly, I assumed the link was a label for an organization if it linked to an article that contained a company or

organization infobox.

In all of the experiments in this chapter, I trained on one third of the Wikipedia corpus using the above labeling scheme, and then tested on a different third of the corpus. For both training and testing, I applied Chinese text segmentation to the entire corpus to divide it into words (because the Chinese written language does not use spaces to divide sentences into words). Then I assigned a feature vector to each word based on the word itself, as well as the word that precedes and follows it. I trained and tested each of the three types of named entities separately.

Using the openNLP toolkit [97], I then assigned conditional probabilities to each word for each sub-label conditioned on its feature vector. Because the probabilities given were for a word being the beginning 'label', middle 'label', end 'label', complete 'label', or not a 'label' (where 'label' is person, place, or organization), I had to find the highest probable legal path through the output. In order to be a legal path sub-labels must be in correct order, for example end 'label' cannot precede middle 'label' legally. Similarly beginning, middle, and end 'label' cannot be surrounded by not 'label' on both sides. In order to accomplish this I used the fact that the output of the openNLP ME toolkit is a directed acyclic graph. Therefore I was able to preform a topological sort to find the highest probable legal path.

For testing or for the actual probing experiments, I take an unlabeled corpus of text (or a test set where the labels are withheld), and then assign a label to each word based on the ME model of the training set. I scale the conditional probabilities in the model linearly so that I get a desirable fraction of labeled words.

See Figure 3.1 for a high-level workflow of my implementation. For probing, I have written parsers for seven Chinese-language news websites (Epoch Times, My China News Digest, Popyard, Radio Free Asia, Sina News, Voice of America, and Wenxuecity). My framework downloaded news from these websites every day from

March 8 to April 8, 2011 and performed named entity extraction based on the model that was created using Wikipedia. For any word that is labeled as a named entity, I include that word in my list of keywords to probe with on that day. My probing infrastructure performs two kinds of probes, it tests twelve servers in China for HTTP GET request filtering based on forged RSTs, and it tests two search engines (baidu.com, and sogou.com) to see if the word elicits a legal message in Chinese stating that entries have been removed from results for the search query. My probing infrastructure has multiple priority levels for each tested keywords, with levels with lower numbers being higher in priority for testing. If a word is ever interpreted to be blacklisted, it is placed in priority 0 so that it will be tested every 12 hours for the remainder of the probing. Words enter the probing infrastructure at level 1. Every 12 hours level 0 words are probed, followed by level 1 words, then level 2, and so on. If a word does not appear to be blacklisted, it is moved down one priority, except if it is in priority 0 in which case it remains in priority 0. There are 15 priorities, with the lowest being 14. After a word has been probed 14 times and does not appear to be censored, it falls off the bottom of the list.

In order to get search engine results that are independent of GET request censorship, I divided GET requests for the two search engines I test against into separate packets that were sent separately to evade GET request filtering, but were reassembled the server. When testing for forged RSTs, I waited at least 100 seconds between each query for any pair of IP addresses, because RSTs disable further communication between client and server for a period of time less than 100 seconds. As a special consideration, the search engine results do not affect the priorities of keywords, because I found this to cause many words that were not actually targets of censorship to be in priority 0. I record a traceroute to each server every hour, so that any major changes in the keyword censorship that might be due to changes in routing can be explained.

3.2 Experimental Methodology

For the results I present in this chapter, there are two experiments that I performed. One experiment is to measure the specificity and recall of the NEE framework on a different third of Wikipedia from the training set. This provides baseline numbers that access how well the NEE framework is performing. The second experiment is a test run of approximately two months (with some downtime) in which I ran the entire NEE and probing framework and obtained some results that are censored topics from the news.

For the first experiment, I focus on specificity instead of precision because of the context of probing with keywords. Precision is the probability that a word labeled as a named entity actually is a named entity. Since there are no human consumers of the output of my NEE framework, precision is not as relevant, and the lack of precision just adds to the number of non-named entities tested. Any word that is not a named entity but is labeled as such (a false positive) will be probed with, perhaps unnecessarily, but this is relatively acceptable compared to missing named entities (false negatives). Thus, recall and specificity are better indicators of performance in my context than recall and precision. Recall is the probability that an actual named entity is labeled as a named entity (true positive). Specificity is the proportion of words that are not named entities that are not labeled as named entities (true negatives). Thus, as long as the specificity remains high enough that NEE is saving about an order of magnitude of probing compared to just probing with every word, I can trade off precision for recall and achieve a high recall while greatly reducing the amount of necessary probing.

For the second experiment, my initial two months of running the entire infrastructure includes downloading and parsing the news from seven sources every day, labeling the named entities, and probing for both GET request and search engine

censorship. This data has various issues such as downtime and the need to remove some polluted data manually, but gives promising anecdotal evidence that censorship of current events can be detected using NEE. I provide a summary of the types of words I found to evoke censorship and how the different forms of censorship seem to vary with the news, with the caveat that these are preliminary results and no specific conclusions are meant to be drawn from them, beyond the assessment of the whole infrastructure as a viable means with which to dynamically probe a network.

3.3 Results

In this section I present both sets of results: results from testing for the specificity and recall by withholding labels from the Wikipedia dataset, and results from a two-month run of the entire infrastructure that coincided with the Jasmine Revolution of 2011, a movement in China inspired by the Jasmine Revolution in Tunisia.

3.3.1 Specificity and Recall

For labeling the names of people, I obtained the following results:

- Specificity: 83.44%
- Recall: 89.63%
- Precision: 0.42%

A precision of 0.42% is usually not considered to be very good for a named entity extractor, but remember that this context is different. One way to interpret these results is that I can label only 16.6% of the words in my dataset as names of people

(thus reducing the amount of probing necessary by nearly an order of magnitude), and include 89.63% of the actual names of people in my probing by doing so.

The results for names of places are as follows:

- Specificity: 69.80%
- Recall: 96.3%
- Precision: 0.77%

And, finally, the results for names of organizations are as follows:

- Specificity: 88.40%
- Recall: 87.56%
- Precision: 0.28%

One of the more surprising results from my two months (spanning April and May of 2011) of data is that the HTTP GET request blacklist appeared to be fairly static, in contrast to other lists/mechanisms such as search engine censorship. That is, words did not seem to be added to or removed from the particular censorship blacklist on a daily, weekly, or even monthly basis. My data was taken during a time of many reports of arrests and censorship related to the Jasmine Revolution protests in China in 2011. Despite many of these current events being censored in search engines, I probed with these keywords for HTTP GET request censorship and saw none that were related to any current event. However, this technique did aid in the addition of words to the HTTP GET request blacklist, discovered by ConceptDoppler [30]. Discovered keywords words can be found in Table 3.2. As stated before, it is most likely that these words were previously unknown, as opposed to changes to the list

Text	**Translation**
dajiyuan	Pinyin for Epoch Times
罢课	Strike
freedom	Freedom
请愿书	Petition
华夏论坛	China forum
学潮	Student protests

Table 3.2: **HTTP GET request blacklist additions.**

during the probing period, due to their lack of relationship to events that transpired during the probing period.

I did notice that current events evoked censorship in search engines. Specifically, certain words caused the search engine to return a warning that results had been removed due to local laws. Note that this probably means that a website was removed that contained the word I probed with and was highly ranked, it does not mean that the word itself is on any keyword blacklist. This is an important distinction, because it would be incorrect to confuse censorship of a word with censorship of a webpage that contains said word. I determined that this is probably the case with the following experiment. I searched for both "fuck" and "fuck you" in both search engines that I used for probing. The word "fuck" causes the message saying results have been removed to appear, while "fuck you" does not cause the message to appear. This suggests that this form of censorship is more topical and not based solely on a certain byte string appearing in the query. However this does not preclude a blacklist for search engine censorship.

I witnessed search engine censorship of certain words from the news that I assert was definitely censorship based on current events due to their subject matter. The censored words related to current events can be found in Table 3.3:

Term	Meaning	Reasoning
茉莉花	Jasmine Flower	Related to the Jasmine revolution protests
七 十 七, 77, 七七	The number 77	Elicited censorship at a time when China's President was being criticized by many Chinese citizens. He had visited a woman at her home during a live newscast, and asked her how much she pays for rent.She replied that she paid 77 RMB
王府井	Wangfujing	An area in Beijing where some of the Jasmine Revolution protests happened.

Table 3.3: **Search engine censored keywords.**

3.4 Censorship Post 2011

In the time following the publication of this work, censorship has continued to be an issue worldwide [50]. Censorship in China is a highly studied topic due to their advanced censorship methods (*e.g.* [89, 31, 41]). The ability to dynamically discover possibly censored content and probe servers is the core of this chapter, and the tool presented is still employed by censorship tracking systems. Filterweb [35], for example, is a recent tool that dynamically discovers content to probe for DNS poisoning (a type of censorship similar to GET request censorship, where DNS requests are intercepted and incorrect results are returned).

3.5 Concluding Remarks

In conclusion, my results are promising in terms of building an infrastructure that can probe censorship with words from current events. My system was the first of its kind and preceded similar systems [136, 120] that perform dynamic analysis to study censorship. My NEE algorithm gives a good specificity and recall, and I demonstrated that this infrastructure can produce instances of censorship that are related to current events.

3.6 Chapter Acknowledgments

I would like to thank the anonymous FOCI reviewers for their insightful comments. I would also like to thank the many people who helped me improve my translations and gave feedback on other aspects of the paper. Fletcher Hazlehurst, Veronika Strnadova, Leif Guillermo, and Ronald Garduno helped with various aspects of implementation or understanding of the maximum entropy framework and features. This material is based upon work supported by the National Science Foundation under Grant Nos. #0844880 and #1025442. Antonio Espinoza and Jedidiah Crandall were also supported by the DARPA CRASH program.

Chapter 4

Vector-Based Dynamic Information Flow Tracking

Modern Dynamic Information Flow Tracking (DIFT) (often called Dynamic Taint Analysis or DTA) systems (*e.g.* [28, 95, 23, 71, 122]) are designed to be run on production systems. Thus, they were created with performance in mind with the focus being on simple applications such as tracking malicious inputs from the network into control data or tracking how data is copied throughout a system.

In this chapter I consider a different design point: applications where propagating taint through indirect flows is necessary and where relatively high performance overhead is tolerable. I consider the following problem: Given a program binary and an input, analyze a *single* trace of the program and identify any cryptographic keys that are used anywhere in the trace. Analysis of a single trace is important, because in reverse engineering it is often not realistic to expect the analyst to provide the tool with a representative set of traces as it would require a test suite of inputs taylored to each program. My V-DIFT implementation can accomplish this based entirely on information flow analysis, without any prior knowledge or heuristics about the

cryptographic algorithm itself.

My overarching goal is not to replace modern DIFT systems, but to apply DIFT to new areas of research that can benefit from offline analysis. However to accommodate these new applications, tradeoffs must be made, namely, performance for the ability to track indirect information flows.

4.0.1 Adequate Information

What modern DIFT systems possess in speed they lack in information necessary to meet my desired applications. Being only able to taint a location with a tag (many of which are binary) is limiting and can cause problems with information flow tracking; the worst being the problem of overtainting. Overtainting usually occurs when the instruction pointer (EIP in x86) becomes tainted, which leads to the program's entire memory quickly becoming tainted. Thus no useful information can be derived from the DIFT system. My focus in this work is on measuring the actual information flow in a meaningful way. I accomplish this by using vectors as taint marks.

Another problem that stems from a lack of information in DIFT systems is taint attribution, a problem that is common in systems with binary taint marks. Because a cryptographic key must be stored in memory before use, being able to attribute the taint of an output to its source memory location is essential to the task of locating cryptographic keys in memory. These problems exist because current systems lack the ability to handle indirect flows in a general way. Indirect flows can take the form of what Suh *et al.* [122] define as address and control dependencies.

In V-DIFT I use vectors as a representation for taint tags, and provide a way to combine them so that they generate actionable information flow of a program trace. Vectors allow me the ability to easily combine taint and propagate information about its source. Using vectors allows the use of simple linear algebra to approximate quan-

titative information flow, so that I am never faced with the dilemma of whether or not to propagate taint. Propagating taint is simply a matter of degree, with taint being combined *via* vector addition, and comparing taint values can be accomplished by simply taking the cosine similarity. This allows me to answer attribution questions, and the amplitude of taint marks serves as a proxy for how much mutual information there is between a taint sink and any taint source.

V-DIFT's use of vectors does not give a complete picture of the information flow but an approximation. This is due to the fact that I am only approximating information flow instead of strictly adhering to any information-theoretic definition of quantitative information flow. This relaxation of information flow allows me to "tease out" useful information from the trace without over- or undertainting (an issue in classical DIFT) rendering the results useless, thus supplying adequate information for analysis.

4.0.2 Terminology

The vocabulary used when discussing DIFT until now has largely been the same as static information flow described by Denning [38]. In this work, I propose separate terminology for static systems *vs.* dynamic systems, because the goals of each are different and the information available is not equivalent. I view *explicit* and *implicit* flows as being properties of a program and thus useful when describing static information flow analyses. Similarly, I use *direct* and *indirect* for describing the information flow analysis of program runs or traces. Thus *implicit* and *explicit* flows are properties of programs, and *direct* and *indirect* flows are properties of traces. Since DIFT can be viewed as a single-pass analysis of a program trace, *direct* and *indirect* flows are the terminology I prefer in this work.

I use the term *indirect* because some *implicit* flows cannot be measured in a

DIFT system that does not execute both paths of the code. For example, for a trace through this code:

```
y = 1;
if (x == 0) {
  y = 2;
}
```

I can never know that information flowed from x to y when x is non-zero because the code `y = 2` was never executed. This is an implicit flow. By contrast, an indirect flow only occurs when x is equal to 0. In the case where x is non-zero the assignment $y = 2$ is never observed and there is no flow, except an implicit flow which DIFT can never reason about because it operates on single traces. I define an *indirect* flow as occurring when information dependent on program input determines from where and to where information flows, so a DIFT system can only measure this flow of information from x to y when the indirect flow actually occurs. Thus, more traces of the program with different inputs yield different information flows.

By contrast, *direct* flows are flows that occur on any execution of an instruction regardless of the program's input. For example in: `add eax ebx` it is clear that information always flows from `ebx` to `eax` (because in x86 the result of the addition is stored in `eax`). By this definition copy and computation dependencies are direct flows.

4.0.3 Contributions

The work in this chapter makes the following contributions:

- I propose V-DIFT, a general and practical solution to the problem of over-tainting that uses vectors as tags and approximates quantitative information

flow for offline analysis.

- I provide insights about the overtainting problem. Specifically, indirect flows result from a lack of necessary information when a DIFT system must make a propagation decision. Separating this problem from the more general problem of implicit flows suggests that indirect flows can be solved and need not be a fundamental limitation of DIFT systems.

- I demonstrate a DIFT-based method for locating the source of cryptographic keys based entirely on information flow, *i.e.*, without relying on any information or heuristics about the cryptographic algorithm employed or making any assumptions about how it is implemented.

4.1 Related Work

All work on DIFT systems either assumes that the information flow will be reasoned about statically or uses simple heuristics that only work in narrow domains and do not generalize. To the best of my knowledge, my work is the first to handle indirect flows in a general way.

TaintBochs [22] is the first paper that I know of to apply DIFT in the modern sense. TaintBochs was used to analyze data lifetime for privacy reasons. Shortly after, several research groups concurrently and independently developed DIFT as a way to track malicious inputs and prevent attacks [122, 28, 95, 26]. These early DIFT systems largely ignored indirect flows, or had propagation rules based on very simple heuristics. For example, in Suh *et al.* [122] address dependencies are not propagated if the address is calculated using a scaled index base (an x86 construct for calculating addresses). In addition, control dependencies were not propagated at all in Suh *et al.*.

In Minos [28, 29] many assumptions were made, such as address dependencies only being propagated for 8- and 16-bit loads and stores, but not for 32-bit loads and stores (Minos was based on a 32-bit system). For a detailed discussion see Crandall *et al.* [29], Slowinska and Bos [121], and King *et al.* [72].

Later research in DIFT systems attempted to address indirect flows. For example, many DIFT system frameworks that are designed for flexibility [34, 103, 125, 23] enable address and/or control dependencies to be tracked, but offer no solution to the overtainting problem. Panorama [132] relies on the user to manually label for which address and control dependencies the DIFT system should propagate tags.

DTA++ [71] only taints indirect flows deemed *culprit implicit flows* found through static analysis. This requires multiple traces retrieved through special test executions that fully exercise the portion of code that potentially has undertainting (when executed ignoring control dependencies). DTA++ also suffers from a lack of byte-level taint attribution.

V-DIFT offers a way to handle indirect flows, that is not only general, but can be applied to a single trace (important for reverse engineering applications) and is byte-level attributable.

The main distinguishing features between existing analyses of cryptography in binary programs and my work is: (1) past work focuses on locating and identifying the cryptographic algorithm while my work focuses on locating the key; and, (2) past work relies on both the input/output relationship of cryptography algorithms and detailed knowledge about specific cryptography algorithms (*e.g.* [14, 135, 78]), while my work simply examines the information flow of a trace and requires no algorithm-specific information about the symmetric cipher. To the best of my knowledge, this is the first work to apply DIFT to the problem of reverse engineering cryptographic keys, using the program's information flow as a means to locate a cryptographic key.

V-DIFT allows me to define symmetric algorithms in general terms based on the key. Specifically, I can look for a small set of contiguous bytes that affect a significant amount of the output in a way that is correlated. Further analysis to detect which cryptographic algorithm is being used once the key is located is relatively straightforward.

Gröbert *et al.* [59] finds cryptographic primitives in binaries and identifies the algorithm. Their approach is relatively general compared to other past works, but relies on heuristics about instruction mixes, sequences, and loops. My method is more comprehensive, testing 27 programs as opposed to 11 programs and does not rely on heuristics. Because their technique requires no algorithm-specific signatures, templates, high-level reference implementations, and their experimental methodology is based on readily-available, open source libraries, I based my experimental methodology on that of Gröbert *et al.*. Wang *et al.* [127] and Caballero *et al.* [13] were earlier works that are similar works to Gröbert *et al.*. More recently, Hosfelt [63] presents an approach that is also similar to Gröbert *et al.* but is based on machine learning.

Whelan *et al.* [130] propose a method to characterize cryptography using their Pirate system, which is a DIFT system. However they only illustrate the correlation between inputs and outputs of cryptographic algorithms. Pirate required the tracking of address dependencies to be turned on to track AES CBC mode correctly, which is congruent with my results. Lutz [88] presents an approach to analyze cryptography that is based on DTA [95], suffering from its limitations, and the DIFT (*i.e.*, taint) analysis is only a small part of Lutz's technique. Ming *et al.* [91] also present cryptography algorithm detection as one application of DIFT, in a manner that is similar to Whelan *et al.*.

4.2 V-DIFT Implementation

I use vectors as taint marks to store and utilize information unavailable to other DIFT systems. This facilitates both constant time combination of taint as well as taint attribution. In this section I will discuss how vectors are handled in the system and how the information they carry is assigned, propagated, and utilized.

I developed V-DIFT in C and designed it to work with 32-bit x86 GNU/Linux executables. V-DIFT forks, attaches, and single-steps through processes using similar APIs as a conventional debugger like GDB. As it single steps, V-DIFT extracts a trace of an attached process in real time, allowing the system to access any memory or address location value in order to calculate address lookups. Each vector in the V-DIFT system contains $n = 200$ floating point values. My system assumes the program is not adversarial, *i.e.*, they do not contain anti-reverse engineering mechanisms.

Like traditional DIFT systems, V-DIFT assigns taint at designated *sources*. V-DIFT considers any input made through the read system call (`SYS_read`) a source. Additionally, any region of memory can be designated as a source, causing any reads from that memory to be tainted. Memory ranges are designated through a file that V-DIFT reads on startup. To track data provenance, each byte of source data is assigned a random taint vector that is, with high probability, close enough to orthogonal to other taint marks in the system for the results of DIFT analysis to be meaningful.

Dependency	**Combination**	**Example**
Computation	$\vec{dst} := \vec{src_1} + \vec{src_2} + \ldots$	`add eax ebx` $\rightarrow \vec{eax} := \vec{eax} + \vec{ebx} + \vec{eip} \cdot \alpha$
Copy	$\vec{dst} := \vec{src_1} + \vec{src_2}, + \ldots$	`mov eax ebx` $\rightarrow \vec{eax} := \vec{ebx} + \vec{eip} \cdot \alpha$
Address	$\vec{dst} := (\vec{src_1} + \vec{src_2} + \ldots) \cdot \beta + \vec{drf}$	`lea eax [ebx + 4 * ecx]` $\rightarrow \vec{eax} := (\vec{ebx} + \vec{ecx} + \vec{eip} \cdot \alpha) \cdot \beta + \vec{drf}$
Control	$\vec{flag_1}, \vec{flag_2}, \ldots := \vec{dst}$ $\vec{eip} := \vec{eip} + \vec{flag_1} + \vec{flag_2} + \ldots$	`cmp eax esi` $\rightarrow \vec{of}, \vec{sf}, \vec{zf}, \vec{af}, \vec{cf}, \vec{pf} := \vec{eax} + \vec{esi}$ `jnz` $\rightarrow \vec{eip} := \vec{eip} + \vec{zf}$

Table 4.1: **Dependency Table:** $\vec{drf}$ **is the vector of the dereferenced address. In the computation dependency case** $\vec{eip}$ **is always one of the sources added. Note that every time vectors are combined (represented with a + in this table) the vector scaling described in Section 4.2 is applied.**

Conceptually, I want to combine taint vectors such that each vector's length reflects the amount of mutual information the vector carries with any given byte of the taint source. The logic for combining taint vectors estimates this, but I do not constrain myself to any strict information-theoretic model. I desire that if the vectors are perpendicalar, their combined vector length increases, whereas if they are parallel, *i.e.*, the same taint vector, the vector length is unchanged since new information cannot be created by adding information to itself.

V-DIFT *combines* vectors a and b by computing their combination $c = a + b$. Then, the length of c, $\|c\|$, is scaled such that it is equal to $\min(\gamma, \max(\|a\|, \|b\|) + \min(\|a\|, \|b\|)\,(1 - sim(a, b)^n))$ where n is the number of elements in each vector, γ is a user-defined parameter, and sim is the cosine similarity function.

In addition to γ, V-DIFT has two other user-designed parameters that affect taint propagation: α and β, which affect the propagation of two kinds of indirect flow. α affects taint propagated by the instruction pointer, and β affects taint propagated by address dependencies, *i.e.*, when memory or register locations are used to calculate an address. See Table 4.1 for a complete list of how each type of dependency is handled and how the parameters affect each type of dependency.

In V-DIFT, the sink is the trigger for measuring a traced program's taint marks. This measurement can be triggered at any point during program execution. To detect cryptographic keys in memory the sink is any write (`SYS_write`) system call, since I am looking for output bytes heavily influenced by taint marks in each program.

Every time a sink is reached, V-DIFT uses the cosine similarity function to compare the taint vector of each output byte with every initial source vector, creating an m by n matrix, where the rows and columns are the input and output respectively. Each element in the matrix represents the similarity between the particular input

and output byte. This allows me to determine how much influence each source byte had on each output byte.

4.3 Experimental Methodology

In this section, I will discuss the experimental setup, from the system running V-DIFT to the details of parameter testing.

4.3.1 System Specifications

Tests were run on an Ubuntu 14.04.3 machine running Linux kernel 3.16.0-45. The machine had 256 GB of RAM and an Intel(R) Xeon(R) CPU E5-2637 v2 3.50GHz processor. The large RAM of the machine performing the tests is not a reflection of the specifications needed to run my V-DIFT system but facilitated running multiple tests simultaneously. No test used more than 300 MB of RAM.

4.3.2 V-DIFT Parameters

As discussed in Section 4.2 there are three user-defined parameters: α, β, and γ. I made these values adjustable to the user because these values may be application-dependent in order to properly propagate taint in the system in a meaningful way. I performed a full factorial experimental design for the parameters to discover the ideal values for my application.

Each cryptography program was executed 30 times, once for each configuration of the following parameter sets for α, β, and γ, respectively: {0.0, 0.0625, 0.125, 0.25, 0.5, 1.0}, {0.0, 0.25, 0.5, 0.75, 1.0}, and {8}.

Beecrypt	✓	✓							
Crypto++		✓	✓	✓	✓		✓	✓	✓
OpenSSL	✓	✓	✓	✓		✓			
	AES	Blowfish	Camelia	DES	IDEA	RC4	RC5	TEA	Twofish

Table 4.2: **Cryptographic libraries tested.**

4.3.3 Cryptography Programs

Thorough testing of my system was carried out by testing multiple algorithms across multiple implementations. I chose to test OpenSSL (version 1.0.1), BeeCrypt (version 4.2.1-4), and Crypto++ (version 5.6.1-6), all implementations with available source code. Although there is not a complete overlap of all implementations, the Blowfish algorithm was found in all libraries. I also tested both the ECB and CBC block cipher modes where possible. A total of 27 programs were tested (see Table 4.2). Each program reads 128 bytes of data, encrypts the data using one of the algorithms from Table 4.2, and writes the encrypted data to standard out. Each program had the symmetric key in the read-only data section, with typically dozens of kilobytes of other data (such as S-boxes, initialization vectors, locale info, and other program constants) to make locating the key very challenging.

All programs were compiled using the -m32 and -static flags using gcc version 4.8.4. I compile to a 32-bit binary since my implementation only covers 32-bit x86 machine code. The static compilation made running the program much faster and allowed me to reduce the tainted input, as all dynamically loaded libraries would have been tainted otherwise when they were read.

4.3.4 Detecting Keys

Key detection is performed by finding regions of memory that have a large effect on the program's output in the trace. Because the entire key affects every encrypted

block of data, I expect it to be highly correlated with the encrypted output. Furthermore, the different bytes of the key will be correlated with each other as they appear in the output. To determine the key region, I use the inner product of the matrix produced from the sink to reveal how correlated inputs are with other inputs at the time of output, as well as the input bytes' effects on the outputs.

Next, I search along the diagonal of the inner product matrix adding up all values in a square window the size of the key. This window size is 16x16 (as the keys were 16 bytes and I taint on the byte level) for all algorithms tested except for DES, because its key size is only 8 bytes. When the number of tainted inputs is much greater than 300, I find the top 300 regions of memory that are most correlated with the output and group together any contiguous memory ranges. These regions are presented to the user (*i.e.*, the reverse engineer using my tool) to examine as likely locations of the key.

4.3.5 Timing Tests

The average runtime of each algorithm was calculated by running each program 25 times with $\alpha = 0.0$, $\beta = 0.25$, and $\gamma = 8$, since these parameters produced the best results in terms of locating cryptographic keys. Different modes for an algorithm are considered separate programs. The tests were performed on a single thread with the test program selected at random from the set of 27 test programs.

4.4 Results

Figure 4.1 shows the results for ECB and CBC modes. If the cryptographic key is found in the top memory range suggested to the analyst then that is represented by a red square, and similarly, according to the legend. The y-axis triple repre-

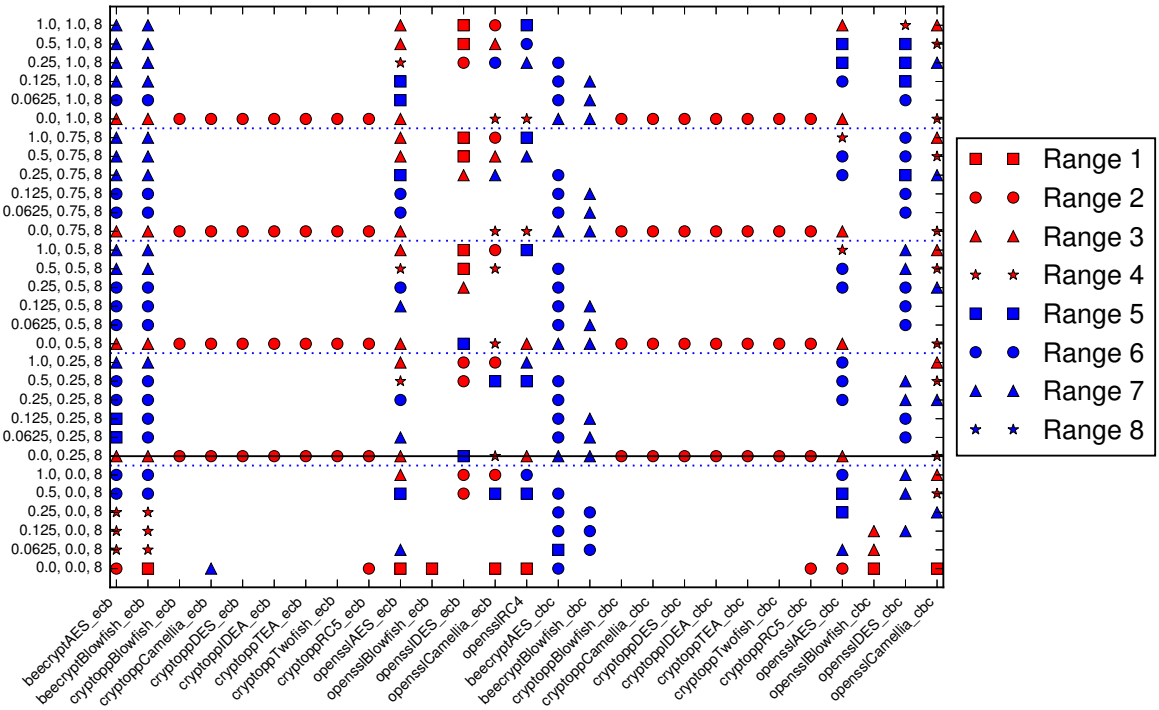


Figure 4.1: **Key detection rankings for parameters α, β, γ.**

sent the: scale value applied to control dependenceis, scale value applied to address dependencies, and maximum length allowed for the vector. The x-axis contains a cryptograpic algorithm prepended with the library it is implemented and the mode appended. For example, a y-axis triple of (0, 0, 8) means, ignore control and address dependencies, which is equivelant to only propagating direct flows. A y-axis triple of (0.5, 0.5, 8) indicates that the coltrol and address dependencies are multiplied by 0.5 (effectively cutting the lengh of each vector associated with each dependency in half) before propagating taint for each dependency.

The most surprising result was that control dependencies were not critical in locating the majority of keys in memory except in the case of OpenSSL DES, where control dependencies are necessary (see y-axis triple (1, 1, 8) above opensslDES_cbc). The fact that the Crypto++ library implementation of DES in CBC mode did not require control dependencies when the OpenSSL implementation did, reveals that

library implementation is more important than the encryption algorithm when detecting keys based on information flow.

My method for detecting keys had varying degrees of accuracy. As seen in Figure 4.1, the cryptography algorithms from the same library are more likely to be found in the same range, than the same algorithm from different libraries. This idicates that algorithms from the same libraries share more in common with each other than do the same algorithms from different libraries. For example Crypto++ was consistently found in the second memory range for all algorithms. The Beecrypt implementations were close in range as well. OpenSSL did not follow the trend, having caught two and missed two for CBC and only missed one for ECB.

The values chosen as the best parameters (seen as a line in Figure 4.1) match both ECB and CBC modes, but if the library is known ahead of time the user can adjust parameters accordingly.

Unintuitively, the best parameters for my application for α, β, and γ are 0.0, 0.25, and 8, respectively. Informally, this means "propagate taint for address dependencies, and none for control dependencies." Note that these parameters are specific to the application of locating cryptographic keys. The most surprising of these is the α value of 0.0. This means, for my application of DIFT, tracking control dependencies was unnecessary and doing so introduced noise into the system that reduced detection accuracy (with the exception of OpenSSL DES in CBC and ECB modes and OpenSSl Blowfish in CBC mode). This is surprising because I believed the algorithms that would benefit the most from control dependencies were cryptographic algorithms. The low value of β parameter was also surprising. My results indicate that address dependencies, even when using only a fraction of their taint, provide the information necessary to locate most keys.

4.4.1 Timing

Library	Algorithm	Mode	Average Runtime
Crypto++	Blowfish	CBC	20.5
Crypto++	Blowfish	ECB	19.8
Crypto++	Camellia	CBC	2.5
Crypto++	Camellia	ECB	2.3
Crypto++	DES	CBC	3.5
Crypto++	DES	ECB	3.4
Crypto++	IDEA	CBC	2.7
Crypto++	IDEA	ECB	2.5
Crypto++	RC5	CBC	2.6
Crypto++	RC5	ECB	2.4
Crypto++	TEA	CBC	2.6
Crypto++	TEA	ECB	2.4
Crypto++	Twofish	CBC	2.8
Crypto++	Twofish	ECB	2.7
OpenSSL	AES	CBC	0.4
OpenSSL	AES	ECB	0.3
OpenSSL	Blowfish	CBC	10.3
OpenSSL	Blowfish	ECB	10.3
OpenSSL	Camellia	CBC	0.4
OpenSSL	Camellia	ECB	0.3
OpenSSL	DES	CBC	0.2
OpenSSL	DES	ECB	0.3
OpenSSL	RC4	N/A	0.3
BeeCrypt	AES	CBC	0.3
BeeCrypt	AES	ECB	0.2
BeeCrypt	Blowfish	CBC	1.1
BeeCrypt	Blowfish	ECB	1.0

Table 4.3: **Average runtimes (seconds).**

The average runtime in seconds for each test can be found in Table 4.3. The overall average for all tests was 3.6 seconds. The Blowfish implementations for Crypto++ and OpenSSL were the outliers taking the longest time to complete, about 20 and 10 seconds respectively. However, the long runtime does not appear to be due to the cryptographic algorithm, but the libraries' implementation of the cryptographic algorithm. This is apparent when comparing BeeCrypt's implementation, which

completes in about 1 second, to the other two.

4.5 Summary

This chapter shows that vectors as taint marks (which I call V-DIFT) are a viable method for storing information in a DIFT system, and they give the information necessary to perform complex tasks such as key location (a feat impossible with traditional DIFT because of its lack of taint attribution). This is for two reasons: Vectors can distinguish input bytes, and they provide a means for handling indirect flows of information. I have applied V-DIFT to solve a common problem that reverse engineers face: locating cryptographic keys. I found that tracking indirect flows, specifically address dependencies, is important for any DIFT system to produce meaningful results. Because V-DIFT can be executed in seconds (as opposed to days), performance overhead was acceptable for offline analysis applications such as reverse engineering. I anticipate, with further research and development, that V-DIFT will lead to whole new ways for reverse engineers to use information flow in their analysis tasks.

4.6 Chapter Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. #1017602, #1314297, #1420716, #1518523, and #1518878. This work was also supported by the DARPA CRASH and Transparent Computing programs. I would like to thank the ARES reviewers and Geoffrey Alexander for valuable feedback. I would like to thank for Jeffrey Knockel for his code contributions to the project.

Chapter 5

Analysis of End-to-End Encryption in the LINE Messaging Application

Security and privacy features (*e.g.,* default HTTPS and multi-factor authentication) are becoming increasingly standardized in popular consumer applications. This shift is particularly apparent with the adoption of end-to-end Encryption (E2EE) in chat applications. Beginning in late 2015, popular chat applications (*e.g.,* WhatsApp, Facebook Messenger, Viber, LINE, and KakaoTalk) started to introduce E2EE features. While this trend is encouraging and some applications (*e.g.,* WhatsApp) have adopted well documented and reviewed encryption protocols (*e.g.,* Signal Protocol) there is a general lack of independent security research assessing the implementation of E2EE for many of these apps.

In this chapter I provide the first independent security analysis of E2EE features in LINE, a messaging application popular in Asian markets that has a user base of over 200 million monthly active users. My analysis reveals a vulnerability to a replay

attack and a vulnerability to an attack on a lack of forward secrecy.

The attacks I describe are within the capabilities of a well resourced attacker (such as a state actor), but the countries where users are the most at risk of surveillance of their LINE communications may be unlikely to carry out such attacks. For example, if a government cannot coerce LINE into colluding with them or breach LINE's infrastructure, then attacks that require the LINE private key may be unfeasible. If a replay attack requires physical access to a phone, and users rarely delete messages, is the state actor more likely to inspect the message history than to carry out a sophisticated cryptographic attack? Finally, how do researches and vendors find a good compromise between "worst-case" scenarios and scenarios that are generalizable to the greatest number of users?

These questions underline differences between how researchers and vendors may evaluate threats. Vendors attempting to implement security at scale for millions of users face hard decisions over balancing security, usability, and resources. Researchers finding and reporting vulnerabilities may be well-versed in security best practices, but they are unlikely to appreciate the decision-making processes that led to security designs and implementations. Meanwhile, the average end-user is presented with security features, that he or she is unlikely to be able to assess. In this chapter I first describe my analysis of E2EE in LINE, and then discuss how to better bridge researchers, vendors, and end-users.

5.1 Background

This section provides an overview of trends in encrypted messaging and communications security in LINE.

5.1.1 Encrypted Messaging

Security researchers were concerned about message security long before popular chat applications began adding security features to their clients. OTR [11] was proposed in 2004 as an alternative to PGP [137]. OTR offers users forward secrecy (discussed more in Section 5.2.6) and is designed for use in conjunction with messaging protocols such as XMPP. E2EE was introduced in many popular chat applications within two years of the 2013 Snowden revelations, which invigorated public debate over digital communications interception. The invention of the double ratchet [40] solved the problem of devices needing to decrypt messages received while offline, and it is now widely adopted. However early E2EE implementations such as that of LINE use other solutions. See [124] for a more complete review of secure messaging.

5.1.2 LINE Overview

LINE was released in 2011 by LINE Corporation, a Japanese subsidiary of South Korea's Naver Corporation. Since its release, LINE experienced rapid growth in Japan, and later in other Asian markets (*e.g.,* Thailand, Taiwan, *etc.*). In 2016, LINE reported over 200 million monthly active users (MAU) [90].

Amid this growth, LINE has come under government pressure to implement content controls and provide government access to user communications. Previous work documented client-side keyword filtering enabled for users based in China to comply with Chinese content regulations [60]. In 2013, the government of Thailand announced plans to monitor LINE communications. LINE responded by stating that access to user information would only be permitted with a Japanese court order and that no official request had been filed by Thai authorities [75]. Recent LINE policy documents [85] state that it responds to non-Japanese requests for user data through the mutual legal assistance treaty (MLAT) process.

In addition to these pressures, LINE has incrementally improved how it encrypts traffic. Version 3.9.2 and earlier LINE releases only encrypted client-server communications over WIFI and not 3G [75]. There was speculation that the lack of encryption on 3G may have been intentional to facilitate compliance with lawful interception requests [1]. In a blog post, LINE explained that when it introduced the SPDY protocol into its platform it decided to allow non-encrypted connections over mobile networks to avoid slow connection and transfer times [114]. In version 3.9.3 (released in October 2013) LINE introduced encryption over both WIFI and 3G [75].

In 2014, LINE announced [2] its "Hidden Chat" feature, a special feature of a conversation that users could enable where subsequent messages were "sent in a secure state." This feature was implemented at a time when other messaging apps such as KakaoTalk [3] and Telegram [4] implemented or announced improvements to "hidden chat" or other non-default encrypted communication features. In 2015, LINE announced its "Letter Sealing (End-to-end Encryption)" feature [83], and in 2016 this became a default feature [80], which sought to extend the "sense of security" that people had using Hidden Chats as the default for all messages. Two months later, LINE updated the Letter Sealing feature [81] to include a lock in the UI that informs users when messages were being "stored on LINE's servers in an encrypted state." LINE then published a "Technical Whitepaper" [82] describing Letter Sealing's cryptographic implementation in detail, which I refer to in subsequent sections of this chapter.

Note that LINE added message security features throughout 2014, 2015, and 2016 [2, 80], whereas the double ratchet algorithm [40] was not published until late in 2016. This timeline is one reason why LINE does not use a double ratchet, but there are also usability concerns discussed in Section 4.4.1.

5.2 LINE Technical Analysis

This section describes my threat model, attack implementation, and proof-of-concept exploits. The motivation for my analysis is to compare the implementation described in LINE's "Technical Whitepaper" to that of the version of LINE released shortly after the whitepaper's publication.

5.2.1 Threat Model

For the attacks described in Sections 5.2.3 and 5.2.7, I assume that the client is running LINE version 6.7.1. This version was selected because LINE claimed it would be the first to implement the security features described in the white paper. The threat model assumes the attacker is the server, or an attacker with the server's private key and perspective of the network. This threat model is the same threat model as outlined by LINE in their Letter Sealing blog post [86]. In regards to their old message encryption protocol, "While only hypothetical, there is one flaw with this method. A hacker inside the LINE servers could still be able to compromise the safety of message data." I believe this to be a reasonable threat model as it is presented as LINE's reasoning for implementing "Letter Sealing."

For the attack described in 5.2.7, I assume the above threat model and additionally that one of the clients' private keys is compromised. This is reasonable since state actors could coerce individuals into unlocking their phones, which would give an attacker the ability to obtain the necessary keys. The attacker could be a nation state or an entity with the ability to coerce LINE into retaining detailed message logs.

For the attack described in 5.2.3, there is no need for either the clients' private key or a device to be compromised. Rather, the LINE server can replay messages

by simply sending the ciphertext again. This property is not one that an end-to-end encryption system that follows cryptography best practices would have.

5.2.2 Attack Environment

My attacks assume that the attacker can see what the LINE server sees. In order to accomplish this I used the network setup shown in Figure 5.1. In this setup all traffic

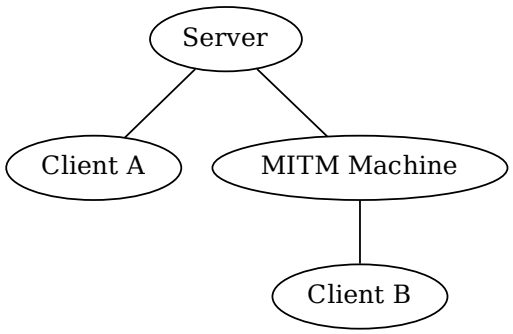


Figure 5.1: **Attack network setup. Note that the setup relies on *Client B* colluding with the MITM machine, this allows it to simulate having the same privileges as LINE's server.**

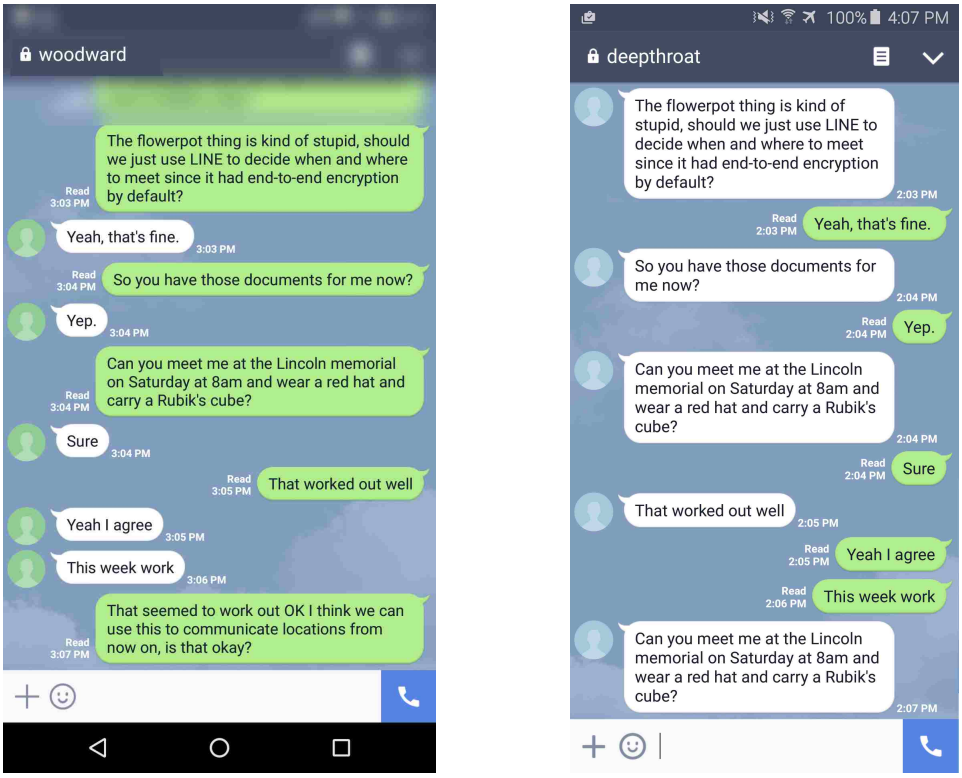
to and from *Client B* must pass though the man-in-the-middle (MITM) machine. I give the MITM machine the AES key used by *Client B* and the server to communicate with each other. Each message from the server to *Client B* is encrypted with this key and a static initialization vector that is hard-coded into the LINE app. With these two values the MITM machine can decrypt and re-encrypt any client-to-server LINE message (note the E2EE portion of the message is still encrypted). This allowed me to simulate the attack from the vantage point of the LINE server.

5.2.3 What is a Replay Attack?

A replay attack is an attack where an adversary records messages between two parties and can later replay any of those messages to either party member as though it were sent legitimately. The attacker does not need to know what the message decrypts to in order to send it. My replay attack does not send messages, but replaces the body of a message in transit with any message seen before. I chose to implement the attack this way because it did not require me to completely learn LINE's protocols for sending and receiving messages. See Figure 5.2 for screen shots of the attack's effects on both message sender and recipient. Notice that the last message seen in Figure 5.2a is not the same as the last message seen in Figure 5.2b. Instead the message has been replaced with a previous message sent from "deepthroat" to "woodward." This attack is enabled by a problem in the end-to-end encryption protocol, specifically the way deepthroat's messages are authenticated by woodward.

5.2.4 What is a MAC?

Message Authentication Codes (MACs) are used to guarantee the integrity of a message, ensuring that the message has not been altered in any way. When a message is received, the receiver calculates the MAC and compares it to the MAC the sender calculated, this proves that message received is the same as the original message sent. A good MAC has a different key from the one used to encrypt a message. In addition, a good MAC authenticates additional data, such as source and destination information and a message number. This additional data protects the message from replay attacks.



(a) **What deepthroat sees** (b) **What woodward sees**

Figure 5.2: **A conversation thread as viewed from 'deepthroat' (a) and 'woodword' (b).**

5.2.5 Replay Attack Implementation

Through reverse engineering I was able to discern which parts of the packet contain various components of the end-to-end communication. The three important sections are: the salt, the encrypted message, and the MAC. When the server sends a new message with the first two fields the same as an old recorded message, and the correct MAC, the message is replayed. The MAC used is a custom implementation not well documented by LINE called, "LEGY HMAC." Although it contains the name 'HMAC' it is not an HMAC implementation. The salt is a random 8-byte

value used to derive the encryption key using the following function:

$$Key_{encrypt} = SDHA256(SharedSecret||salt||\text{“}Key\text{”})$$

In order to demonstrate the replay attack, I waited for a message of similar size to be sent from one client to another and replaced the three critical sections: the salt, the encrypted message, and the MAC. I can replace these values in any message I see in transit on the network (using the threat model described in Section 5.2.1). This allowed me to carry out the attack without needing to completely reverse engineer the client-to-server protocol. Once the message is replaced, the LEGY HMAC is recalculated and replaced. This simple attack is outlined in Figure 5.3. Variable sized messages with the attack are possible, as are attacks not requiring a new message to be sent, but I did not implement these features in the attack because I only sought to demonstrate that replay is possible. Although in Figure 5.2 the replayed message happens only minutes later in my example attack, the same message played a week later would have a totally different context.

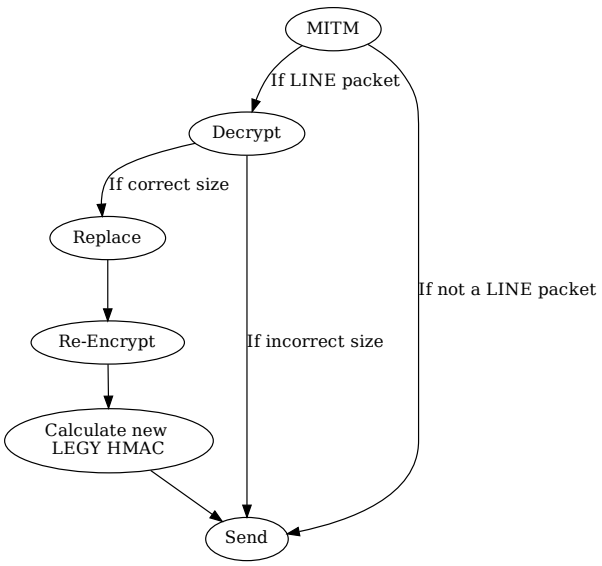


Figure 5.3: **Attack flow graph.**

Note that the packet has two layers of encryption, the E2EE and the client to server encryption. The E2EE is encapsulated by the client to sever encryption. I used a MITM to remove the client to sever layer because the threat model views the server as the attacker. Note also that a more sophisticated version of the attack could replay messages of arbitrary length, or not even wait for new messages to replay old messages, but the proof-of-concept exploit was kept simple for demonstration.

The replay attack described above is possible because LINE only authenticates the message itself, leaving open the possibility of replay attacks. It also deviates from cryptography best practices because LINE uses the same key for the MAC as for encryption (although my replay attack does not exploit this weakness). Both this attack and the attack on forward secrecy assume that the attacker has the same privileges as the LINE server (Section 5.2.2 describes my implementation).

5.2.6 Forward Secrecy Implementation

Forward secrecy is a property of an encryption system that removes an attacker's ability to decrypt past messages, even if one or more users' private keys are compromised [124]. For client-to-client communication, forward secrecy is implemented by generating a new key for each session or message exchanged between users (called an ephemeral key). The most important properties for security are that the key for this session be generated in some way that is not predictable or deterministic and that the encryption is "end-to-end," meaning that it is encrypted from the user sending the message to the intended recipient.

LINE, however, only offers forward secrecy from client-to-server, meaning that ephemeral keys do not protect the user from a malicious actor with the same privileges as the LINE server. The company could be compelled to save communications between two individuals and decrypt them. This attack is possible if the attacker

gains access to the secret key of just one of the users' devices. This method of attack requires physical confiscation of a device, by law enforcement, government agents, employers, *etc.*, but an attacker with one private key of one user would be able to recover messages, even if they have been deleted from both users' devices. A detailed example of such an attack is presented in Section 5.4.2.

5.2.7 Attack on Lack of Forward Secrecy

To illustrate the vulnerability posed by LINE deciding not to include forward secrecy from client-to-client, I collected messages as the server would see them using the same setup as for the replay attack. This allows me to view the messages with the first layer of encryption removed. If the device is then compromised—by an adversary confiscating it, for example—they would need only to retrieve the shared secret (by obtaining one of the users' private keys) that is used between users and the salt from the message to derive the initialization vector and key needed to decrypt the message. I demonstrated this by using my private key from one device to decrypt a message from the E2EE ciphertext that the server sees.

5.3 Responsible Disclosure

Becuase reverse engineering is an important tool used to strengthen the security of commonly used closed source applications, all my findings were disclosed to the LINE security team. I first disclosed my findings on forward secrecy to LINE December 20, 2016. On December 21, 2016 LINE replied to my initial disclosure. Shortly after their reply to my initial disclosure, I discovered the replay attack, which I disclosed to LINE on January 13, 2017. On January 27, 2017, LINE replied to my second disclosure. See Table 5.1 for a detailed timeline of the disclosure process.

Date	**Contact**
12/20/2016	Forward secrecy disclosure.
12/21/2016	Reply to initial disclosure concerning forward secrecy.
1/13/2017	Replay attack disclosure and inquiry about inconsistencies.
1/27/2017	Reply to replay attack and inconsistency questions.

Table 5.1: **Disclosure timeline.**

In reply to my first disclosure, LINE agreed that forward secrecy would improve the security of their E2EE implementation, but they explained they had decided not to include it in the first release of the feature. They acknowledged my threat model, but argued that their threat model addressed more immediately practical concerns:

> While FS [Forward Secrecy] for messaging could be added in a future version, currently FS is only available at the transport level. I believe this covers the more realistic case where LINE server keys are leaked, stolen, or confiscated by authorities. As for your threat model, if a device is confiscated, whoever has the device will be able to read stored messages, even without going through the trouble of extracting keys.

I replied with the following:

> Regarding the practicality of the attack I proposed, against an attacker with the privileges of the LINE server the True Delete feature described here [84] is ineffective. Because a user has no guarantee that the LINE server is not recording ciphertexts, this means that deleting messages on their device, even if both the sender and receiver delete their copy of the message, does not preclude the possibility of plaintext

> being recovered in the future. LINE could collude with law enforcement to confiscate a phone and recover all deleted past messages by extracting the E2EE private key of the user.

LINE responded that the True Delete feature is only intended to be used against an attacker with physical access to the device, and it is not intended to guarantee users that copies of messages on servers are deleted, which LINE argued would be unnecessary anyway because an attacker who could compromise transport security could make copies of the messages.

In this second round of communication, LINE also provided a detailed response to my questions conveying their design decisions with respect to forward secrecy. They explained that forward secrecy was left out of the first release due to the complications of synchronizing between mobile and secondary devices (primarily desktop clients). They said that they would "strengthen the MAC calculation" in the next Letter Sealing version, and work to address the replay attack issues, but that the secondary applications synchronization issue, in addition to being a challenge for implementing forward secrecy end-to-end, also presents some challenges for making other algorithm changes.

In the second round of communication I also pointed out inconsistencies between LINE's whitepaper [82] and the implementation, namely:

- The whitepaper says that the Client-to-Server Transport Encryption protocol uses an ephemeral Initialization Vector (IV) along with the ephemeral encryption key for AES, but I found that the IV is hard-coded and never changes.
- The whitepaper says that the Client-to-Server Transport Encryption protocol uses AES-GCM, but I found that it uses AES in CBC mode.
- LEGY-HMAC, the MAC used for client-to-server communications (not doc-

umented in the whitepaper) has only a 32-bit digest and is based on a hash algorithm that is not cryptographically strong.

- For E2EE encryption, the same key is used for encryption as well as for the Message Authentication Code (MAC). It is considered a common best practice to use separate keys, to preclude the possibility of chosen plaintext attacks leading to message forgery.
- For E2EE encryption, the MAC is a simple "hash and encrypt," compared to something like HMAC that precludes length extension attacks.

LINE responded that these issues had been fixed, but the fixes were disabled in the version I reverse engineered because of a bug.

5.4 Discussion

While consumer applications steadily announce new features to protect the confidentiality of user communications, there exists a divide between how different stakeholder groups conceptualize and communicate their threat models to one another [106]. Building from my case study I discuss gaps in mutual understanding and communications between researchers, companies, and end-users and ways to address them.

5.4.1 Bridging Gaps Between Researchers and Vendors

Security researchers seek to probe popular and emerging systems for novel vulnerabilities that can contribute to the research literature and general security understanding. In some cases, exploitation scenarios for novel vulnerabilities can be difficult to communicate to other stakeholder groups. My case study provides an example of

how such difficulties can impact the responsible disclosure process. While I sought to communicate to the LINE security team the applicability of my replay attacks and the lack of client-to-client forward secrecy despite their stated threat model, there was a disconnect in terms of views of the severity of the threats. A common theme was that the complexity of the interaction between my reported vulnerabilities and the broader design decisions made by LINE allowed both sides of the communication to reframe discussions.

For example, when I reported vulnerabilities in the LINE cryptographic protocol using the same threat model that motivated LINE's E2EE implementation [81, 83], part of their response was to point out that an attacker could just confiscate a user's phone and see past messages that way (an argument that ignores the threat model). As pointed out in Section 5.3, another way to view the attacks that take advantage of the lack of forward secrecy (and the ability to replay messages from the server, for that matter) is to view them as violations of LINE's True Delete feature [84], which physically removes the messages on a user's device when deleted so that common forensics techniques for persistent storage cannot be used to retrieve deleted messages. Even if both sides of a conversation delete a message from that conversation, it can be replayed or recovered using my attacks.

In summary, my disclosure process with LINE began with a discussion of forward secrecy as a transport security issue; LINE replied that physical device confiscation was a more realistic threat; I recast forward secrecy as a forensics issue; and LINE cited the lack of perfect solutions for transport security as a fundamental limitation on anti-forensics techniques. Both parties apparently have the same goal (making the application more secure), but cryptography and the real-world threats that give it context are so complexly intertwined that technical conversations about threats tend to shift in topic very easily.

5.4.2 The Importance of Forward Secrecy

To better understand forward secrecy as an anti-forensics technique, imagine the following hypothetical situation. Alice the reporter has a source for a story who is code named Bob. The story is about the government of Country X. Using LINE for messaging, Alice and Bob have a private conversation about the story. Bob uses a "burner" phone to communicate with Alice, meaning that after their communications, but before Alice's story is published, he physically destroys his mobile phone and all the data on it. After Alice publishes the story, she plans to travel to Country X. Knowing that LINE has end-to-end encryption and the True Delete feature, she decides that she should delete all of the messages between her and Bob, but keeps LINE installed on her phone to preserve other conversations and contacts in the LINE app.

Alice is detained at the border of Country X and her phone is confiscated by authorities. Several hours later during an interrogation with authorities she is presented with a plaintext decryption of her entire conversation with Bob. Remember that Bob's phone was destroyed, Alice's phone had no copies of the messages on it, and they always used LINE's end-to-end encryption feature. So how did authorities obtain the plaintext conversation?

The answer is that LINE's end-to-end encryption does not have forward secrecy for end-to-end client communications. Therefore, when the government confiscated Alice's phone they were able to extract her private key and use it to decrypt all of the encrypted messages that LINE had recorded. Realistically, however, many governments prefer less technically sophisticated attacks and many users do not bother to delete old messages, so keeping threat models grounded in likelihood is another challenge for communication between researchers and vendors.

Part of the general disconnect between researchers and vendors may be lack of

common understanding. LINE has a bug bounty program [79] to encourage independent security research of its platform and they engaged with me in good faith. However, like other major software companies it has hundreds of millions of users and must prioritize many security issues and consider impact on usability, uptime, and other variables. Researchers analyzing the platform from outside these processes may not fully appreciate the competing priorities. An example of this challenge is the technical problem of secondary devices for chat apps with E2EE features. In WhatsApp, for example, a master device must be online before secondary devices (*e.g.*, their web app) can encrypt and decrypt messages. LINE did not want to restrict their users similarly, which prompted the forward secrecy to be client-to-server, rather than client-to-client.

While ample literature exists on end user understanding and implementation of security advice [5, 66, 48, 108], there is little comparable research examining how security teams at software vendors understand and act on vulnerability reports. Existing work focuses largely on the process and timing of disclosure, patching, and publication, and not on the effectiveness of communications (*e.g.*, [17, 7]). A study focused on the substance of and reaction to vulnerability disclosure communications could help both security researchers better communicate their results and vendors better appreciate the severity of issues.

5.4.3 Better Communicating Research

The community (both academic and the privacy community at large) discovers attacks against E2EE [53, 27, 70, 68, 129, 134], reverse engineers apps such as LINE for a variety of purposes [111, 112, 113, 110, 52], performs formal security analyses of protocols [51, 25], and compares E2EE implementations [131, 6]. Even within the community, threat models range from showing a lack of semantic security without demonstrating practical attacks to very real attacks against widely used apps [53].

There are few actual attacks by state actors and contractors to use as case studies, and very little information about those that are reported [47, 49]. This wide variety of threat models with no common understanding about which ones present real risks makes it difficult to communicate with vendors and users.

5.4.4 Better Educating Users

Vendors, the media, and in some cases, security trainers, all communicate with end users about how application privacy and security features work and how to stay safe online. Despite these efforts, studies show that end users have difficulty understanding the basic premises of end-to-end encryption [5] and they have different mental models for how to stay safe online than experts [66]. Other research shows that even users with higher levels of security literacy are not necessarily more secure when considering the security of their devices [48]. Further work has demonstrated a "digital security divide," whereby users with lower socioeconomic status or education levels rely on lower quality security information [108].

Media organizations often write sensational stories about security vulnerability reports (*e.g.*, [87]), spreading uncertainty to users who already have difficulty adopting secure practices. While sensational media stories are certainly not limited to security vulnerabilities, given the already low level of security literacy among the general public, researchers have the responsibility to educate and inform the media, who in turn have the responsibility to provide balanced and accurate information to the public. Research examining the communication between security researchers and journalists, focusing on how journalists interpret vulnerability publications, could help encourage more thoughtful and accurate media stories. Such an investigation could provide recommendations that could improve reporting on security vulnerabilities.

5.5 Summary

I reverse engineered version 6.7.1 of LINE and discovered it was vulnerable to a replay attack and an attack on the lack of end-to-end forward secrecy between clients. These attacks assumed the same threat model described in LINE's security documentation. Based on my analysis and communications with LINE I identified an open question for vendors and security researchers: How do we, as researchers, find a good compromise between "worst-case" scenarios and scenarios that are generalizable to the greatest number of users?

As pointed out by Rogaway [115], researchers should "Regard ordinary people as those whose needs [we] ultimately aim to satisfy." The challenge is how to do so while avoiding unintended consequences and with deference to the perspectives of other stakeholders. Users who decide not to use a specific messaging application because of security concerns discovered by researchers, for example, might simply fall back to SMS messaging, which has no end-to-end security at all. Practical concerns, such as secondary devices, are rarely considered in academic work, forcing vendors to create their own solutions. With this I work was able to begin the discussion about what role the research community should play in addressing these types of issues.

5.6 Chapter Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. #1314297, #1420716, #1518523, and #1518878. I was supported by the Open Technology Fund Information Controls Fellowship Program. This research is part of the Net Alert (https://netalert.me) project funded by the Open Technology Fund. I would like to thank Jeffrey Knockel for commenting on drafts, providing his library for modifying packets in flight, and implementing LEGY

HMAC in python source code. I also thank the LINE security team for comments on drafts and their engagement during the disclosure process. I am also grateful to the anonymous FOCI reviewers and shepherd, Nick Weaver, for valuable feedback. Finally, I would like to thank Andrew Hilts and Masashi Crete-Nishihata for their writing contributions to this chapter.

Chapter 6

Internet-scale Study of TCP Initial Sequence Numbers

TCP three-way handshakes are the first step for virtually all web connections on the Internet, and have a history that goes back decades. Much of the evolution of this part of the TCP protocol has revolved around choosing TCP initial sequence numbers in a way that mitigates various off-path attacks. In this chapter, I seek to understand how well hosts on the Internet choose initial sequence numbers with respect to algorithmic complexity and mutual information (as measured by independent hosts connecting to the same destination IP). I collected samples from every IPv4 Internet host that replies on port 80 (more than 60 million), and found that over 11.5% (7,053,001) of these hosts choose initial sequence numbers in a vulnerable manner which leaves them susceptible to blind spoofing attacks. I present an analysis of the most prominent clusters of vulnerable hosts.

A significant amount of research and thought has gone into choosing TCP Initial Sequence Numbers (ISNs) over the years, with ISNs being a security concern well before TCP was even defined as a protocol [33, 123]. Well crafted ISNs protect

TCP connections from many attacks such as *blind RST*, *blind injection*, and *blind spoofing* attacks. These attacks are described in more detail in Section 6.1.3, and are possible when an attacker can correctly guess the: source IP, source port, destination IP, destination port, and sequence number of a remote connection. Blind injection attacks, where an attacker opens a TCP/IP connection without being able to see the SYN/ACK (and therefore ISN) from the server, are particularly interesting because the attacker initiates the connection and does not need to have precise timing or guess any IP addresses or ports. Therefore only the apparent algorithmic complexity in the server's ISN (from the attacker's perspective) provides any security against this type of attack. Algorithmic complexity is sometimes informally called "entropy" or "randomness".

One of the most salient studies on TCP initial sequence numbers was that of Zalewski [133] in 2001. His in-depth analysis of how specific operating systems generate ISNs was not broadly applicable across the Internet. My work is one of breadth, and, to the best of my knowledge, it is the first of its kind to study how well the Internet as a whole chooses ISNs. Specifically, in this work I answer two questions:

1. How many servers on the Internet lack algorithmic complexity in their ISN creation algorithm that leave them vulnerable to attacks?
2. Is there observable mutual information in ISNs among servers that lack algorithm complexity in their ISN creation algorithm?

Although placing a lower bound on the algorithmic complexity (also known as Kolmogorov complexity or algorithmic entropy) of a binary object (or string) is formally undecidable [19], a description of a binary object that is itself a certain size can place an *upper* bound on algorithmic complexity. Put another way, I cannot prove that a binary object (such as a sample of ISNs) has at least a certain amount of

algorithmic complexity, but I can demonstrate that it has at most a certain amount of algorithmic complexity. A caveat for the terminology used in this paper is that all ISNs that comply with relevant RFCs have low algorithmic complexity. However, I (or the attacker) would need to know the cryptographic secret used to generate a sequence of ISNs in order to produce a concise description of the sequence of ISNs. Throughout the paper, when I refer to algorithmic complexity I mean specifically the *apparent* algorithmic complexity from the perspective of an off-path attacker who does not know this cryptographic secret.

I scanned all IPv4 addresses that respond to SYNs with SYN/ACKs on port 80. I then scanned the roughly 61 million machines I found (I verified that this is roughly all web servers on the Internet using Censys [18] data) with the single point and double point scans described in Section 6.2. My analysis focused on web servers because they are the dominant users of the TCP protocol. Of the scanned machines I found three classes of ISNs. First are those that I believe, based on my measurement methods, adhere to the relevant RFCs (that describe how to generate ISNs) and have ISNs with good algorithmic complexity. The second class of ISNs are ambiguous with less algorithmic complexity than an RFC-compliant implementation would demonstrate, but I do not provide a concise description of the binary object that describes these ISN samples. The third class comprises those servers that are definitely and quantifiably of lower algorithmic complexity. I demonstrate this both through concise descriptions and through demonstrated mutual information between client IP addresses measuring the same server. This last group is concerning, because there are known methods for choosing ISNs securely [9]. One constituent of this group, for example, is the 11.5% of machines scanned that are quantifiably vulnerable to blind spoofing attacks. This is not a restatement of the problem with SYN cookies, SYN cookies account only for 5.3% of the 11.5% that are vulnerable.

I considered two information theoretic measures: algorithmic complexity and

mutual information. Informally, algorithmic complexity is the "randomness of an object," as viewed from the perspective of the attacker. Algorithmic complexity is used because the ISN list is the object I have access to as opposed to the algorithm that generates the ISN which would be measured in terms of entropy. Mutual information is the measure of mutual dependence between two variables. The double point scan provides two vantage points from which to measure the mutual information of the ISN sequences from a given IP address.

I approached the measurement of mutual information from two different angles and found that many of the ISN sets that lack algorithmic complexity also share mutual information when tested from different vantage points. Like algorithmic complexity, the presence of mutual information is highly undesirable in ISN creation. If an attacker can determine that there is a lack of algorithmic complexity then the search space required to guess the ISN of the victim is reduced. Similarly, if an attacker can measure mutual information in a server's ISNs, they can connect to the same server as the victim and exploit the mutual information to reduce the set of ISNs they need to guess to carry out an attack.

To measure algorithmic complexity, or lack thereof, I viewed my data through three different lenses, each illuminating different aspects of the data: standard randomness tests, classifiers for common discovered patterns, and range. Likewise, to measure mutual information I viewed my data from two different lenses (fitting with common patterns and Pearson coefficient), each lens offering a perspective the other could not.

The rest of this chapter is structured as follows. I first provide background information (Section 6.1) on ISNs and attacks specific to adversaries being able to predict ISNs. I follow the background with the experimental design (Section 6.2) where I describe how the scans were performed and why, including caveats and limitations to the methodology. I next discuss implementation-specific details;

(Section 6.3). Then I describe the analysis in detail (Section 6.4); and summarize the main results (Section 6.5). This is followed by a discussion of ethical considerations (Section 6.6), related work (Section 6.7), and my conclusions (Section 6.8).

6.1 Background

In this section I give a general background on ISNs, followed by a section describing how pseudorandom number generators relate to ISNs. Finally I discuss some attacks that become feasable when ISNs lack algorithmic complexity.

6.1.1 Background on ISNs

An ISN is the first sequence number to fill the sequence number field in a Transmission Control Protocol (TCP) packet, by a host for a given connection. The ISN field is 32 bits in the TCP header, which means the search space for guessing a randomly generated ISN is 2^{32} (roughly 4 billion). However, TCP implementations do not use randomly generated values for ISNs, because near collisions could lead to confusion between connections. The method for picking an ISN was first described in RFC 793 [100], which outlined the TCP protocol. However, there were issues with the description which left the protocol susceptible to attacks. The first RFC to address the issue of ISN attacks on the Internet was RFC 1948 [8], written in 1996. In it Bellovin explains a simple attack that is possible when the attacker can guess the ISN of someone else's connection and describes what would later be called a *blind spoofing* attack. Bellovin also proposes a fix stating the ISN should be calculated with the following function:

```
ISN = M + F(localip, localport, remoteip, remoteport)
```

where `M` is a 4 microsecond timer and `F()` is a pseudorandom function of the connection-id and some "secret data". No further RFCs were written on the subject until 16 years later when RFC 6528 [9] was released making RFC 1948 obsolete. The proposed fix changed the suggested method for generating ISNs by explicitly adding a secret key to the function:

```
ISN = M + F(localip, localport, remoteip, remoteport, secretkey)
```

The RFC also outlines the specifics of the function `F()` along with how to choose a suitable `secretkey`. In both RFCs the `M` value is necessary to keep ISNs created with the same source IP, destination IP, source port, and destination port from repeating. If the RFCs were perfectly followed, this `M` clock should not be seen in my results due to the fact that I choose source ports uniformly at random when performing my scans (see Table 6.1).

6.1.2 PRNGs

Zalewski [133] performed in-depth work on identifying and exploiting Pseudorandom Number Generators (PRNG) of specific operating systems, specifically how they pertain to choosing ISNs. In contrast, my study is one of breadth where my goal is to see if I can find ISNs that were created non-compliant with the RFCs across the whole Internet. Zalewski's work utilized strange attractors to help guess ISNs. The strange attractors were created by using the ISNs as delayed coordinates, putting the ISNs in 3D space. The ISNs Zalewski used were gathered through passive measurement of a network. Zalewski's work provided two interesting use cases. One, he was able to guess what the next ISN was going to be for fifteen operating systems. Two, he was able to use the shape of the PRNG to fingerprint what operating system a server was running. Zalewski's strange attractor technique is not feasible in my domain due to

the large amount of data (approximately 50,000 ISNs per IP) the technique relies upon.

6.1.3 Attacks

There are two kinds of off-path attacks possible when the ISN is known, active and passive. Passive attacks, such as that of Knockel *et al.* [73], allow the attacker to count packets between two arbitrary hosts off-path. Knockel *et al.*'s passive attack is on the IP protocol itself rather than TCP, although it can infer ICMP, UDP and TCP traffic. Passive attacks are powerful because they leak information being passed between two hosts that the attacker does not control. All of the TCP/IP hijacking methods described in Section 6.7 can also be used as passive attacks, and they typically infer the TCP sequence number meaning that they could even count bytes.

Active attacks escalate the severity of the vulnerability allowing an attacker to control aspects of the connection. The attacks I focus on happen in the TCP layer and the attacker needs to know the source and destination ports, the source and destination IPs, and the sequence number (note my work in this chapter only aids in the discovery of the TCP sequence number). The simplest such attack is known as a *blind reset* attack. In this attack a victim's connection can be reset by an off-path attacker that can guess the correct sequence number of the connection. The natural extension of the *blind reset* attack is a *blind injection* attack. This attack is nearly identical to the *blind reset* attack, but instead of sending a RST packet to terminate the connection, the off-path attacker sends a legitimate packet with data to the victim. The last active attack I consider is a *blind spoofing* attack. In this attack the off-path attacker does not interfere with an active connection between client and host. Instead, the attacker opens a new connection to the host appearing to come from the victim. This attack requires the attacker to send more packets than before

because they must complete a "handshake" (blindly) and send some data, which in turn requires more time. However because the attacker is not interfering with an active connection, the attacker has more time to complete the attack. The ability to successfully implement these attacks is limited by the bandwidth of the attacker, or how many bytes per second the attacker can send. For a detailed description of *blind spoofing* see [61].

Blind reset attacks, before the release of RFC 5961 [107], were relatively easy to carry out due to the standards only requiring the RST to be in the receive window (as described in RFC 793 [100]). RFC 5961 changed this behavior and added a layer of security by sending challenge ACKs when the TCP sequence number was in window but not the exact number expected. My work assumes I can guess the exact TCP sequence number thus removing the obstacle of challenge ACKs.

For my analysis of all three attack types (blind spoofing, blind reset, and blind injection) I assume that the attacker knows the 4-tuple of IPs and ports that describe the connection. This is definitely true of blind spoofing attacks because the connection is originated by the attacker, *i.e.*, they send the initial SYN packet and choose the 4-tuple. For the other two types of attacks the attacker may know the client's IP but not the client's port, but would typically always know the server IP and port (*e.g.*, 80 for web). However, as is done by RFC 5961 [107], I conservatively assume the attacker knows the connection 4-tuple for blind reset and blind injection attacks. For example, if the attacker has an SSH shell on a shared server along with the victim, this 4-tuple information is easily available in the application layer *via* the `netstat` command.

Many off-path attacks require malicious code to be on the client of the off-path machine [55, 56, 58, 101, 102]. If the method for creating ISNs has poor algorithmic complexity, then this need for client side code execution is obviated. The strength of these attacks is that they work even when the server chooses ISNs correctly and

the RFCs are followed. My study, in contrast, is concerned with attacks that are possible because ISNs are not generated correctly.

In addition to active and passive attacks that are off-path, I believe it is possible to play tricks with the 3-way handshake required by TCP/IP in a form of on-path attack, if one can guess the sequence number with which the server is going to reply. Consider the following: The client sends a SYN and ACK back to back such that the ACK received by the host is the same (in terms acknowledging the server's ISN) as the ACK that would have been received during a standard 3-way handshake. Assuming the server processes the packets in the order sent, this should establish a legitimate connection. In order to accomplish this the client must be able to guess the correct ISN with which the server was going to reply. This could potentially allow for a 3-way handshake to be complete without correctly following the protocol. I believe this may have implications in systems that rely on observing 3-way handshakes such as an Intrusion Detection System (IDS). I leave this as future work, but present it here as yet another reason why secure ISN generation is important.

6.2 Experimental Methodology

In this section I will describe the types of scans performed, discuss the role of SYN cookies and backlogs, explain caveats, and describe the limitations of my study.

Scan Table

Scan name	**Source IPs**	**Source port**	**Sender ISN**	**Sender UID**
Single point scan	One	Pseudo Random	Pseudo random	Pseudo random
Double point scan	Two	Pseudo Random	Pseudo random	Pseudo random

Table 6.1: **Each source IP sends 100 SYN packets to the IP being scanned.**

6.2.1 ISN Scans

I performed two different ISN scans to measure different aspects of ISNs: single point and double point scans (see Table 6.1). The single point scan was performed from a single vantage point, meaning the source IP was the same for every destination IP scanned. The double point scan scanned each destination IP address from two different IP addresses in my research network. In each scan 100 SYN packets were sent from each IP address. For example, if there was only one source IP, each destination IP received 100 SYN packets. For each SYN sent I randomly generated an IPID, source port, and sequence number; as per the RFC, the ACK value was set to 0. Due to the fact that I need to know if the SYN/ACK received from a test machine is in the SYN backlog or a SYN cookie, I did not send RST packets after receiving a SYN/ACK.

These two scans allow me to answer my main questions:

- Does the ISN have poor algorithmic complexity? If so, then an attacker only needs to guess the reduced set of ISNs and any off-path attacks become quicker.

- Do clients connecting to the same server have mutual information? If the ISN is properly chosen, I do not expect there to be any practically discernible mutual information between the measurement machines. According to RFC 6528 [9] the ISN should be generated as described in Section 6.1.1. If I can determine that there is some structure to how the ISN is calculated in the double point scan then I can be certain that the server is not creating the ISN as RFC 6528 suggests (or that `F()` is not cryptographiclly secure). I can come to this conclusion because I vary the source port in the double point scan. Therefore, if I can find a structure in the creation of ISNs it must be a flaw in `F()` or a lack of compliance to the RFC.

The single point scan is technically a subset of the double point scan, however they were performed at different points in time and analyzed separately so I present both here. Also, there is a small chance for interference when scanning from two vantage points, but when analyzing the algorithmic complexity of the single point scan I can disregard this possibility. Therefore I kept both the single point and double point scans in my analysis.

6.2.2 Cookies and Backlogs

All my traffic causes the creation of half open connections. All half open connections should fall into one of two categories: SYN cookies or part of the SYN backlog. I expect to see both in my results, with half open connections in the SYN backlog being the majority. The main difference between the two is that SYN cookies are stateless and have no retransmissions associated with them. Items from the SYN backlog however, are retransmitted a finite number of times allowing me to estimate whether I am seeing SYN cookies or SYN backlog entries.

For the purpose of my analysis, it is sometimes necessary to know if I am measuring SYN cookies or connections in the backlog. Therefore, I do not send RST packets after receiving SYN/ACKs, instead opting to continue to collect as much information about the connection as possible. In my entropy range classifiers, I made a conscious effort to only test values I knew to be in the SYN backlog because it is known that SYN cookies have roughly 2^{24} bits of entropy, as the top 8 bits are known to be low entropy. The top 5 bits are a low resolution clock, and the following 3 bits are an encoding of the maximum segment size [10]. Because SYN cookies are stateless and have no retransmissions, I determined a value to be in the SYN backlog if I received duplicate ISNs from the server being scanned. In the worst case I mis-classify a SYN backlog entry as a SYN cookie if only one of multiple transmissions were received, but never mistake a SYN cookie for a backlog entry based on retransmissions because

SYN cookies cannot be retransmitted due to the fact that their purpose is to not keep any state on the server.

In the case of my bi-level and up-trending classifier (these classifiers are discussed in detail in Section 6.4.1) analysis I focused on unique ISNs, not separating out cookies from backlog entries. I did this because the tendency to be classified as either bi-level or up-trending is interesting despite the cause. The SYN cookie should take into account the source port and IP, as well as the destination port and IP, and a notion of time. Therefore any tendency to be classified is still undesirable.

To the best of my knowledge no widely distributed operating system generates ISNs in the same manner as SYN cookies if the half-open connection will reside in the SYN backlog, though this has been proposed [77]. As a result, any lack of algorithmic complexity in SYN backlog ISNs is not due to the lack of entropy expected in SYN cookies. Aside from SYN cookies, it is possible that a server's implementation of the function that creates ISNs encodes data in some way. In this work I seek only to find ISNs that lack algorithmic complexity and possibly have mutual information in their ISNs—I do not attempt to determine the reason as to why.

6.2.3 Caveats

Two small details about my experimental methodology are important to understand, but had negligible effects on the data because they are rare cases:

- In order for my probes sent as part of a given experiment to one IP address to be independent and identically distributed (*i.i.d.*), I choose random source ports *with* replacement, meaning that there is a small chance that the same source port is repeated in an experiment. The RFC-compliant behavior in this scenario should be that the responses to identical SYN packets have ISNs that

increase with time. Because this happens infrequently (only when there are collisions in the 2^{16} source port space) it only affects the results in a negligible way (even when accounting for the birthday paradox).

- Because the focus of my study is on common failures in pseudorandom number generation (PRNG) and not on rare corner cases, I "flattened" all results by including only unique ISNs returned (including each unique ISN only once), and then applied thresholds to various experiments so that I am only considering data points where the machine at least attempted to send unique ISNs. A small number of machines (less than 1%) failed to return more than one unique sequence number. This is clearly a failure of ISN generation, but not the focus of my current study.

6.2.4 Limitations

I did not consider the cases where the system I am attempting to measure is actively returning data to subvert my scans. That is, there is a possibility that my scan is perceived as a malicious activity and the data received from scanning a server is purposely misleading. I make no attempt to check for this case and perform my analysis assuming this is the exception rather than the norm. I am also limited by the fact that my double point scan has IP addresses that originate from the same network and IP address range/subnet.

Another limitation is that I can only overestimate algorithmic complexity. That is, there is a possibility there are patterns I missed and the algorithmic complexity of a measured ISN set is worse than I estimate. This means that I err on the side of caution and the actual state of the Internet can be no better than my estimate, but is possibly worse.

6.3 Implementation

Before each scan to measure ISNs I first had to run a scan to see what IPs on the Internet were open on port 80 (ignoring any IPs that requested I not scan their network, see Section 6.6). To accomplish this, I used ZMap [42]. Once I knew what IPs were replying on port 80 (*i.e.* were alive), I proceeded with the main scan to measure ISNs (ISN scan). If a ZMap scan was more than a week old a new ZMap scan was performed to give a current view of the Internet.

The ISN scan was written in C. The code used to analyze and parse the pcaps created by each scan was written in C, python, and Julia. The NIST STS implementation used was Johnston's [37] python implementation. To make the data easily available, the parsed pcap information is stored in a psql database that is available upon request. The measurement machine has a direct link to the Internet backbone and is not filtered. Each scan produces between about 2-4 terabytes of pcaps (2 for a single point scan and 4 for the double point scan), therefore all pcaps are backed up to redundant storage and can be made available by request. To make the database small enough to be easily accessible the database holds the following information for each entry:

- ISNs returned
- Destination IP
- Source IP
- Date scanned
- Unique ID
- ISN array range in log_2

6.4 Analysis

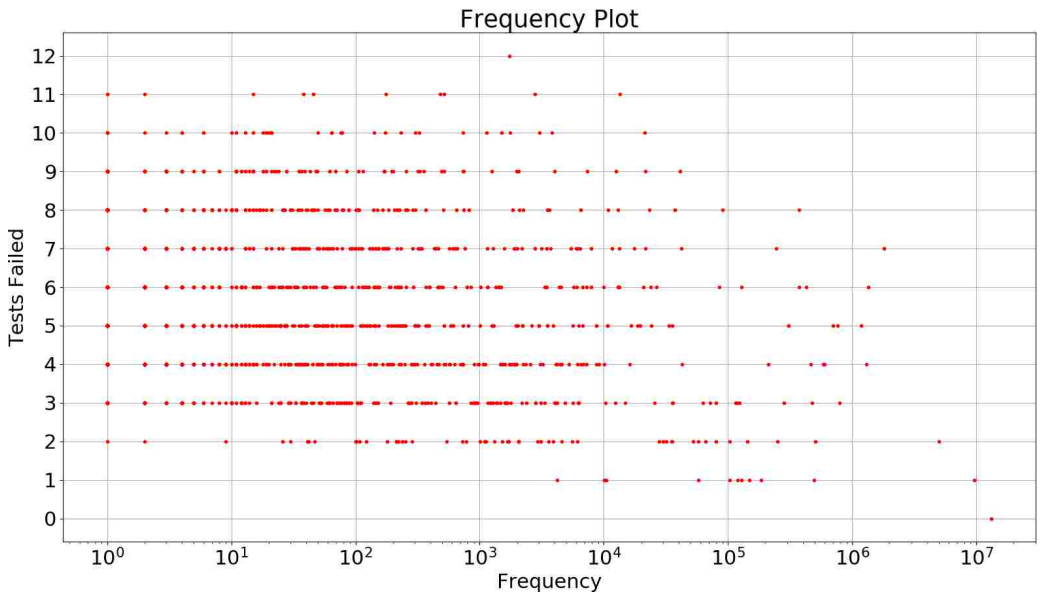


Figure 6.1: **Semi-log plot of pass-fail vectors for the single point scan.**

Due to my large data set, I decided to focus my attention on machines that appear to have low algorithmic complexity and are common. I might have studied the machines that have the worst algorithmic complexity, however they are in the minority and I chose to focus on modes of failure that affect the Internet as a whole more broadly. Among the ISNs that have the worst complexity were those that increased their ISN by a constant value or even those that had static ISNs. Although these were not the focus, they were included in the results of the range classifier described in 6.4.1.

The large amount of data points to examine caused me to use a top down approach when analyzing the data. I started by utilizing the NIST STS to get a general overview of the complexity of the ISNs collected. Next, I examined the data returned by the NIST STS and performed hierarchical clustering on a subset of the data that all failed the same set of tests. A subset was used due to the fact that hierarchical

clustering requires quadratic space and 1.8 million ISN sets would require an infeasible amount of computational resources. Then, by examining the results of clustering, I created classifiers that could be run on the entire data set. Finally, in order to find correlations when classifiers could not be utilized, I applied Pearson correlation.

6.4.1 Estimating Algorithmic Complexity

In order to estimate the number of machines that exhibit poor algorithmic complexity when choosing ISNs, I utilized the following methods as different lenses through which to view the data:

- National Institute of Standards and Technology Statistical Test Suite (NIST STS)
- Classifiers for two identified patterns (up-trending and bi-level)
- Pearson correlation

NIST Tests

The NIST STS is the standard test suite used to determine the "randomness," *i.e.*, algorithmic complexity, of a byte stream. The classifiers were created after I discovered many ISNs created patterns that had a discernible structure. Pearson correlation gives the linear correlation of two vectors as a value between one and negative one. The Pearson correlation is an interesting case because I can assume that any IP pairs that have ISNs with high Pearson correlation must have some underlying structure and thus must have poor algorithmic complexity. In other words, a lack of algorithmic complexity and mutual information are sometimes related.

Tests Failed	Feature Vector	Total IPs
0	[1,1,1,1,1,1,1,1,1,1,1]	13,248,072
1	[0,1,1,1,1,1,1,1,1,1,1]	9,658,797
2	[0,1,1,1,1,1,1,1,0,1,1]	5,000,456
3	[1,1,1,1,1,1,0,0,1,1,0]	796,388
4	[0,1,1,1,1,1,0,0,0,1,1]	1,300,015
5	[0,1,0,1,1,1,0,0,0,1,1]	1,187,689
6	[0,1,0,1,1,1,0,0,0,1,0]	1,350,120
7	[0,1,0,1,0,1,0,0,0,1,0]	1,814,211
8	[0,0,0,1,0,1,0,0,0,1,0]	372,172
9	[0,0,0,0,0,1,0,0,0,1,0]	40,908
10	[0,0,0,1,0,1,0,0,0,0,0]	21,400
11	[0,0,0,0,0,1,0,0,0,0,0]	13,416
12	[0,0,0,0,0,0,0,0,0,0,0]	1,735

Table 6.2: **Skyline results for the frequency plot figure.**

I used nine (9) of the NIST STS [116] tests. I could only use nine because having only 100 (in most cases) 32-bit integers did not allow me to use the full battery of the NIST STS tests, many of which require more data. Therefore I only used the tests from Table 6.3.

Column	**Test**
1	Monobit
2	Frequency within block
3	Runs
4	Longest run ones in a block
5	DFT
6	Non overlapping template matching
7	Serial
8	Approximate entropy
9	Cumulative sums

Table 6.3: **NIST STS tests used.**

In addition to the NIST tests, I added three more tests which I surmised would allow me to discern interesting information. The first test simply checks to see if

the data is in strictly increasing or decreasing order, and fails if so. The second test checks whether the vector of ISNs are mostly unordered. It does this by counting the number of elements that are greater than the previous element in the sequence. If this number is less than 1/3 or greater than 2/3 the test fails. The last tests checks the number of bits of entropy in the range of ISNs returned. It does this by calculating log_2(MAXVAL - MINVAL). For a significantly long sequence of randomly chosen 32-bit numbers this should be close to 32, therefore any sequences that have a value less than 31 fail. The NIST tests need minimum amounts of data to be effective, therefore I did not attempt to run them on any ISN arrays that had less than 200 values, with the expectation that many of them had to be retransmissions. This allowed me to run the NIST tests on 45,389,737 values, or approximately 75% of the data. In the event that the NIST test did not have enough data to complete, the test was counted as a fail.

Once the first battery of tests (NIST plus a few custom tests) was run I created a boolean vector of pass-fail values for each ISN sequence associated with an IP address. This allows me to group together IP addresses that fail tests in the exact same way. Figure 6.1 is semi-log plot of these vectors for the single point scan. The x-axis is the number of tests that fail with the same pass-fail vector, and the y-axis is the number of tests failed. The most interesting points on this plot are along the skyline (the rightmost point of each row, where many data points fail the same number of tests in the same way). A detailed table of the skyline is given in Table 6.2, where the feature vector is given along with the exact number of IP addresses that fail with that feature vector. The column field from Table 6.3 relates to the columns in the feature vectors. The last three columns are the added tests described earlier, respectively.

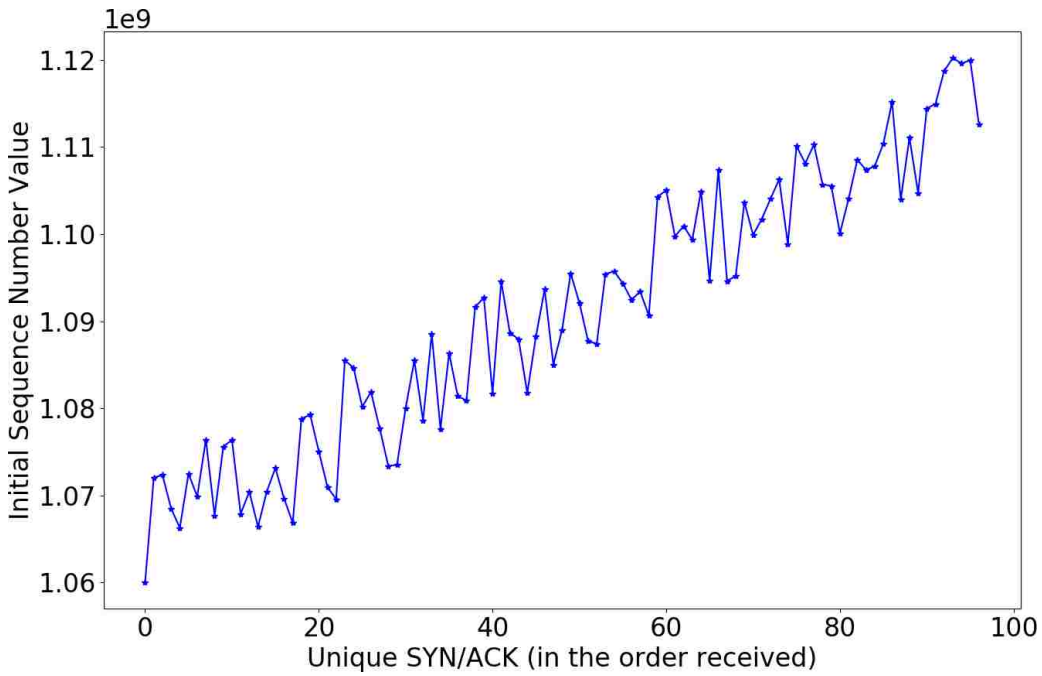


Figure 6.2: **Up-trending Initial Sequence Numbers. All duplicate ISNs were removed in this plot.**

Classifiers

Upon visually examining the data along the skyline, I considered the ISNs in the rightmost point where seven tests were failed (seven on the y-axis). This point was chosen because it is indicative of many ISNs failing the exact same test and is where I expect to find many ISNs failing for similar reasons. To gain an initial understanding of the data as a whole, I employed standard data mining tools. I randomly sampled a set of 5000 ISNs from this group. I then performed hierarchical clustering of the ISNs using the single linkage technique in Euclidean space. Finally I visualized the dendrogram and identified a small number of large clusters. Upon further examination of the clusters, I identified two patterns as being particularly common. One was a bi-level pattern where the ISN seemed to come from two different ranges (see Figure 6.3). The other was an up-trending pattern where the overall

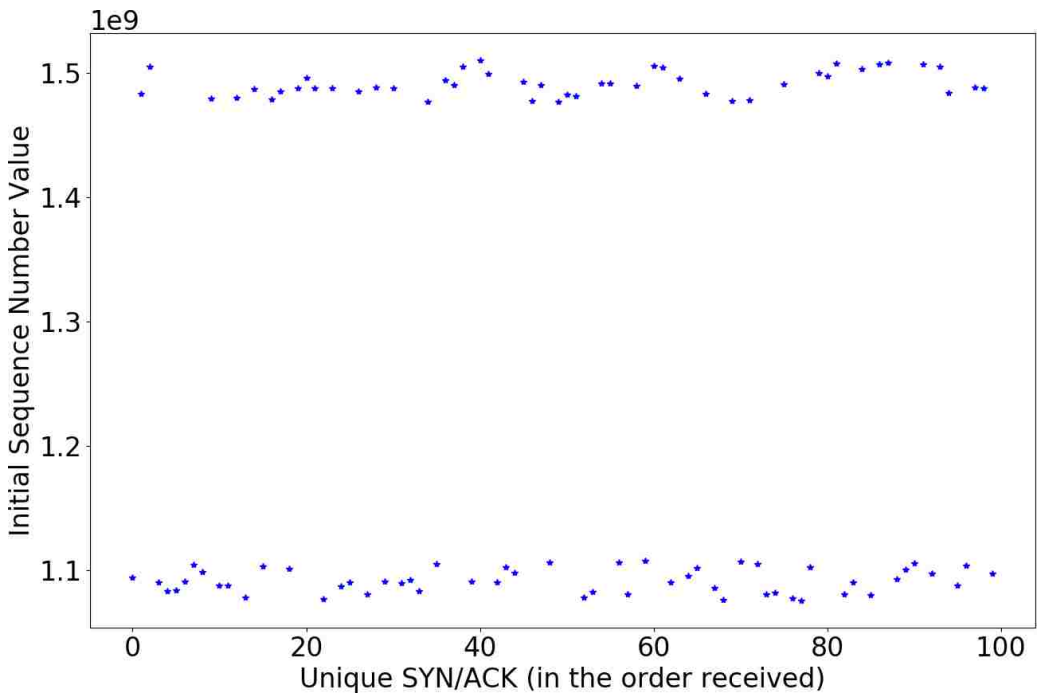


Figure 6.3: **Bi-level Initial Sequence Number. All duplicate ISNs were removed in this plot.**

ISNs appear to increase. In this pattern each ISN was not necessarily increasing with respect to the previous point, however the overall trend was to increase over time (see Figure 6.2). All figures of ISNs classified as one of the two classes were created by removing any duplicate ISNs. As mentioned before, duplicate ISNs are expected in connections that are part of the TCP backlog (see Section 6.2.2.)

I built classifiers for each of these sequence types and ran them on the data collected from one of the vantage points used for the double IP scan. This was done because I wanted to see if there was mutual information between the IPs later in the analysis. About 9.5% of the IP addresses that respond to port 80 fall into one of these two classes, 4,653,395 of which are bi-level and 1,181,853 are up-trending. The classifiers were not without some overlap, with a small fraction (876) classifying as both up-trending and bi-level. These overlapping classifications appeared, when

examined, to share the same pattern of being bi-level but starting at one distribution and transitioning to the second distribution bouncing between the two roughly in the middle (see Figure 6.4). I do not require the sets to be disjoint, only that they illuminate a lack of algorithmic complexity where it exists. Properly implemented ISN creation algorithms are astronomically unlikely to match any of the classifiers.

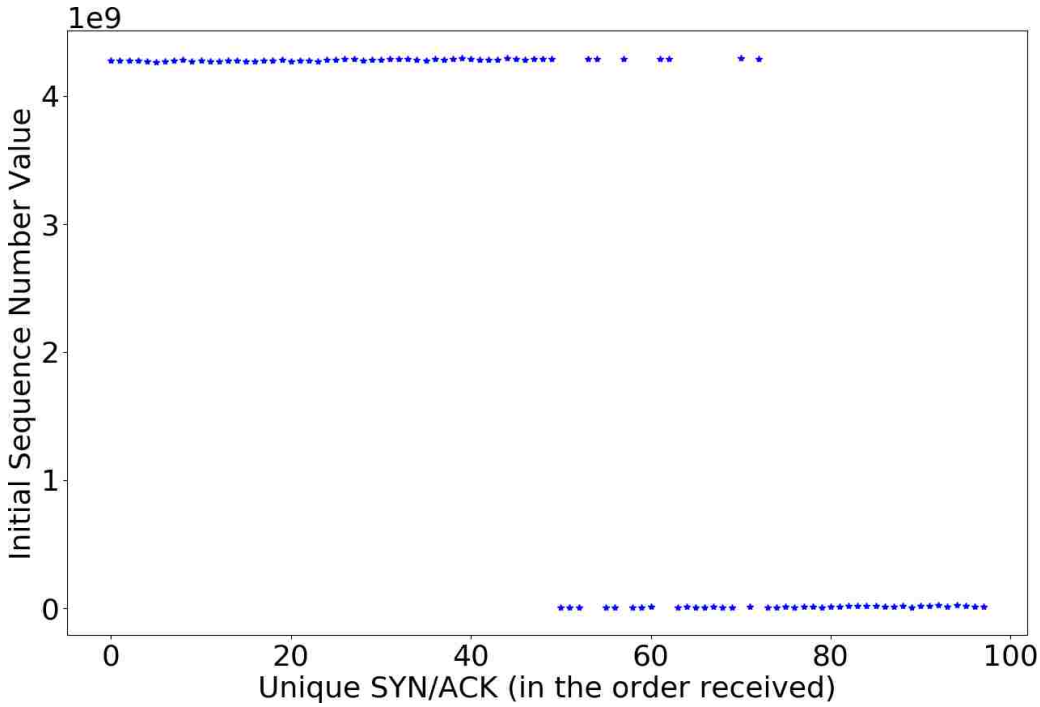


Figure 6.4: **Matches both classifiers. All duplicate ISNs were removed in this plot.**

A note on classifiers and mutual information: Mutual information will be discussed in Section 6.4.2, but here I make a note about the classifiers and mutual information. I tested both types of patterns to see if the patterns were indicative of mutual information existing between IPs connected to the corresponding servers. I noted many of these have mutual information and knowing the ISN of one measurement machine greatly reduces the search space for guessing the ISN of the other. Of the 4,653,395 ISN sets classified as bi-level, 4,529,908 had mutual information based

on my metric. Of the bi-level sets tested only 123,432 were determined to not have mutual information, meaning about 97% of bi-level sets appeared to have mutual information. For the up-trending class I found 971,703 had mutual information based on my metric, and 209,825 I identified as having no mutual information. I note a small caveat that not all IP addresses were found in both scans and I could only test those that returned ISNs to both vantage points.

Other patterns discovered but not investigated further: Other types of patterns were found in the data such as sawtooth (see Figure 6.5) patterns, however, they were not as common as the bi-level or up-trending patterns, totaling in at only 54,202 instances. I suspect that the majority of sawtooth graphs also have mutual information as I found them when looking at ISNs with high Pearson correlation. The values found in the sawtooth patterns are not repeating, as all duplicates were removed before plotting. Due to the fact that the majority of patterns were either up-trending or bi-level I focused my findings on these two more common classes.

Entropy range

The fact that I could classify ISN sets is indicative of an underlying structure in ISN creation. That is, any graphs that fall into these two classes lack the algorithmic complexity I expect to see in ISN creation. For example Figure 6.6 is a plot of an ISN set that passed every NIST STS test. The Figure's range nearly spans the whole ISN range of 2^{32}, or roughly 4 billion. In addition to the expected range, there is no discernible pattern like those found in Figures 6.2, 6.3, 6.4 and 6.5. As seen in Table 6.2, a little over 13 million IPs returned ISNs that passed every NIST STS test and likely have the necessary algorithmic complexity.

In addition to these two shapes as classifiers I also classified the data based on the *entropy range* (log_2(MAXVAL - MINVAL)). I define the entropy range as a naive

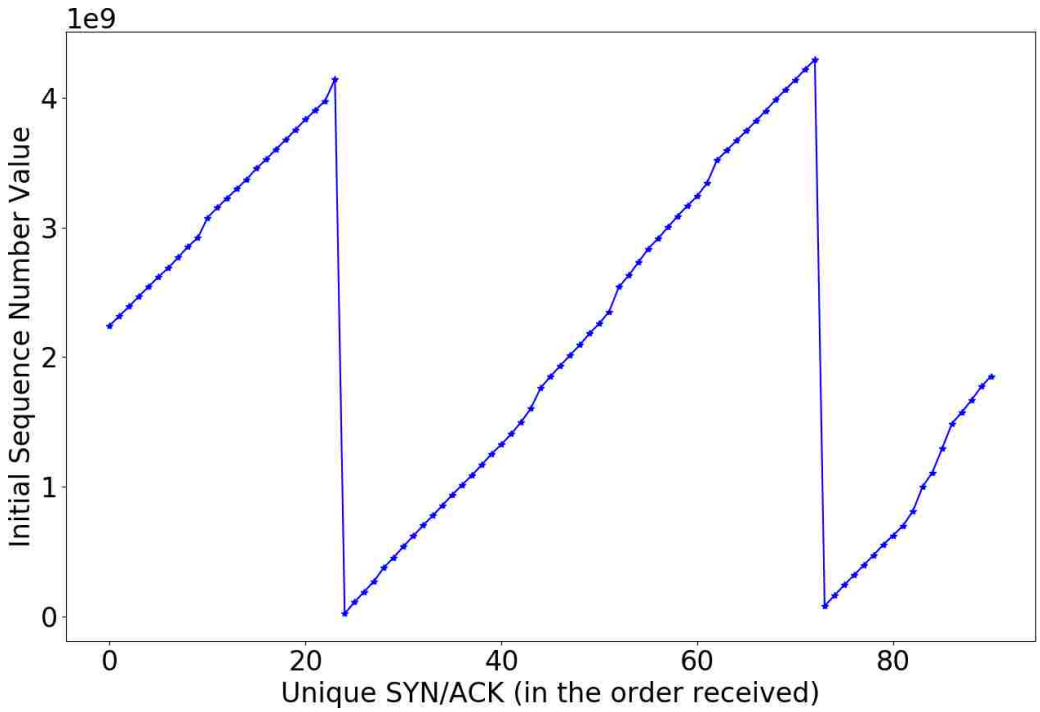


Figure 6.5: **Sawtooth Initial Sequence Numbers. All duplicate ISNs were removed in this plot.**

classifier that calculates an upper bound on algorithmic complexity based simply on the range of values. I have four different levels based on the number of guesses one might need to randomly guess the range, which I call r12, r16, r20, and r26. According to Zalewski [133] if the number of guesses is 5000 or less the ISN is easy to guess with common networking hardware. Therefore, if the entropy of the range was less than 2^{12} I consider the algorithmic complexity to be poor and classify it as r12. Zalewski also noted that guessing the ISN in less than 60,000 guesses is feasible, albeit harder. With this I made a different class such that the entropy range was greater than r12 and less than 2^{16}, denoted as r16. Although 2^{12} is not exactly 5000 and 2^{16} is not exactly 60,000, these were the closest powers of two and are estimates in any case.

Zalewski's work was performed at a time when gigabit connections were not so

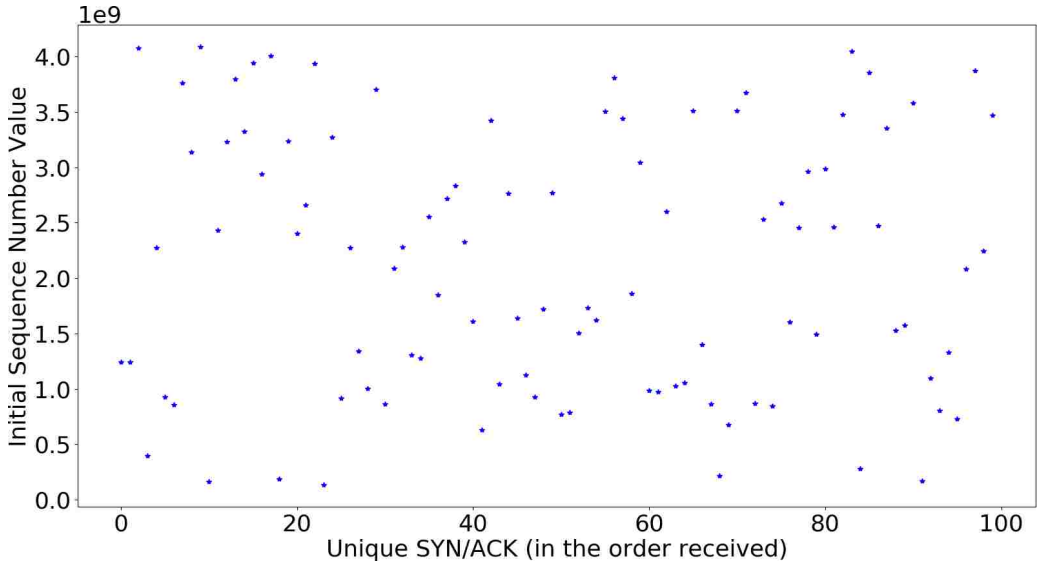


Figure 6.6: **Ideal ISN set where ISNs passed all NIST STS tests. All duplicate ISNs were removed in this plot.**

readily available to the public, but I believe that my threat model should include an adversary with a gigabit connection. Therefore the following two classes are based off of attacks possible assuming a gigabit connection. If I assume a gigabit connection and the fact that a blind injection attack would use about 100 bytes per packet, I calculate that an attacker could perform a *blind injection* attack in one second if the range were r20 or below (see Table 6.4). Given that an attacker can take as much time executing a *blind spoofing* attack as they need, I set the last cutoff at r26, giving an attacker about four and a quarter minutes to execute the attack. Note that SYN cookies are equivalent to r24 (see 6.2.2) and are susceptible to *blind spoofing* attacks in about 64 seconds.

Classifier Implementation

The up-trending classifier was constructed based on the following python code:

	Attack Type		
Time in seconds	Blind RST	Blind Injection	Blind Spoofing
	68 bytes	100 bytes	400 bytes
1	20.8	20.53	18.253
2	21.8	21.53	19.253
4	22.8	22.53	20.253
8	23.8	23.53	21.253
16	24.8	24.53	22.253
32	25.8	25.53	23.253
64	26.8	26.53	24.253
128	27.8	27.53	25.253
256	28.8	28.53	26.253
512	29.8	29.53	27.253

Table 6.4: **Attack range chart.**

```
def uptrend(seq):
    zn = zNorm(seq)
    t = np.array(range(1, len(zn) + 1))
    a = np.polyfit(t, zn, 1)
    b = a[0] * t + a[1]
    score = sum(np.power((zn - b), 2))
    return score < 50
```

This code works by first standardizing the ISNs using z-normalization, which essentially converts the data to have zero mean and one standard deviation. Then a line to fit to the normalized data set is calculated using the polyfit function which returns a polynomial. Next, a vector is created that is the first `n` values of the line where `n` is the length of `seq`. The score is the Euclidean norm of the error in the fit from the data. The score is calculated by taking the vector containing the calculated line that fits the data and subtracting it from the normalized data, squaring the errors and finally, summing the squared errors. Finally, the function returns true if the score is less than 50, which is equivalent to Pearson's correlation coefficient of

0.75. The bi-level classifier is calculated by making sure the data passes a battery of tests:

1. Gap between levels at least 33 million
2. The data is split close to 50 / 50 between the levels ± 20
3. The range of each bi-level must be less than 2^{25}
4. There must be multiple transitions from level to level (> 10)

Essentially, this battery of tests tries to ensure that I have two ranges of data that the ISN was chosen from. To ensure the gap is not by chance, Item 1 is used. This gap is the smallest gap between the two sets allowed. Item 2 assures ISNs are picked from both distributions with equal probability. Item 3 makes sure that the gap of each level is not so large that the desired shape is lost. Finally, Item 4 enforces the expectation of randomness expected from Item 2. Each of these tests alone are insufficient, but together they decrease the likelihood of any non bi-level graphs being falsely labeled.

The range classifiers were run on values that I determined not to be SYN cookies. That is, I only performed the test on ISN values that repeated in the set (*i.e.*, were retransmissions). This means that the classifier was only performed on ISN sets that had at least two unique repeating values (although the vast majority were 100).

The values used for each classifier were selected empirically by studying a set of known graphs classified by hierarchical clustering. The classifiers are empirically designed to fit exactly the structure they are intended to classify so it would not make sense to categorize data points as, *e.g.*, false negatives or false positives. However, by informally visualizing the classified data I found the classifier output to match my intuitions.

Class	Occurrences
r12	40,666
r16	22,384
r20	206,139
r26	6,888,938

Table 6.5: **Entropy range classifier results. Data points are put into exactly one class.**

6.4.2 Mutual Information

Mutual information allows one host to know a property of another host based only on a property they have themselves. I seek to identify mutual information in ISNs between two clients on the Internet connected to the same host. The existence of mutual information is indicative of an information leak, or side channel that can be used to estimate the ISNs of other clients with which you have no contact with, but are connected to the same server/host. Mutual information was discovered in the data set using two different methods to analyze the double point scan data: Classifiers and Pearson Coefficients.

Mutual information in classified patterns

To verify the existence of mutual information leveraging the bi-level and up-trending classifiers, I first ran the classifiers on the data acquired from both vantage points in the double point scan (only looking for mutual information between two clients probing the same host). Next I compared features of the parameters used by each classifier requiring that they met specific criteria in order to be considered as containing mutual information. These features are specific to each classifier type and are as follows:

For bi-level graphs I looked for two different indicators when determining if ISNs

have mutual information. Both indicators must be met:

- Range levels: the ranges of both levels are the same ±5000000
- Gap length: log_2 of the gap between the levels are the same ± .28.

For up-trending graphs I first removed any amplitude based similarities by calculating the z-normalization ($z_i = (x_i - \mu)/\sigma$) of the ISNs, then fit a line to the data and checked the following indicators. As before both indicators must be the satisfied.

- Slopes: the slopes must be the same ±.01
- y-intercepts: the y-intercepts must be within ±.1.

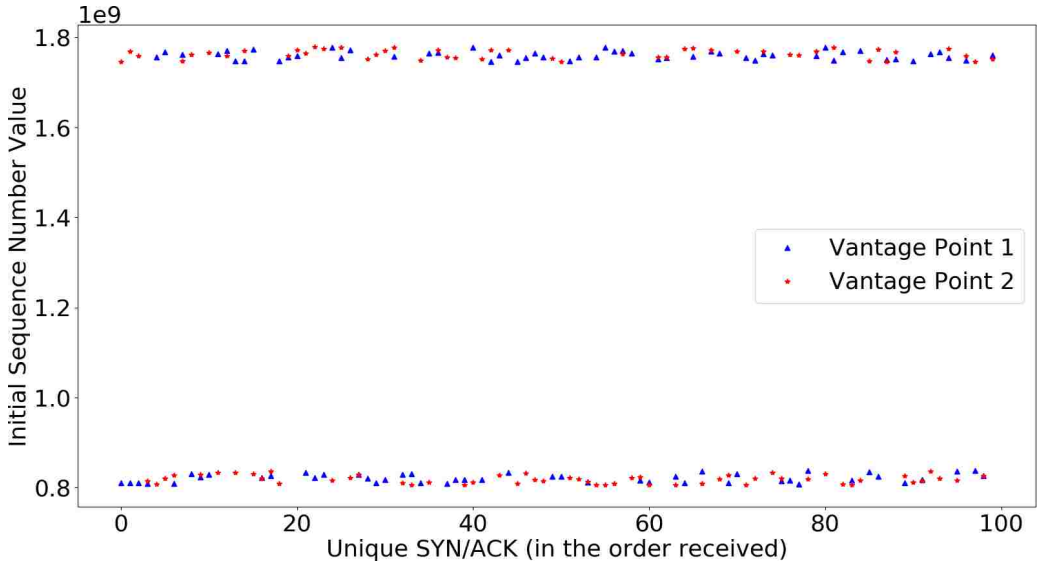


Figure 6.7: **Mutual information in a bi-level graph.**

Graphically I can see in Figure 6.7 that both vantage points have ISNs that come from the same distributions. Thus by knowing the distribution of ISNs from vantage point 1 I know that the ISNs from vantage point 2 will have a similar distribution. For

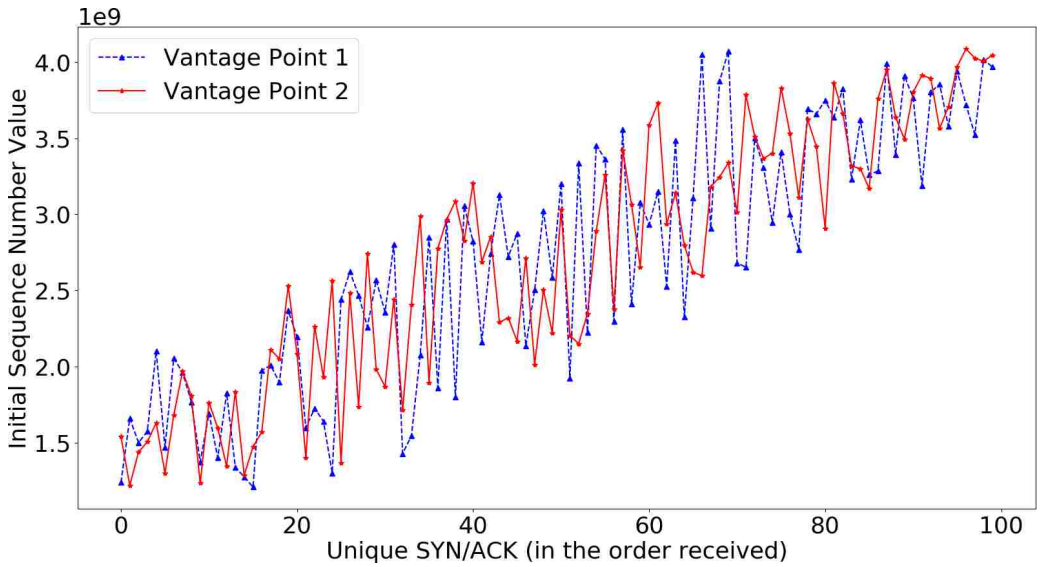


Figure 6.8: **Mutual information in an up-trending graph.**

up-trending ISN sets with mutual information I see something similar (see Figure 6.8). The information from vantage point 1 for these ISNs can give an attacker knowledge of the slope of the line that fits the data and allows one to make an educated guess of the range from which the ISNs will be chosen for different vantage points. In both cases it is apparent that the RFC is not followed and it is likely the case that the recommended values that should be used to generate an ISN are not used.

The benefit to finding mutual information this way as opposed to using the Pearson Coefficient, is that the ISN sets I am comparing to one another do not need to have the same length.

Mutual information *via* Pearson Coefficients

To further examine the mutual information between the ISNs of two clients I calculated the Pearson Coefficient of all ISNs from the double point scan that returned

the same number of unique ISNs. Of the roughly 60 million IP addresses, about 39 million received the same number of unique ISNs during the scan allowing me the ability to calculate the Pearson Coefficient for about 2/3 of the data. Of the roughly 39 million data points where I could apply the Pearson Coefficient, 4383 had r values of 1. This means that they are directly correlated and as one increases the other either increases or decreases. An r value of 1 basically tells me that the ISN is increasing by a constant value (wrapping may occur). Values of r close to one corresponded to ISNs that increase by a relatively constant value, but that value may change over time.

Using only the ISNs with an r value greater than or equal to 0.9, I ran the data through a basic neural network. The neural network had three layers and each layer was initialized with linear activation. I trained my neural network using the data from one IP's ISNs as the training data input and the other IP's ISNs as the training data's desired output. I trained on 95% of the data and calculated the root-mean-square-error (RMSE) values based on the predictions for the last 5%.

With this data I created Figure 6.9 illustrating the potential reduction in algorithmic complexity using a simple heuristic. On the x-axis I have the RMSE value returned by the prediction. On the y-axis I have log_2 of the range of data points. I assume that the range of the data points seen is a rough approximation for the number of guesses one would naively have to attempt before correctly guessing the next point. The red points I claim are within attack range return RMSE values below 2^{16} using the same metric as described at the end of Section 6.4.1. In this graph all points have mutual information of varying degrees, given by its r value. Although all points in red are within the attack range, points that are towards the top left have the most reduction in algorithmic complexity. In contrast, points that are close to or on the slope that cuts the graph in half diagonally, have little to no reduction. Note that many points in the top left blue area still have a great reduc-

tion in algorithmic complexity, some showing nearly 14 bits of entropy reduction. In total, 309,478 points have an r value greater than or equal to 90%. Of those points 83,751 are within attack range after running the neural network. All but 340 have a reduction in algorithmic complexity by at least one bit, 184,517 have a reduction of 5 bits, and 25,118 a reduction of 10 or more bits. A reduction of 5 bits for example would bring an IP with a data range of 20 bits down to 15 bits, which would bring it down to attack range.

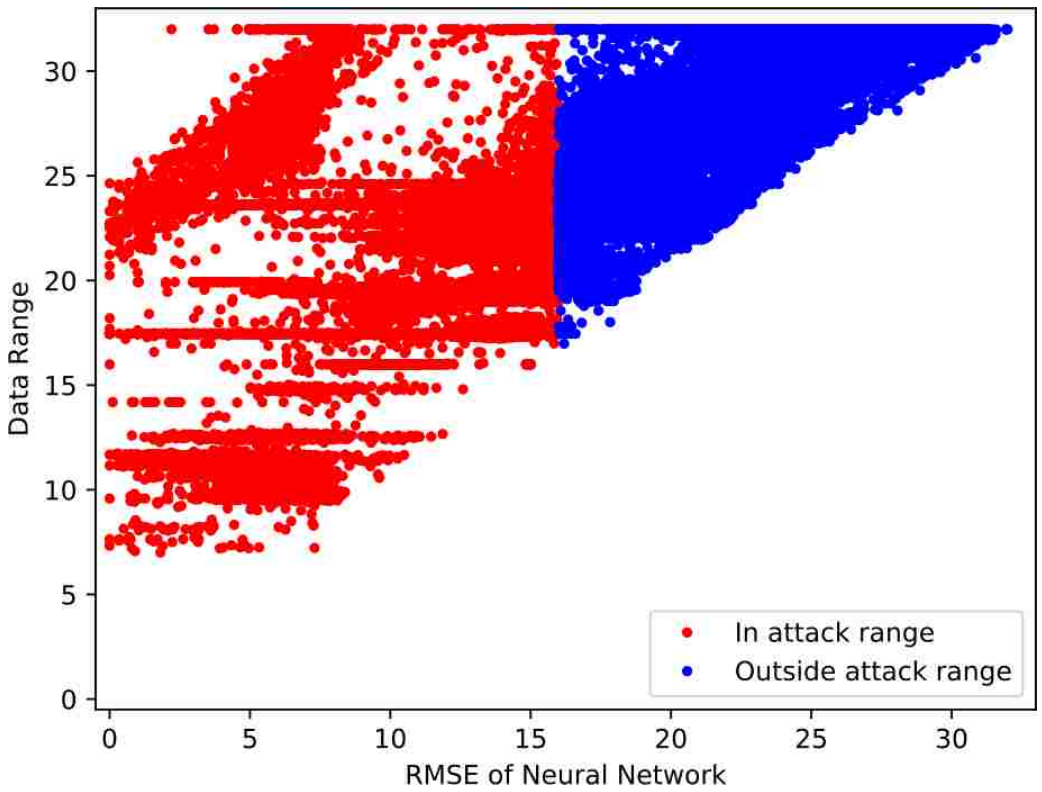


Figure 6.9: Log_2 **of RMSE** ***vs.*** log_2 **of the data range.**

The benefit of using the Pearson Coefficient is that I do not need the data to fit one of the patterns identified by the classifiers.

6.5 Summary of Results

In this section I summarize my results with respect to both the reductions I see in algorithmic complexity and the mutual information present in ISNs.

6.5.1 Reductions in Algorithmic Complexity

Lens	Count
r12-r24	2,525,538
Bi-level	4,653,395
Up-trending	1,181,853
Pearson Correlation	251,105
Total unique	8,222,988

Table 6.6: **Complexity reductions.**

Table 6.6 shows all the reductions to algorithmic complexity I can find in the data set, whether it is useful in an attack or not. As I have noted, Table 6.4 shows that *blind spoofing* attacks are viable in 256 seconds on any server that has ISNs with a range less than or equal to 79,972,463 or r26.253. By taking all the lenses used to measure algorithmic complexity and picking a suitable method for entropy reduction I can conservatively calculate how many IPs generate ISNs for each r-value. The graph representing this data is shown in Figure 6.10. For the purpose of simplicity I rounded all r-values to the nearest whole number. As can be seen, approximately 11.5% of my data points fall in the r26 bucket or below. This equates to 11.5% of the TCP IPv4 traffic on port 80 being susceptible to *blind spoofing* attacks. I analyzed these data points to determine (based on retransmissions) if they are SYN backlog entries or SYN cookies, only 5.3% of these vulnerable machines sent SYN cookies. Since some types of attack would require not just a reduction in algorithmic complexity, but also mutual information, I verified that it is almost always the case that mutual information is present for the data in Figure 6.10.

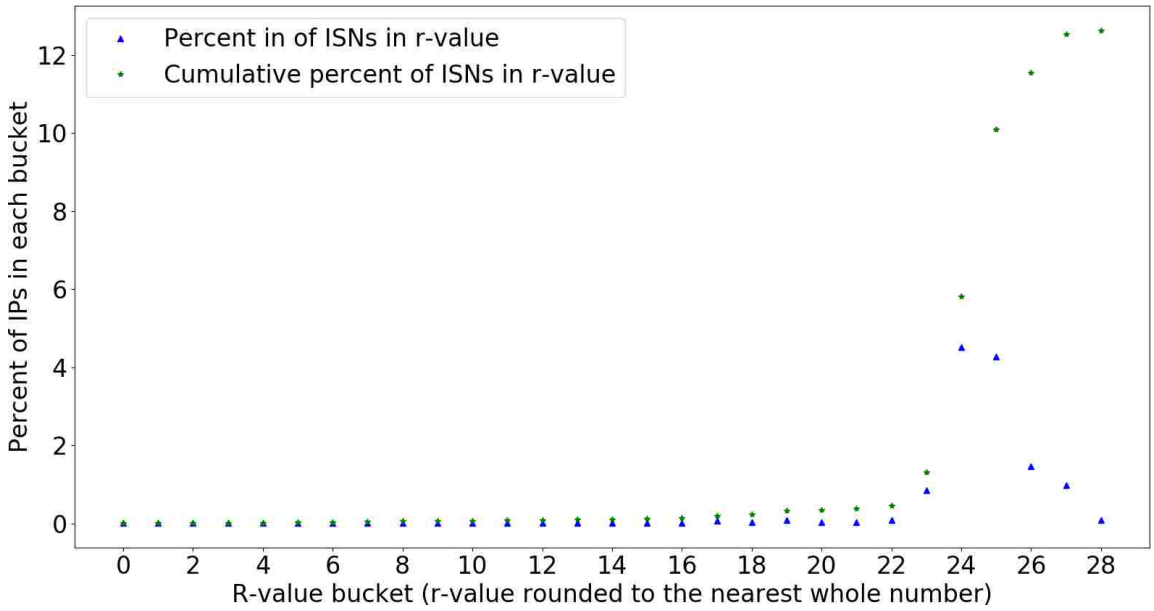


Figure 6.10: **Percent of ISNs in each r-value.**

6.5.2 Mutual Information

All 309,478 ISN sets found with the Pearson correlation method have mutual information as that is the implication of having a high r value. In addition most of the ISN sets classified as bi-level or up-trending were also found to have mutual information. See Table 6.7 for a summary of the mutual information results. I would like to re-iterate that the presence of both mutual information as well as the lack of algorithmic complexity is very dangerous with regards to vulnerability.

Lens	**Count**
Pearson Correlation	309,478
Bi-level	4,529,908
Up-trending	971,703
Total unique	5,607,745

Table 6.7: **Results with mutual information.**

6.6 Ethical Considerations

All IPs I test from have a DNS entry that resolves to `this-weird-ip-traffic-is-for-research-measurements.cs.unm.edu` where I have a web page explaining various projects performed on the research network. The page has the contact information of the researcher(s) associated with each project. Additionally, all abuse complaints related to IPs on the research network are forwarded to me based on an MOU with the backbone network operators. Because many networks on the Internet may not wish to be scanned, I maintain a do-not-scan list containing subnets that my scanner should not target. All abuse complaints are responded to by email where I explain I am doing Internet measurement research. If an IP address or network is given in the initial abuse complaint, it is promptly added to the do-not-scan list. Otherwise, I ask what IP address(es) the complainant would like me to add to the list and cease scanning them for all future measurements. In order to give network administrators enough time to analyze the traffic and contact us, I waited at least 24 hours between any subsequent ISN scans.

The system causes no more than the usual risk for denial-of-service associated with having a public facing IP on the Internet, as modern operating systems are designed to handle on the order of hundreds of connections per second and can be configured to handle thousands.

6.7 Related Work

In addition to the closely related works that have been discussed throughout the chapter, there are two general areas of research that are closely related to my study that were not discussed already: TCP/IP hijacking attacks and large-scale Internet measurement for vulnerabilities.

6.7.1 TCP/IP Hijacking Attacks

Qian and Mao [101] did early work exploiting weaknesses in TCP sequence numbers to perform off-path hijacking. Through off-path hijacking of TCP connections they were able to inject malicious content into connections. The attack was contingent on both sequence-number-checking firewall middleboxes and unprivileged malware on the victim machine.

Subsequent work exploiting ISNs to hijack attacks was performed by Qian *et al.* [102], in which they were able to remove the limitation of sequence-number-checking firewall middleboxes. In it they leveraged unprivileged malware to quickly determine sequence numbers, in order to perform TCP injection and hijack attacks.

Following the work of [102, 101], Gilad *et al.* [57] were able to remove the constraint of malware installed on the victim in their work "Off-path TCP injection attacks." Their attack had a weaker assumption only requiring the victim to visit a malicious website as opposed to malware installed and running on the victim's machine. The malicious website performed a web cache poisoning exploit, granting them the information necessary for the attack. Despite this weaker constraint, they were still able to perform TCP injection attacks to off-path victims.

After the release of RFC 5961, Cao *et al.* [15, 16] were able to finally remove any requirements of unprivileged malware running on the victim machine or the entering of malicious websites. In this work they instead were able to rely on a side channel in Linux's implementation of Challenge ACKs as introduced by RFC 5961 [107] to determine ISNs and ultimately perform off-path TCP injection attacks. As stated in Section 6.1.3, this side channel was mitigated and patches have been released for the Linux kernel. In addition, this work was only applicable to Linux, as they were the only major distribution that attempted to follow the RFC faithfully.

As seen before [33, 123], sequence number issues affect protocols other than TCP.

Munir *et al.* [94] did work analyzing weaknesses in Multipath TCP (MPTCP). In it they were able to leverage mutual information between sub-flows in local and global sequence numbers in MPTCP headers, from the perspective of an attacker eavesdropping on one MPTCP sub-flow. Using this weakness, as well as a vulnerability that allows one to stall a sub-flow, they were able to perform a traffic diversion and hijack attack.

A recent attack that can be used to infer TCP sequence numbers is described by Chen *et al.* [20]. The attack is possible only on wireless connections (specifically IEEE 802.11), and is due to a side channel in the protocol itself. This means that this particular attack relies more on the hardware of the access point the victim is connected to as opposed to specifics of the victim's software. This vulnerability allows an attacker to perform a blind off-path attack that does web cache poisoning in a matter of minutes.

6.7.2 Large-scale Internet Studies

One of the first papers to perform a large-scale Internet measurement was Paxson's seminal paper "End-to-end Internet packet dynamics" [99]. He was able to capture and verify network pathologies and other interesting features of the Internet, by simply performing and analyzing network scans in both directions. That is, he traced the connection from client A to client B *and* client B to client A.

Another large-scale study was that of Weaver *et al.* [128], which detected and analyzed forged TCP Reset Packets. Their data set contained 30.2 million TCP flows collected over 19 hours. Using this passive network measurement data they were able to identify machines that were injecting RSTs into active connections.

A large scale Internet study involving inadequate randomness in TLS certificates and SSH hosts was done by Heninger *et al.* [62]. This work found insecure PRNGs

on the Internet that caused the creation of vulnerable RSA and DSA keys, used by TLS and SSH respectively. Their results include the discovery of SSH hosts that use the same keys as other SSH hosts and TLS hosts that use manufacturer default keys. They also found vulnerable hosts for which they could compute private keys in 0.5% of TLS and 1.06% of SSH hosts in their data, by exploiting known weaknesses in RSA and DSA.

Quach *et al.* [104] monitored the Alexa top 100 sites daily for 6 months checking to see when/if the vulnerability discovered in [15, 16] would be patched.

6.8 Summary

I was able to find cases where there was a lack of algorithmic complexity as well as mutual information in ISN creation on the Internet. I demonstrated multiple ways to find mutual information and a lack of algorithmic complexity in ISNs, through Pearson correlation, classifiers, and the NIST-STS. I can be certain that the RFCs are not followed in most (if not all) of the ISNs that have poor algorithmic complexity. I suspect that the clock described in 6.1.1 may be influencing the ISNs in IPs that are up-trending. It is also possible that the `M` counter is used by ISNs that have the sawtooth structure as well.

More work is needed to fully understand failures in TCP ISN generation at the scale of the entire Internet. My tests were conducted under the most ideal of conditions under scenarios well described by RFCs, and still a large number of machines were found to be vulnerable. In future work I would like to test how machines respond to similar tests, but vary other fields in the TCP header. For example, the sender's ISN could be kept static, or the source port could be kept static. Based on the RFCs, the behavior of the later should be something similar to an incrementing clock, however, as I have seen, the RFCs are not always adhered to and I believe

these cases are worth examining. Generally speaking, less than a quarter of the machines scanned passed the full battery of tests for algorithmic complexity. Others may have failed specific tests by chance, but all of the data might have undiscovered structure in it, especially the other roughly three quarters that failed at least one test.

My main result is that 11.5% of the hosts that listen on port 80 on the Internet generate ISNs in a way that is vulnerable to blind spoofing attacks, with only 5.3% of this 11.5% explained by SYN cookies. Because a lack of algorithmic complexity and mutual information can both take many forms, and because there are theoretical limitations preventing me from knowing that something has at least a certain amount of algorithmic complexity, my results should be taken as a lower bound on how many machines on the Internet do a poor job of TCP ISN creation. In future work, I plan to further explore other modes of failure in this space as well as develop OS fingerprinting methods suitable for large-scale summarization of what types of OSes/devices are the main culprits.

6.9 Chapter Acknowledgments

This material is based on research supported by the U.S. National Science Foundation under grant Nos. #1518878 and #1518523. The work in this chapter involving hierarchical clustering and the creation of the dendrogram was performed by Mueen Abdullah. The use of the word 'I' was maintained to preserve the flow for the reader.

Chapter 7

Conclusion and Future Work

The need to use reverse engineering to extract ephemeral secrets from binary objects and dynamic systems is growing as they are increasingly: interacting with online servers, packed, only existing in memory, or running remotely. My work confirms that even under these difficult conditions, with the aid of dynamic analysis and automation, ephemeral secrets in these types of domains can still be obtained. The work in this dissertation exemplifies the kind of reverse engineering tasks possible when ephemeral secrets are explicitly considered when analyzing a reverse engineering task. It also accentuates that there are many times when ephemeral secrets are desired from remote dynamic systems, such as discovering a censorship list in China, where traditional reverse engineering techniques are insufficient. Although the traditional reverse engineering techniques are lacking, the work in this dissertation demonstrates that ephemeral secrets are accessible by combining the common techniques of dynamic analysis and automation.

Dynamic analysis is central to the process because ephemeral secrets can only be measured while a system is running. Automation, on the other hand, is essential to the process due to the fact that the secret is often changing with each interaction of

the dynamic system (such as the derivation of ephemeral keys) or the secret is being discovered in pieces (such is the case in Chinese censorship measurement). This necessitates a controlled repetition, making automation paramount. By using these techniques in unison, the art of extracting ephemeral secrets is possible. Furthermore, the added benefit of using automation and dynamic analysis in unison is that they allow a reverse engineer the ability to extract ephemeral secrets from systems to which they may have only partial access. This may remove the need for direct access to static objects, allowing the uncovering of ephemeral secrets in systems reverse engineers classically did not consider *reverse engineering* tasks.

In this dissertation I have demonstrated how automation and dynamic analysis enable reverse engineers the ability to extract ephemeral secrets from dynamic systems. The composition of automation and dynamic analysis was a necessary and powerful combination employed in each work presented. Together they gave me the ability to retrieve the respective ephemeral secret from each dynamic system. I have demonstrated the nature of ephemeral secrets and how they take on many forms, such as: a value, a list of values, or even a process to generate a value in a dynamic system. Thus establishing the discovery of ephemeral secrets as crucial to the future of reverse engineering dynamic systems. I have demonstrated that despite the form of the ephemeral secret, automation and dynamic analysis are key components in the reverse engineering process to uncover the ephemeral secret.

For example, this dissertation presents four novel techniques that utilize automation and dynamic analysis to extract ephemeral secrets. The system described in Chapter 5 is generalizable and can be applied to different languages, or even the probing of dynamic systems for other types of ephemeral secrets (*e.g.*, fluctuations in online shopping prices, other censorship mechanism, *etc.*). The system from Chapter 4 can be used to track any output, or even any region of memory of interest to a reverse engineer. Because it was the first DIFT system to track indirect flows in

a general way, it is the first DIFT system that can be used to discover ephemeral objects in memory. In Chapter 5 the technique described to perform a man-in-the-middle attack despite certificate pinning (what I call an ephemeral key dump) can be used on any application that performs certificate pinning. The work in Chapter 6, is generalizable to scan and study other discrepancies between Internet RFCs and server's implementations of them.

Although ephemeral secrets are prevalent in more domains than presented in this dissertation, this dissertation covers a variety of applications where ephemeral secrets are successfully retrieved. Using the exemplary techniques to uncover ephemeral secrets described in this dissertation, reverse engineering can be performed in new domains.

7.1 Future Work

Future work can be broken up into two categories: extensions of work presented in this dissertation, and new applications where ephemeral secrets are present.

7.1.1 Extensions of Work

A natural extension of the Chinese censorship work from Chapter 3 would be to apply the technique to other languages and regions. My technique presented to study censorship would be an extremely powerful tool to aid in censorship measurement systems such as the Open Observatory of Network Interference [96] (OONI) probe. One could supply the OONI project with a framework that requires: a list of news sites to read, a named entity extractor for the given language, and a list of sites on which to test the named entities. This would allow the OONI probe to test potentially new sensitive words, as opposed to a list of human maintained keywords.

The V-DIFT work in Chapter 4 can be extended to other architectures (such as ARM or 64-bit x86), or even used to create a similarity metric between two programs. V-DIFT could also be applied to the reverse engineering process of malware and proprietary software to track the origin of any data in the program. For example, one could take a trace of a program up to a point of interest, mark the data that is of importance, and step backwards through the trace to discover how the data flowed in the program up until that point. This could potentially highlight where a program derived any value, making it easy for a reverse engineer to quickly locate in a program keys, keywords, magic values, or any relevant data.

The natural extension of the LINE work from Chapter 5 would be to find other applications that perform certificate pinning, and apply the ephemeral key dump method. This would allow a reverse engineer the ability to decrypt traffic such that the reverse engineer can view packets in the same manner as the server. A tool such as mitmproxy could benefit from a plugin that takes an ephemeral key and decrypts data from a given IP address granting the mitmproxy the same view of packets on the network as the server. This would allow for the tools available in mitmproxy to be used in a man in the middle attack against systems that perform certificate pinning.

The work on algorithmic entropy in ISN creation in Chapter 6 just scratched the surface of the information that can be learned by sending well crafted packets to machines on the Internet. By changing the fields in the packets sent one could measure how systems respond when the client's port stays static. For example, the impact of the client's ISN on the server's ISN could be studied by adjusting the scan such that the client's ISN is static. In addition, the ISN study only covered port 80. The same analysis could be used to study other common ports such as 22 or 8080. The ISN study could also be extended by discovering what types of systems exhibit the different ISN trends when graphed. For example, one could study what systems

mostly tend to be up-trending, or bi-level.

7.1.2 New Applications

An interesting application, in yet another domain, of the techniques presented in this dissertation, is that of reverse engineering protocols. Previous work in this area has been done by Cui *et al.* [32] which relies on network traces and leaves application based reverse engineering for future work. Work by Caballero *et al.* [13] has been done on protocol reverse engineering, focusing on protocol format extraction from the binary. However, this work leaves the discovery of the process that models the protocol state machine to future work. Using automation and dynamic analysis, a system could be created which automatically tracks network data to functions that parse the aforementioned data. Once the program parses the data, the system would then track the data to functions that utilize it giving a reverse engineer important data such as: the number of bytes in each field, the data in each field, as well as a function tree displaying how the data is used. This data could then be utilized to model the process of the protocol state machine which, in the words of Caballero *et al.*, "captures the sequences of messages that represent valid sessions of the protocol." Meaning the proposed work would not only discover the structure of the protocol (*e.g.*, the first 6 bytes are the version number, the next 4 are a time stamp, *etc.*) but also the semantic meaning behind a series of messages from client to server.

One of the most interesting applications I can see for discovering ephemeral secrets in reverse engineering tasks would be to automatically discover the process performed by packers. A packer is a program that obfuscates a program's binary making reverse engineering the binary anywhere from complex to impossible. Packers are an enormous hurdle to malware reverse engineers who deal with obfuscated code. Early work has been done on packers and other obfuscation techniques, by Quist *et al.* [105]. Quist's work however has become dated as packers have contin-

ued to become more sophisticated. Often times sophisticated packers require reverse engineers to take snapshots of memory which are inspected in hopes of finding any clues that hint at where else to look. This problem appears to be a great opportunity for the application of dynamic analysis and automation to uncover the ephemeral secret that is the packing process. The goal would be to automatically unpack (or deobfuscate) the code such that the reverse engineer is free from performing the deobfuscation and can focus on their original task.

As I have demonstrated in this dissertation, by making the discovery of ephemeral secrets explicit in the process of reverse engineering we can begin to solve many reverse engineering problems presented by dynamic systems.

References

[1] LINE vulnerable to man-in-the-middle attack. https://www.telecomasia.net/blog/content/line-vulnerable-man-middle-attack.

[2] New "hidden chat" feature released, enables sending of time-limited messages : LINE official blog. http://official-blog.line.me/en/archives/1006361166.html, July 2014.

[3] Secret chat and decline invites now available on kakaotalk | kakao blog. https://blog.kakaocorp.com/?p=943, December 2014.

[4] Usernames and secret chats 2.0. https://telegram.org/blog/usernames-and-secret-chats-v2, October 2014.

[5] R. Abu-Salma, M. A. Sasse, J. Bonneau, A. Danilova, A. Naiakshina, and M. Smith. Obstacles to the adoption of secure communication tools. In *Security and Privacy (SP), 2017 IEEE Symposium on (SP'17). IEEE Computer Society*, 2017.

[6] How private are your favourite messaging apps? https://www.amnesty.org/en/latest/campaigns/2016/10/which-messaging-apps-best-protect-your-privacy/.

[7] A. Arora, R. Krishnan, R. Telang, and Y. Yang. An empirical analysis of software vendors' patching behavior: Impact of vulnerability disclosure. *ICIS 2006 Proceedings*, page 22, 2006.

[8] S. Bellovin. RFC1948: defending against sequence number attacks. *Status: INFORMATIONAL*, 1996.

[9] S. M. Bellovin and F. Gont. RFC6528: defending against sequence number attacks. Technical report, Technical report, Feb, 2012.

[10] D. J. Bernstein. SYN cookies, 1996. *URL http://cr. yp. to/syncookies. html*, 2016.

[11] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

[12] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *In the Proceedings of the Sixth Workshop on Very Large Corpora*, pages 152–160, 1998.

[13] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 621–634, New York, NY, USA, 2009. ACM.

[14] J. Calvet, J. M. Fernandez, and J.-Y. Marion. Aligot: Cryptographic function identification in obfuscated binary programs. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 169–182, New York, NY, USA, 2012. ACM.

[15] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel. Off-path tcp exploits: Global rate limit considered dangerous. In *USENIX Security Symposium*, pages 209–225, 2016.

[16] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel. Off-path TCP exploits of the challenge ACK global rate limit. *IEEE/ACM Transactions on Networking*, 26(2):765–778, 2018.

[17] H. Cavusoglu and S. Raghunathan. Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. *IEEE Transactions on Software Engineering*, 33(3), 2007.

[18] Censys. `https://censys.io/`.

[19] G. J. Chaitin. Information-theoretic limitations of formal systems. *J. ACM*, 21(3):403–424, July 1974.

[20] W. Chen and Z. Qian. Off-path TCP exploit: How wireless routers can jeopardize your secret. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018.

[21] H. L. Chieu and H. T. Ng. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[22] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding data lifetime via whole system simulation. In *USENIX Security Symposium*, pages 321–336, 2004.

[23] J. Clause, W. Li, and A. Orso. Dytan: A generic dynamic taint analysis framework. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ISSTA '07, pages 196–206, New York, NY, USA, 2007. ACM.

[24] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. *I/S: A Journal of Law and Policy for the Information Society*, 3(2):70–77, 2007.

[25] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the Signal messaging protocol. Cryptology ePrint Archive, Report 2016/1013, 2016. `http://eprint.iacr.org/2016/1013`.

[26] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, C. Shannon, and J. Brown. Can we contain Internet worms? In *HotNets III*, 2004.

[27] G. Couprie. Telegram, AKA "Stand back, we have Math PhDs!". `https://unhandledexpression.com/2013/12/17/telegram-stand-back-we-know-maths/`.

[28] J. Crandall and F. T. Chong. A security assessment of the Minos architecture. In *Workshop on Architectural Support for Security and Anti-virus*, 2004.

[29] J. R. Crandall, S. F. Wu, and F. T. Chong. Minos: Architectural support for protecting control data. *ACM Trans. Archit. Code Optim.*, 3(4):359–389, 2006.

[30] J. R. Crandall, D. Zinn, M. Byrd, E. Barr, and R. East. ConceptDoppler: a weather tracker for Internet censorship. In *Proc. of 14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[31] M. Crete-Nishihata, A. Hilts, J. Knockel, J. Q. Ng, L. Ruan, and G. Wiseman. Harmonized Histories? A year of fragmented censorship across Chinese live streaming applications. Technical report, Technical report, Citizen Lab, University of Toronto. Available at https://netalert.me/assets/harmonizedhistories/harmonized-histories. pdf, 2016.

[32] W. Cui, J. Kannan, and H. J. Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *USENIX Security Symposium*, pages 1–14, 2007.

[33] Y. K. Dalal. More on selecting sequence numbers. In *ACM SIGOPS Operating Systems Review*, volume 9, pages 25–36. ACM, 1975.

[34] M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A flexible information flow architecture for software security. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 482–493, New York, NY, USA, 2007. ACM.

[35] A. Darer, O. Farnan, and J. Wright. Filteredweb: A framework for the automated search-based discovery of blocked urls. In *Network Traffic Measurement and Analysis Conference (TMA), 2017*, pages 1–9. IEEE, 2017.

[36] A. Darer, O. Farnan, and J. Wright. Automated discovery of Internet censorship by web crawling. *CoRR*, abs/1804.03056, 2018.

[37] David Johnston. sp800_22_tests. Available at `https://github.com/dj-on-github/sp800_22_tests`, November 2017.

[38] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[39] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan. Repeatable reverse engineering for the greater good with PANDA. *Columbia University Computer Science Technical Reports. New York*, 2014.

[40] The Double Ratchet Algorithm . `https://whispersystems.org/docs/specifications/doubleratchet/`.

[41] A. Dunna, C. O'Brien, and P. Gill. Analyzing China's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, 2018.

[42] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast internet-wide scanning and its security applications. In *USENIX Security Symposium*, volume 8, pages 47–53, 2013.

[43] A. M. Espinoza and J. R. Crandall. Work-in-progress: Automated named entity extraction for tracking censorship of current events. 2011.

[44] A. M. Espinoza, J. Knockel, P. Comesaña-Alfaro, and J. R. Crandall. V-dift: Vector-based dynamic information flow tracking with application to locating cryptographic keys for reverse engineering. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 266–271. IEEE, 2016.

[45] A. M. Espinoza, W. J. Tolley, J. R. Crandall, A. Hilts, and M. Crete-Nishihata. Alice and Bob, who the FOCI are they?: Analysis of end-to-end encryption in the LINE messaging application. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*. USENIX Association, 2017.

[46] D. Fifield and L. Tsai. Censors' delay in blocking circumvention proxies. 2016.

[47] Spy Tech 'Hacks WhatsApp Encrypted Chat From A Backpack'. Forbes, `https://www.forbes.com/sites/thomasbrewster/2016/09/29/wintego-whatsapp-encryption-surveillance-exploits/#5b6aaccb1aa9`.

[48] A. Forget, S. Pearman, J. Thomas, A. Acquisti, N. Christin, L. F. Cranor, S. Egelman, M. Harbach, and R. Telang. Do or do not, there is no try: user engagement may not improve security outcomes. In *Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[49] Iranian Hackers Just Cracked This Super Secure Instant Messaging Service. Fortune, `http://fortune.com/2016/08/02/telegram-hackers-iran/`.

[50] Manipulating Social Media to Undermine Democracy. `https://freedomhouse.org/report/freedom-net/freedom-net-2017`.

[51] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz. How secure is TextSecure? In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 457–472, March 2016.

[52] Multiple vulnerabilities in LINE instant messenger platform. Full Disclosure, `http://seclists.org/fulldisclosure/2016/Aug/45`.

[53] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks on Apple iMessage. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 655–672, Austin, TX, 2016. USENIX Association.

[54] The Great Firewall Revealed. Whitepaper released by the Global Internet Freedom Consortium in December of 2002.

[55] Y. Gilad and A. Herzberg. Off-path attacking the web. In *WOOT*, pages 41–52, 2012.

[56] Y. Gilad and A. Herzberg. When tolerance causes weakness: the case of injection-friendly browsers. In *Proceedings of the 22nd international conference on World Wide Web*, pages 435–446. ACM, 2013.

[57] Y. Gilad and A. Herzberg. Off-path TCP injection attacks. *ACM Transactions on Information and System Security (TISSEC)*, 16(4):13, 2014.

[58] Y. Gilad, A. Herzberg, and H. Shulman. Off-path hacking: The illusion of challenge-response authentication. *IEEE Security & Privacy*, 12(5):68–77, 2014.

[59] F. Gröbert, C. Willems, and T. Holz. Automated identification of cryptographic primitives in binary programs. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 41–60, Berlin, Heidelberg, 2011. Springer-Verlag.

[60] S. Hardy. Asia chats: Investigating regionally-based keyword censorship in LINE. Technical Report Research Brief No. 25, Citizen Lab, November 2013.

[61] B. Harris and R. Hunt. TCP/IP security threats and attack methods. *Computer communications*, 22(10):885–897, 1999.

[62] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, volume 8, page 1, 2012.

[63] D. D. Hosfelt. Automated detection and classification of cryptographic algorithms in binary programs through machine learning. *CoRR*, abs/1503.01186, 2015.

[64] A. Hounsel, P. Mittal, and N. Feamster. Automatically generating a large, culture-specific blocklist for China. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*. USENIX Association, 2018.

[65] "Race to the Bottom": Corporate Complicity in Chinese Internet Censorship, August 2006. `http://www.hrw.org/reports/2006/china0806`.

[66] I. Ion, R. Reeder, and S. Consolvo. "... no one can hack my mind": Comparing expert and non-expert security practices. In *Proc. SOUPS*, 2015.

[67] A. Jain, H. Gonzalez, and N. Stakhanova. Enriching reverse engineering through visual exploration of android binaries. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, page 9. ACM, 2015.

[68] J. Jakobsen and C. Orlandi. On the CCA (in)security of MTProto. Cryptology ePrint Archive, Report 2015/1177, 2015. `http://eprint.iacr.org/2015/1177`.

[69] J. Jermyn and N. Weaver. Autosonda: Discovering rules and triggers of censorship devices. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI 17)*, Vancouver, BC, 2017. USENIX Association.

[70] A. Jung-Loddenkemper. A 2^{64} attack on telegram, and why a super villain doesn't need it to read your telegram chats. `https://www.alexrad.me/discourse/a-264-attack-on-telegram-and-why-a-super-villain-doesnt-need-it-to-read-your-telegram-chats.html`.

[71] M. G. Kang, S. McCamant, P. Poosankam, and D. Song. DTA++: Dynamic taint analysis with targeted control-flow propagation. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2011.

[72] D. King, B. Hicks, M. Hicks, and T. Jaeger. Implicit flows: Can't live with 'em, can't live without 'em. In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, pages 56–70, Berlin, Heidelberg, 2008. Springer-Verlag.

[73] J. Knockel and J. R. Crandall. Counting packets sent between arbitrary internet hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, 2014.

[74] J. Knockel, J. R. Crandall, and J. Saia. Three researchers, five conjectures: An empirical analysis of TOM-Skype censorship and surveillance. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI 11)*, 2011.

[75] C. Lab. Asia chats: LINE corporation responds. `https://citizenlab.org/2013/12/asia-chats-line-responds/`, 2013.

[76] T. K. Landauer, P. W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[77] J. Lemon et al. Resisting SYN flood DoS attacks with a SYN cache. In *BSDCon*, volume 2002, pages 89–97, 2002.

[78] P. Lestringant, F. Guihéry, and P.-A. Fouque. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 203–214, New York, NY, USA, 2015. ACM.

[79] LINE. LINE security bug bounty program, 2017. https://bugbounty.linecorp.com/en/.

[80] Hidden chat users to enjoy "Letter Sealing" from July! : LINE official blog. http://official-blog.line.me/en/archives/1058913293.html, June 2016.

[81] Letter sealing gets enhanced! http://official-blog.line.me/en/archives/1060089042.html, August 2016.

[82] LINE encryption overview: Technical whitepaper. https://scdn.line-apps.com/stf/linecorp/en/csr/line-encryption-whitepaper-ver1.0.pdf, September 2016.

[83] New generation of safe messaging: "Letter Sealing". LINE Blog, https://engineering.linecorp.com/en/blog/detail/65.

[84] True Delete. LINE Engineering Blog, https://engineering.linecorp.com/en/blog/detail/64.

[85] LINE transparency report (July-December 2016). https://linecorp.com/en/security/tr_report_2016_2, April 2017.

[86] LINE engineers' blog: New generation of safe messaging: "letter sealing". https://web.archive.org/web/20151221181555/http://developers.linecorp.com/blog/?p=3679. Accessed: 2018-02-05.

[87] N. Lomas. Security researchers call for Guardian to retract false WhatsApp "backdoor" story. Tech Crunch. https://techcrunch.com/2017/01/20/security-researchers-call-for-guardian-to-retract-false-whatsapp-backdoor-story/, January 2017.

[88] N. Lutz. Towards revealing attackers' intent by automatically decrypting network traffic. Master's thesis, ETH Zurich, 2008.

[89] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson. An analysis of China's "great cannon". *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*, page 37, 2015.

[90] S. Millward. Line is getting dangerously dependent on users in its 4 top countries. https://www.techinasia.com/line-q1-2016-dependent-four-countries, April 2016. (Accessed on 05/26/2017).

[91] J. Ming, D. Wu, G. Xiao, J. Wang, and P. Liu. Taintpipe: Pipelined symbolic taint analysis. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 65–80, Washington, D.C., Aug. 2015. USENIX Association.

[92] Man in the middle proxy. `https://mitmproxy.org/`. Accessed: 2018-01-30.

[93] Mr. Tao. China: Journey to the heart of Internet censorship. Investigative report sponsored by Reporters Without Borders For Freedom and Chinese Human Rights Defenders, Oct 2007.

[94] A. Munir, Z. Qian, Z. Shafiq, A. Liu, and F. Le. Multipath TCP traffic diversion attacks and countermeasures. In *Network Protocols (ICNP), 2017 IEEE 25th International Conference on*, pages 1–10. IEEE, 2017.

[95] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2005.

[96] Open observatory of network interference. `https://ooni.torproject.org/`. Accessed: 2018-09-18.

[97] Open NLP. `http://opennlp.apache.org/`. Accessed: 2018-11-10.

[98] J. C. Park and J. R. Crandall. Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of HTML responses in China. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 315–326, Washington, DC, USA, 2010. IEEE Computer Society.

[99] V. Paxson. End-to-end internet packet dynamics, 1997.

[100] J. Postel et al. Transmission control protocol RFC 793, 1981.

[101] Z. Qian and Z. M. Mao. Off-path TCP sequence number inference attack-how firewall middleboxes reduce security. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 347–361. IEEE, 2012.

[102] Z. Qian, Z. M. Mao, and Y. Xie. Collaborative TCP sequence number inference attack: how to crack sequence number under a second. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 593–604. ACM, 2012.

[103] F. Qin, C. Wang, Z. Li, H. seop Kim, Y. Zhou, and Y. Wu. LIFT: A low-overhead practical information flow tracking system for detecting security attacks. *MICRO-39*, pages 135–148, December 2006.

[104] A. Quach, Z. Wang, and Z. Qian. Investigation of the 2016 linux TCP stack vulnerability at scale. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):4, 2017.

[105] D. Quist and V. Smith. Covert debugging circumventing software armoring techniques. *Black hat briefings USA*, 2007.

[106] E. Rader and R. Wash. Identifying patterns in informal sources of security information. *Journal of Cybersecurity*, 1(1):121, 2015.

[107] A. Ramaiah, R. Stewart, and M. Dalal. Improving TCP's robustness to blind in-window attacks, 2010.

[108] E. M. Redmiles, S. Kross, and M. L. Mazurek. How I learned to be secure: A census-representative survey of security advice sources and behavior. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 666–677, New York, NY, USA, 2016. ACM.

[109] M. G. Rekoff. On reverse engineering. *IEEE Transactions on systems, man, and cybernetics*, pages 244–252, 1985.

[110] Codegate 2014. `https://www.alexrad.me/discourse/codegate-2014.html`.

[111] LINE instant messenger protocol documentation. `http://altrepo.eu/git/line-protocol`.

[112] LINE protocol analysis. `https://hexa-unist.github.io/2013/09/05/LINE-protocol-analysis/`.

[113] python wrapper for LOCOP protocol. `http://carpedm20.blogspot.kr/2013/08/python-wrapper-for-loco-protocol.html`.

[114] Adopting SPDY in LINE. part 1: An overview. `http://tech.naver.jp/blog/?p=2381`, 2013.

[115] P. Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015. `http://eprint.iacr.org/2015/1162`.

[116] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Booz-Allen and Hamilton Inc Mclean Va, 2001.

[117] A. Salem and S. Banescu. Metadata recovery from obfuscated programs using machine learning. In *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering*, page 1. ACM, 2016.

[118] R. Sartea, M. D. Preda, A. Farinelli, R. Giacobazzi, and I. Mastroeni. Active android malware analysis: an approach based on stochastic games. In *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering*, page 5. ACM, 2016.

[119] A. Sfakianakis, E. Athanasopoulos, and S. Ioannidis. Censmon: A web censorship monitor. *Free and Open Communications on the Internet. USENIX*, page 113, 2011.

[120] A. Shu. *Data Mining of Chinese Social Media*. PhD thesis, Rice University, 2014.

[121] A. Slowinska and H. Bos. Pointless tainting? Evaluating the practicality of pointer tainting. In *Proceedings of the 4th EuroSys Conference*, Nuremberg, Germany, Apr 2009.

[122] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. *SIGARCH Comput. Archit. News*, 32(5):85–96, Oct. 2004.

[123] R. S. Tomlinson. Selecting sequence numbers. *ACM SIGOPS Operating Systems Review*, 9(3):11–23, 1975.

[124] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: Secure messaging. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 232–249, Washington, DC, USA, 2015. IEEE Computer Society.

[125] G. Venkataramani, I. Doudalis, Y. Solihin, and M. Prvulovic. Flexitaint: A programmable accelerator for dynamic taint propagation. In *HPCA*, pages 173–184. IEEE Computer Society, 2008.

[126] N. Villeneuve. Breaching trust: An analysis of surveillance and security practices on China's TOM-Skype platform. Available at `http://www.infowar-monitor.net/breachingtrust/`.

[127] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. Reformat: Automatic reverse engineering of encrypted messages. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 200–215, Berlin, Heidelberg, 2009. Springer-Verlag.

[128] N. Weaver, R. Sommer, and V. Paxson. Detecting forged TCP reset packets. In *NDSS*, 2009.

[129] R. Weber. How service providers of messengers with end-to-end encryption could be compelled to decrypt messages. `https://plus.google.com/+RolfWeber/posts/SYw6AD8xkK7`.

[130] R. Whelan, T. Leek, and D. Kaeli. Architecture-independent dynamic information flow tracking. In *Proceedings of the 22nd International Conference on Compiler Construction*, CC'13, pages 144–163, Berlin, Heidelberg, 2013. Springer-Verlag.

[131] Wire Privacy. `https://wire.com/en/privacy/#table-competition`.

[132] H. Yin, D. Song, M. Egele, and E. Kruegel, Christopher a nd Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127, New York, NY, USA, 2007. ACM.

[133] M. Zalewski. Strange attractors and TCP/IP sequence number analysis, 2001.

[134] J. Zdziarski. WhatsApp Forensic Artifacts: Chats Aren't Being Deleted. `https://www.zdziarski.com/blog/?p=6143`.

[135] R. Zhao, D. Gu, J. Li, and H. Liu. Detecting encryption functions via process emulation and IL-based program analysis. In *Proceedings of the 14th International Conference on Information and Communications Security*, ICICS'12, pages 252–263, Berlin, Heidelberg, 2012. Springer-Verlag.

[136] T. Zhu, D. Phipps, A. Pridgen, J. R. Crandall, and D. S. Wallach. The velocity of censorship: High-fidelity detection of microblog post deletions. In *USENIX Security Symposium*, Washington, DC, USA, 2013. USENIX Association. `https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_zhu.pdf`.

[137] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.