NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

APPLIED CYBER OPERATIONS CAPSTONE REPORT

INSTANT MESSAGE TRAFFIC META-DATA AND ITS SUSCEPTIBILITY TO TRAFFIC ANALYSIS

by

Jeana M. Verkempinck, Alexander Arnell, and Cassondra C. Bullock

June 2020

Advisor: Vinnie Monaco
Second Reader: Robert Beverly

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2020	3. REPORT TYPE AND DATES COVERED Applied Cyber Operations Capstone Report

4. TITLE AND SUBTITLE INSTANT MESSAGE TRAFFIC META-DATA AND ITS SUSCEPTIBILITY TO TRAFFIC ANALYSIS	5. FUNDING NUMBERS
6. AUTHOR(S) Jeana M. Verkempinck, Alexander Arnell, and Cassondra C. Bullock	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

Instant Message (IM) applications are commonly used by both civilian and DOD personnel for both communication and collaboration. The web-based variants of these applications generally ride encrypted channels for message security. However, these channels may be vulnerable to keystroke timing attacks whereby textual content is determined by the timing of network traffic induced by keyboard events. An example of this induced traffic is the activity notifications common to many of these platforms, indicating when a conversant begins typing. Our aim is to determine whether the network traffic that carries this metadata enables recovering portions of the message or leaks information about the sender's identity. Using a combination of network packet capture analysis and local keystroke logging, we characterize traffic patterns of three widely used web-based IM platforms: Facebook Messaging, Google Hangouts, and Internet Relay Chat (IRC) through the Kiwi IRC web client.

14. SUBJECT TERMS meta-data, chat, Instant Message, vulnerabilities, Facebook, Microsoft Teams, network traffic analysis, covert channels, data exfiltration	15. NUMBER OF PAGES 81
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

INSTANT MESSAGE TRAFFIC META-DATA AND ITS SUSCEPTIBILITY TO TRAFFIC ANALYSIS

PO1 Jeana M. Verkempinck (USN), PO1 Alexander Arnell (USN), and CPO Cassondra C. Bullock (USN)

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED CYBER OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Reviewed by:

Vinnie Monaco	Robert Beverly
Advisor	Second Reader

Accepted by:

Thomas J. Housel
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Instant Message (IM) applications are commonly used by both civilian and DOD personnel for both communication and collaboration. The web-based variants of these applications generally ride encrypted channels for message security. However, these channels may be vulnerable to keystroke timing attacks whereby textual content is determined by the timing of network traffic induced by keyboard events. An example of this induced traffic is the activity notifications common to many of these platforms, indicating when a conversant begins typing. Our aim is to determine whether the network traffic that carries this metadata enables recovering portions of the message or leaks information about the sender's identity. Using a combination of network packet capture analysis and local keystroke logging, we characterize traffic patterns of three widely used web-based IM platforms: Facebook Messaging, Google Hangouts, and Internet Relay Chat (IRC) through the Kiwi IRC web client.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Acronyms and Abbreviations

ACK	Transmission Control Protocol (TCP) Acknowledgement
AJAX	Asynchronous JavaScript and XML
C2	Command and Control
CLI	Command Line Interface
CPU	Central Processing Unit
CSV	Comma Separated Value
DoD	Department of Defense
DOM	Document Object Model
ETL	Microsoft (MS) Event Trace Log File
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IM	Instant Messenger
IP	Internet Protocol
IRC	Internet Relay Chat
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LAN	Local Area Network
MS	Microsoft

NAT	Network Address Translation
netsh	Network Shell - a Windows Command Line Interface (CLI) tool
NPS	Naval Postgraduate School
NTP	Network Time Protocol
OS	Operating System
P2P	Peer-to-Peer
PCAP	Packet Capture
PS	Microsoft PowerShell
PSD	Power Spectral Density
RAM	Random Access Memory
Rx	Receive
SASL	Simple Authentication and Security Layer
SIP	Session Initiation Protocol
SMS	Short Message Service
SSL	Secure Socket Layer
TAP	Throwing Star LAN Tap
TCP	Transmission Control Protocol
TLS	Transport Layer Security
Tx	Transmit
UEFI	Unified Extensible Firmware Interface
USB	Universal Serial Bus
USN	U.S. Navy
XMPP	Extensible Messaging and Presence Protocol

Acknowledgments

The authors would like to thank the following people for the assistance and support provided throughout this capstone process.

- Vinnie Monaco, from Naval Postgraduate School, for his guidance as an advisor and support with the Java key logger application Biologger.
- Robert Beverly, from NPS, as a second reader and insistence for excellence.
- Daryl Williams and ITAC at large from NPS, for expending the time and effort required to procure and access an appropriate network environment in which to conducting our experimentation.
- The Facebook for Developers forum, for providing insight into the Facebook API.
- Zotero for creating a tool that was up to the task of handling the many citations and resources we needed to properly reference this project.
- Overleaf.com, which created an indispensable platform for collaboration and generation of this work.
- Usually last, but most importantly, our spouses and children, for their perpetual love, tolerance, and support shown throughout the many hours we spent physically or mentally absent from home, our time instead spent working on this project.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Keystroke analysis is far from a new topic. Looking back to the second world war, military intelligence was able to distinguish an ally from an enemy based on the rhythm of their Morse Code messages [1]. In more current times, biometric techniques based on keystroke timing has been used as a form of user authentication [1], and to discern what is actually being typed by users based on keystroke frequency [2]. Recent research has even had success with conducting remote keystroke analysis through monitoring network traffic activity by timing the auto-complete response times generated by certain search engines [3].

This creates an interesting question, if packet trace data is capable of providing enough information to successfully discern typed input, what other forms of network traffic could be vulnerable to remote keystroke timing attacks. Specifically, if traffic generated on a key-to-packet basis is susceptible to a keystroke timing attack, would web based Instant Messenger (IM) traffic also be susceptible to this form of attack?

The requirements are simple: an application that generates network traffic as a direct result of each user keystroke. These kind of applications are becoming commonplace due to cloud computing and collaborative tools. An example of this kind of application can be seen when using Microsoft Teams to collaborate on a PowerPoint presentation. Once the PowerPoint is shared, edits can be seen in real time at each device viewing Teams chat. Even the location of the cursor within the presentation is visible to the rest of those in the chat.

It should seem obvious to the casual observer that in order for these edits to become visible in real time, they must be transmitted between the devices over a network. Generally, these packets must traverse a network to a server, then back to the receiving devices. This traffic could then be monitored anywhere along the path between the two users. This indicates the possibility that the packets in that traffic may contain the necessary observable timing data to conduct remote keystroke analysis.

With real time collaboration tools, the signs of keystroke based traffic is obvious. What is less obvious is indicators of activity, such as those used by many popular IM applications

like Facebook. Facebook has a typing indicator that notifies it's clients if somebody is typing. These notifications are associated to a client's typing actions, and may also provide real time traffic that could be used for keystroke analysis.

The uses of typing notification is far from exclusive to Facebook, and many IM applications make use of similar features, including those used by the Department of Defense (DoD) and the U.S. Navy (USN) in particular.

1.1 Motivation

The use of IM applications for real-time communications has become an essential part of daily DoD communications.

Official IM tools such as Internet Relay Chat (IRC) [4] clients like mIRC and more recently MakoChat are used in shipboard tactical environments. Corporate products like Microsoft (MS)'s Skype for Business can also be found within a number of government enclaves.

Un-official messaging applications such as Short Message Service (SMS) and Facebook Messenger can be found in use throughout the military by sailors and soldiers. These IM applications are often used for general communication to individuals. This becomes a problem when used for providing technical details for system troubleshooting, or for scheduling watch rotations. Generally none of this information could be considered classified, however in aggregate this could become a serious risk.

Both official and un-official IM channels carry a large amount of data. An adversary capable of conducting side-channel attacks against the security these applications are built on, has the potential to become a serious threat.

Threats are based on either directly captured and interrupted conversations or tracking relative movements of the sender. Much of the message traffic sent over IM could be aggregated to provide information to an adversary. For example, a Sailor simply asking about their duty schedule could provide an adversary with the watch rotation schedule. An off-site technician queried to troubleshoot minor problems over time could provide an adversary a look into what kind of personnel expertise and capability that command has. These are standard security concerns that apply to all forms of communication.

Even if a conversation cannot be directly interrupted, behavior analysis of traffic might be used to track a user. This would be similar to distinguishing ally to enemy Morse Code messages based on rhythm mentioned earlier or how a device might be uniquely fingerprinted using it's clock [5]. An enemy may not be able to translate the actual message, but use the fingerprint to digitally stalk the user. A particular concern for those who are otherwise trying to remain anonymous.

1.2 Problem Statement

Reading the network traffic between two users through monitoring tools is generally considered a straight forward process when the traffic has not been encrypted. Fortunately, most web based traffic is encrypted using Transport Layer Security (TLS), which makes deciphering traffic between two users difficult at best. So, side channel attacks that exploit network behavior have become more prominent. In particular, recent work on the use of timing analysis for keystroke recognition has resulted in the capability to remotely correlate packet intervals with keystroke recognition [3]. This work showed a text recognition accuracy up to 15.83% assuming the user types common English words.

We are expanding on this work by reviewing a number of commonly used web-based IM applications[1]. Specifically we are looking at whether the "typing" notification or similar traffic induced by keystroke events could lend itself to a keystroke timing attack.

1. *Is encryption used?* An assumption has been made that most IM applications use TLS or similar encryption schemes to secure their message traffic. This must be verified, if some form of encryption is not used, then there is no need to attempt methods that would circumvent encryption.

2. *Does the IM web application provide keystroke interaction based traffic?* If the notifications use a polling mechanism [3] to buffer time interval, this can obscure or eliminate both the number and time of actual keystrokes. Depending on the polling periodicity, this could effectively negate statistical analysis of temporal traffic for keystroke analysis.

3. *Is the meta-data in-band or out-of-band?* Some web based applications may create

[1] "A web-based application is any program that is accessed over a network connection using HTTP, rather than existing within a device's memory" [6].

multiple Transmission Control Protocol (TCP) sockets with the server, or a Peer-to-Peer (P2P) connection directly to the distant client. If multiple connections exist, do they maintain similar levels of encryption and can all the connections be monitored. These factors have the potential to affect an adversaries ability to intercept and interpret the connections.

4. *Is there any other element that lends this application to or against this form of attack?* Some factors unique to a specific application may exist, that might have an influence on how effective a keystroke timing attack might be applied. These factors should be noted and examined for impact against the overall goal.

These questions will be approached through the use of a practical experiment, collecting first-hand data on both keystroke timing and the resultant network traffic. This data will then be synchronized and statistically analysed to determine the practicality of keystroke timing attacks on the basis of distinguishing the possibility of individual keystrokes and use of future behavior timing based analysis.

1.3 Summary of Contributions

- To the best of our knowledge, this is the first work completed on keystroke timing analysis of web-based IM applications.
- Our methodology can be applied to future research with little to no modification of the environmental design.
- We found the use of TLS applied to all of the IM web-based applications we tested.
- We were able to differentiate between encrypted traffic containing chat data and Command and Control (C2) traffic containing notifications.
- We did not find any direct correlation between the timing of individual keystrokes and packet transmission, with the exception of the first key pressed and message transmission.
- All web-based IM applications tested made use of some form of timed padding to indicate client activity.
- All the meta-data was transferred over a single channel unique to the chat session.
- The web-based IM applications tested would not be susceptible to individual keystroke analysis, but longer term behavioral analysis of individualized typing style might be applied for some of the applications tested

1.4 Thesis Organization

The organization of the thesis is as follows:

- Chapter 2 covers background information and a review of works related to the topics covered.
- Chapter 3 discusses the methodologies for data collection and design of the environment used.
- Chapter 4 covers the implemented data capture and process used to analyse the data.
- Chapter 5 provides analysis of the data capture results.
- Chapter 6 concluded our work and provides some potential recommendations towards future research in the area of keystroke-to-packet capture analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background and Related Work

IM has become an increasingly popular form of communication for business and personal use with features that allow participating parties to communicate over a network in real-time [7]. IM applications can be used to share documents, photos, videos, and audio files. IM can be utilized on various devices such as mobile devices or desktop computers; and through various applications like Google Hangouts, Facebook, and Apple's iMessage. IM services also provide the ability to communicate within two-way and multi-party, or group, conversations. Prior to the start of our traffic analysis, we provide information on how instant messaging works. Our research will cover Facebook Messenger, Google Hangouts, and Kiwi IRC's web client for IRC as accessed via a web browser.

2.1 The Process of Instant Messaging

There are three main features to IM: "text-based, bi-directionally exchanged, and real-time" [8]. IM works within a client-server based architecture using protocols such as the Extensible Messaging and Presence Protocol (XMPP), the IRC protocol, and the Session Initiation Protocol (SIP). Generally, within the IM architecture, the server will store the client data and route messages to the proper recipients [9]. When users communicate with each other via IM, there are TCP connections that are established with the IM server for each user [10]. Within a peer-to-peer IM architecture, the connections are made directly between the communicating peers where the only communications with the server are requests to set up the communications between the clients [11]. It should be noted, however, that problems can arise when peers from one network try to begin communicating with peers connected to the Internet through the use of Network Address Translation (NAT) causing the need to implement what is known as a NAT traversal mechanism [12]. Fig. 2.1 shows the layout of the client-server and P2P architectures.

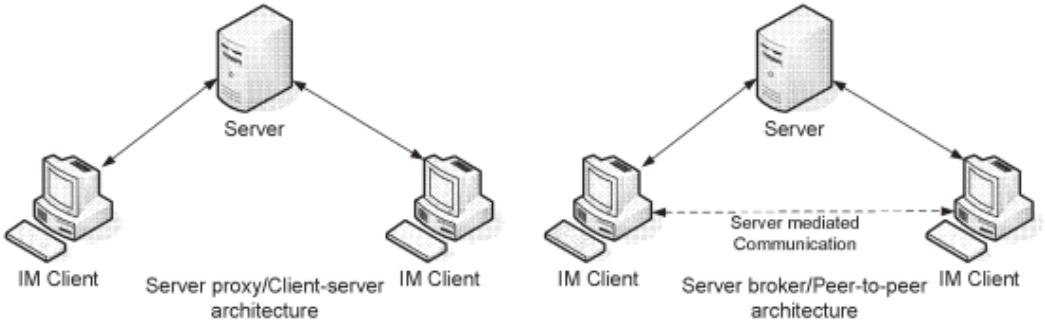


Figure 2.1: Client-Server architecture vs. P2P architecture. Source: [11].

The flow of IM conversations is maintained by the indicators of presence provided to the users within a conversation [13]. When a user begins to type a message, an indicator is then sent to the remote user, which is then displayed as a notification that the user is typing [13]. This notification is meant to signal to the user receiving this notification that they should wait until they receive the message before sending their own. Other indicators of presence can include a simple mouse movement that may change the user's online status from away to available and an application specific symbol, such as a green check, that lets others know that their message has been read – and in some cases, the time at which it was read. Users are also capable of communicating within multiple IM sessions at one time. This means that there can be any combination of two-way and/or multi-party active IM sessions (that a user can send and receive messages in) simultaneously.

2.1.1 Instant Messaging Security

IM communications can be recorded and monitored [14]. These chat communications have been susceptible to eavesdropping and attacks on archived conversation logs. Man-in-the middle attacks are also possible when encryption is not utilized. However, even with encryption implemented into the IM communications design, if the file system is not setup properly, an attacker can easily gain access to data files containing users' passwords [11]. IM traffic can be viewed and analyzed by using packet sniffers, such as Wireshark, to capture that traffic. When utilizing the web-based applications, as is without the use of Secure Socket Layer (SSL)\TLS, the IM communications will travel across the network in plain text. This can be of some concern for users who value their privacy. IM can

also reveal additional information about users and their presence such as whether they are currently logged in, actively typing within an established chat session, and if they have read the messages they have received. Users may, however, have the option to change their status to appear as if they are offline to avoid providing others with the true status of their online presence. One way in which web-based IM communications can be secured is via the use of the TLS protocol. The most common use of TLS is within the Hypertext Transfer Protocol (HTTP) protocol, known as Hypertext Transfer Protocol Secure (HTTPS), and it provides confidentiality, data integrity, authentication and other security features through digital certificates on top of TCP [15]. There is a trust level provided by HTTPS that "prevents eavesdropping of the communication" [15].

2.1.2 XMPP

When using XMPP, the XMPP server manages the traffic in the form of XML streams between the clients and servers [9]. For every direction of the bidirectional conversation during an XMPP session, there is an XML streaming document [15]. XMPP's method for securing and authenticating the traffic is channel encryption through the use of the TLS and Simple Authentication and Security Layer (SASL) protocols [9]. XMPP also provides the ability to collect presence information including a user's communication capability such as whether they have the capability to use additional features like audio, video conferencing, etc. [16]. Fig. 2.2 depicts an example of a XMPP architecture.

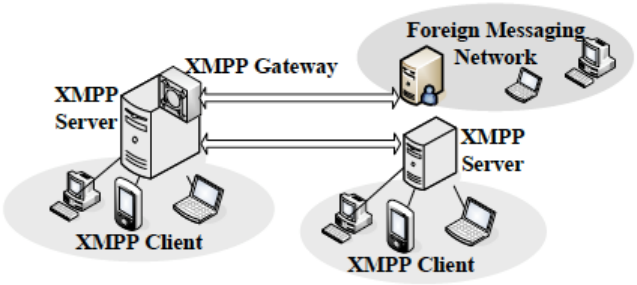


Figure 2.2: XMPP architecture. Source: [9].

Additionally, an XMPP server can be configured to accept WebSocket connections which enable other protocols to be layered on top of WebSocket to "extend applications over the Web" [17]. The WebSocket protocol is defined by RFC 6455 and offers the ability to utilize

a single TCP socket through which clients and servers can communicate bidirectionally over full-duplex communication channels [8]. The use of WebSockets also provides a continuous connection between the client and the server. When WebSockets are in use, the protocol switches from HTTP to the WebSocket protocol which then allows data to be exchanged between the client and server simultaneously [8].

2.2 Instant Messaging Service Applications

Today, there are many web-based IM applications available for use. Some of the currently used IM applications include Facebook Messenger, Google Hangouts, and various services that use the Internet Relay Chat protocol, such as Kiwi IRC. The following sections provide a summary of the IM chat applications that were used in our analysis.

2.2.1 Facebook Messenger

Facebook Messenger's web application located at Messenger.com, is a dedicated chat interface that is utilized through a browser without the use of Facebook.com or the mobile application [18]. Their website mentions that their Messenger service will provide end-to-end encryption by default as they are committed to privacy [18]. According to research and analysis conducted in 2011, when Facebook is accessed via a web-browser, there are caches of small Hypertext Markup Language (HTML) files with specific naming conventions that contain the chat messages [19], [20]. The research also discovered that these files could be located within "Random Access Memory (RAM), browser cache, pagefiles, unallocated clusters and system restore point" and that the message header was stored within the text-based human readable script called JavaScript Object Notation (JSON). Another research study also determined that when using Facebook chat communications, there are differences in the number of artifacts that can be found within browser caches or hard disks depending on the browser being used [21].

2.2.2 Google Hangouts

Google Hangouts web application is accessible via the Chrome extension or by the URL hangouts.google.com. It was launched in 2013 and includes features such as chat rooms, cloud stored conversations, and video conferencing [22]. Google Hangouts replaced their previous application called Google Talk. As a result of the replacement, Google Hangouts

stopped fully supporting its use of XMPP and moved to utilizing a more proprietary protocol [23]. The decreased support for XMPP also reduced user's ability to chat with others across different instant messaging services [23]. Another major change that occurred with the implementation of Google Hangouts, is that there was once the option of disabling chat history in Google Talk. That option was removed, and user's must now turn off conversation history for each contact individually if they wish to avoid saving a copy of their conversation history [23]. Multiple sources have reported that Google will eventually migrate Google Hangouts over to a newer application called Hangouts Chat and Meet [24].

2.2.3 Kiwi IRC

The Kiwi IRC Web IRC client can be accessed at kiwiirc.com and "works with any IRC network out of the box" [25]. In order to use Kiwi IRC, users must first install and configure the client program. The Kiwi IRC webpage provides in-depth guides on how to configure the application to suit the user's needs. The client configuration includes a JSON object that is loaded before Kiwi IRC starts [25]. According to their site, the chat application provides features that include themes, plugins, compatibility with all major browsers, SSL security, and support for multiple languages. Users also have the option to embed the IRC client widget into websites to enable instant and free live chat.

2.3 Security Threats to Instant Messaging

As previously stated, IM communications are thought to be protected from eavesdropping via the implementation of HTTPS. However, side-channel leaks and keystroke timing attacks threaten the confidentiality of users and therefore also threaten the security of instant messaging, even when those communications are encrypted.

2.3.1 Side-Channel Attacks

A side channel provides the ability to observe protected information that an attacker can use to exploit the implementation of a system versus the design of that system [26]. According to [26], side channels can be in the form of "electrical signals, acoustic signals, microarchitectural and architectural effects, application level timing channels, and any other shared resource through which an eavesdropper can detect any information or state left by another program". Information leaks can "occur as a result of normal program execution"

and attackers have the capability to evaluate the leaked information as "side-channel" information that can be used to extract secrets [26]. Furthermore, unique signatures are created on systems based upon a user's inputs, such as URL requests or encryption key information, that are entered into a program [26].

Within web applications, system state transitions are normally triggered by input data based upon smaller input spaces [27]. Techniques, such as Asynchronous JavaScript and XML (AJAX), are becoming utilized more frequently within very interactive and dynamic web interfaces which in turn results in low entropy inputs [27]. When similar techniques are used within the Graphical User Interface (GUI) widgets of web applications (e.g., auto-suggestion or auto-complete), the application is extremely responsive to the user's input [27]. As a result, even the smallest interactions, like selecting a radio button, create the possibility of generating web traffic that updates Document Object Model (DOM) objects in the application's browser interface [27]. These factors, in addition to web-applications being designed to guide users through their server-side component interactions, allow an attacker to map out the possible input values in order to match the web flow vector that they were able to detect [27]. Unfortunately, side-channel leaks can be difficult to prevent. Currently, mitigation techniques would not be easily standardized because side-channel leaks tend to be specific to the application being utilized. Due to the differences in application characteristics, the burden is on each application developer to identify ways to minimize and/or eliminate side-channel leaks altogether [27].

Keystroke timing attacks exploit a user's keyboard input. Attackers can use side-channels in order to acquire keystroke timings. This occurs when "microarchitectural attacks allow monitoring memory accesses with a granularity of single cache lines" which in turn enable the recovery of "keystroke timings with a high accuracy" [28]. Keystroke input data travels a great distance from the hardware component, through the operating system and to the application for which the data is being input. If an attacker focuses on any spot within the keystroke data path, they can detect a keystroke [28]. Prior research suggests that keystroke timing attacks against IM can be performed by employing the use of "a class of inline interception mechanisms", called JitterBugs, that transmit data covertly when situated at an input device nested inside of a trusted environment [29]. Additionally, the research states that since keystroke loggers do not exfiltrate data, the use of JitterBugs in conjunction with keystroke loggers solves that issue by leaking data and enabling a receiver that is monitoring

the traffic to recover that leaked data, even if the traffic is encrypted and the receiver is several hops away.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

The following chapter will discuss the design and implementation of the data capture environment in addition to the rationale behind various design choices. The goal of this environment centers specifically on creating a simulation in which two IM clients are communicating with a web based IM server while being monitored by a third party. Priority was placed on capturing client traffic flows from both client perspectives and the monitor, while minimizing non-native software installation on the clients.

3.1 Physical Environment Design

To minimize the possibility of introducing timing artifacts and reduce overhead filtering of Packet Capture (PCAP), no virtualization technology was implemented in this network. The Local Area Network (LAN) shown in Fig. 3.1 provides a visualization of the infrastructure. Both clients (Alex[2] and Bullock[3]) are physical computers running Windows 10 and the packet sniffer or monitor (Marie) was running Kali Linux with two ethernet cards. A layer 2 switch was selected to insure minimal any plausible latency cause by packet processing on the client side.

[2]Lenovo Flexx 2-15D with a standard Dell Business Keyboard.
[3]Toshiba Satellite A215 with a Das Keyboarad Model S.

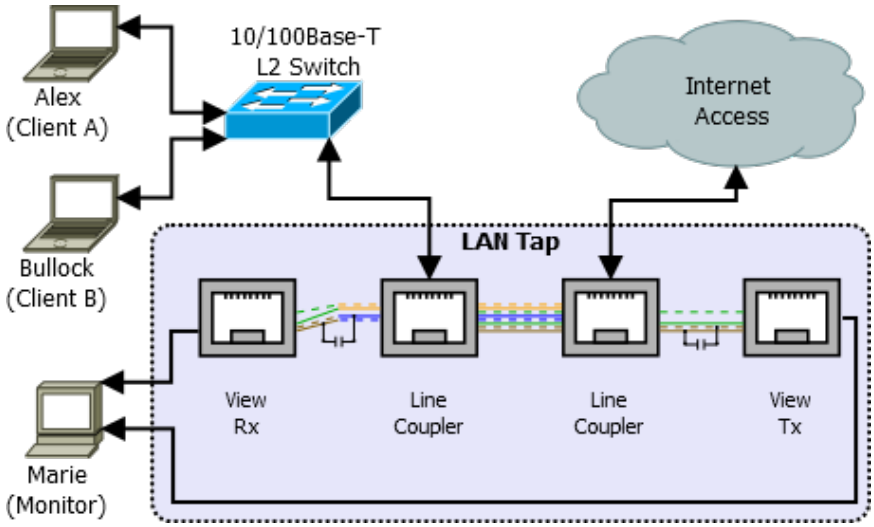


Figure 3.1: Design of network with passive TAP implementation.

3.1.1 Passive LAN Tap

Unfortunately, when using a layer 2 switch, no configuration can be completed on the switch to block accidental packet injection by a monitoring device. The switch is also incapable of mirroring network traffic to a specific port. To account for these issues, a passive TAP was used.

The TAP depicted in Fig. 3.2 and appendix B.1 was chosen over other options due to it's minimal impact on network traffic [30], [31]. This specific TAP was fabricated based on a widely used design B.1. This design also allows for isolated capture of either Transmit (Tx), Receive (Rx), or both directions of network traffic flow as can be noted in F ig. 3.1 by the two network flows directed at M arie. As an added benefit, this style of TAP is physically incapable of injecting traffic onto a network [32] and passively throttles network bandwidth to 100Mb or less.[4]

[4] It is not possible to successfully capture network traffic at gigabit speeds without an active repeater.

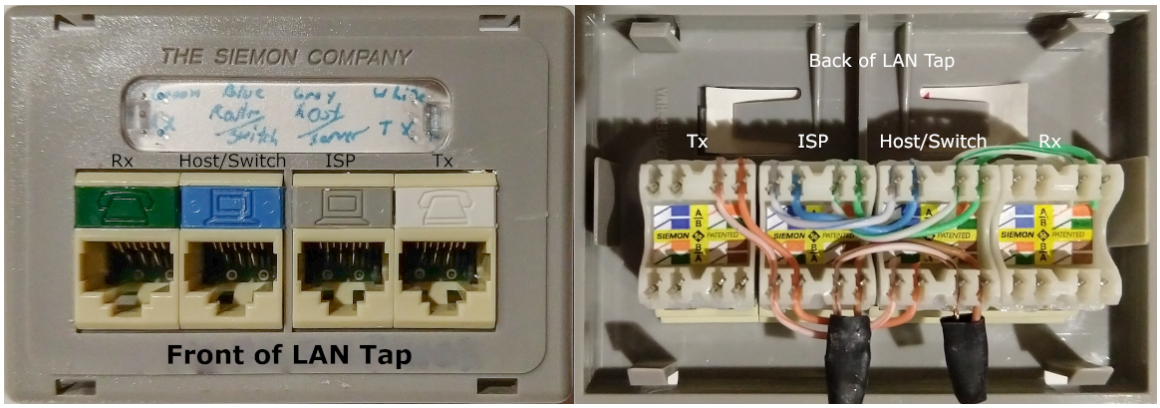


Figure 3.2: Labeled image of passive TAP.

3.1.2 TAP Location Discussion

In theory, a sniffer could be placed anywhere along the path between at least one of the clients and the server at a cost of potential loss of traffic due to possible changes in the packet switching path. However, the fewer hops between the monitor and one of the end points would results in a higher probability of capturing all related network traffic. Therefore, the optimal placement for a network traffic monitor would be directly in-front of the client or the server. This would also serve to reduces the impact any queuing delay might have on packet capture timing.

Future work will need to be conducted to confirm this theory, however it is a reasonable assumption. For the context of this experiment, placing a monitor near the IM server was not feasible, so the only option was to place the TAP near the clients. The TAP was instead placed in such a way that the traffic from both clients could be monitored, as noted in Fig. 3.1.

The TAP could have been placed directly between one of the clients and the switch, but this would have limited its capture to only the traffic between the one client and the server. By placing the TAP between the switch and the access point, we are effectively capturing all communications between both clients and the server. Without having to use multiple TAPs and additional monitor devices. This simplifies the environmental design and reduces the additional resources that would otherwise have been required. In addition, having one monitor device capturing all the traffic, helps insure that timing of captured traffic remains

synchronized throughout the monitor side capture.

It should be noted that if there was a concern that the clients would be communicating directly with each other, a TAP would need to be placed between the client and the switch. This might be applicable to P2P based applications, the web based IM application tested in these experiments operate on a client-server principal. So, no IM traffic or meta-data was passed between clients without first going to the server.

However, there is a possibility of some timing data loss due to the switch being between the TAP and the target clients. This was partially mitigated by specifically selecting an unmanaged switch to reduced packet processing to simple collision avoidance and traffic forwarding. This is not perfect, yet an acceptable trade-off, since loss of any timing information is still kept to a minimum. No timing loss was found to be an issue during these experiments.

It is also worth considering that under normal conditions, it is unlikely that an adversary would be able to monitor both clients simultaneously. However this general method of capture and timing analysis would still apply on a single client basis.

3.1.3 Internet Service Provider (ISP)

For this experiment, substantial effort was taken to acquire an internet facing network connection that was relatively unmonitored. The ISP connection was logically located behind a firewall with no further security between the outbound connection and the cloud. These measures proved to not provide any substantive impact on the experiment.

3.1.4 Client Hardware

Initial consideration, and efforts were taken to try to make use of the same types of hardware and peripherals on both client devices, without success. Fortunately, even thought the differences in the hardware are apparent in the final results, it did not have a meaningfully impact overall. Further discussion on the hardware differences and the impact these differences had can on this can be found in section 5.3.

3.2 Data Collection Types

During this experiment, Three primary forms of data were collected. These were client side keystroke logging, network session traffic packet capture, and TLS session key logs.

3.2.1 Keylogging

To provide accurate metrics when comparing physical keystrokes with network traffic, keylogging software was installed on both clients. The keylogging software used during these experiments, Biologger [33], was written specifically to log the timing of natural keystroke events along side the logged characters. Biologger and Wireshark are both able to capture down to millisecond granularity.

The Biologger provided a means of recording raw keystrokes as seen by the client device. These recordings were then matched to the packet timing to provide a direct comparison of network traffic capture against the client side keystroke actions. It should be noted that Biologger is written in Java, so a Java compiler is needed to run Biologger.

3.2.2 Network Packet Capture

To facilitate network packet capture two tools were used.

- TShark, a Command Line Interface (CLI) version of Wireshark was used on the Monitor.
- The MS Windows CLI tool Network Shell - a Windows CLI tool (netsh) was used on the Windows clients.

3.2.3 TLS Key Logs

To help facilitate network traffic comparison, the session TLS keys need to be capture to decrypt the traffic. This will will become of importance as noted in section 4.2.2. To accomplish this, Mozilla Firefox was chosen as the browser to use during the client web sessions. Firefox is one of the few browsers that allow for easy capture of TLS keys [34].

3.3 Data Generation Points

This section will discuss the specifics of how the clients and the monitor were setup.

3.3.1 Marie

Marie, the primary monitoring device represents an adversary that is capable of sniffing network traffic between at least one of the clients and the IM server. Kali Linux was chosen as the Operating System (OS) for Marie due to having TShark installed by default, and could easily have promiscuous mode enable or disabled.

As noted in Fig. 3.1, Marie will be attached to both the Tx and Rx connections of the TAP on two separate ethernet cards. TShark was run on both Ethernet ports in promiscuous mode to create a combined PCAP of all traffic entering and leaving the switch from the ISP. The TShark was started as the first step in each data capture session, and ran throughout the entire session[5].

3.3.2 Clients

The clients named Alex and Bullock are representative of two separate parties communicating with each other through an IM based service. As shown in Fig. 3.1, both clients are attached to the switch which is then connected through the TAP to the ISP. The only software installed on the clients besides that which comes standard on Windows 10[6] was Mozilla Firefox and Java[7]. The Biologger application [33] and two scripts A.1 and A.2 were copied to the client's local drives.

As a secondary network timing source, network session captures were also taken on the clients. To avoid installing further sofware, netsh was used as the network capture tool. Netsh has been available on MS Windows since Window NT, and provides many useful feature. Unfortunately netsh outputs to a MS Event Trace Log File (ETL) file format, which needs to be converted to PCAP for future analysis. To convert the ETL either the Microsoft PowerShell (PS) script found in Appendix A.2 or the methods describe in the Microsoft Tech Community forum [35] can be used[8].

To provide more consistent results, the batch script shown in Appendix A.1 was written to automate the process. The script gathers and logs a number of useful metrics and automates the majority of the experiment, set up the keylogging and packet capture sessions, and

[5]Example of the command used: # tshark -i eth0 -i eth1 -w M-DDMmmYYYY.pcap.
[6]Windows 10 Consumer Edition.
[7]Java SE Development Kit 13.
[8]The PS method has a bug and only provides to the second granularity.

configure TLS key logging. However, it will not provide the client dialogue needed for each session.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Implementation

This chapter will expound upon the steps taken during data collection, namely the methods used to record the keyboard events and the network traffic, including the timing of these events. It will also discuss the general procedure used to process the data into forms suitable for analysis. The content of the conversation messages was standardized. The keyboard events and network traffic were simultaneously recorded and logged with timestamps.

4.1 Standardizing Message Data

The conversation script in A.3 was utilized in the effort to standardize the message traffic between the clients and the different IM servers. Having a known string to search for within the network traffic aided in parsing the collected transmissions to quickly locate the TCP stream containing message data.

4.2 Capturing Message Data

To determine the existence of a correlation between keyboard events and network traffic events, those keyboard events (including their timing), as well as the network traffic generated (with their own timing) needs to be collected, compared, and analysed.

4.2.1 Keyboard events and timing

The Java keylogger, Biologger v1 (a cross-platform logger that logs keystrokes and other input events through a system-wide hook), was initiated on both clients to collect keyboard event data [33]. The Comma Separated Value (CSV) file created by the keylogger contained mappings of each keyboard event in Unix-time with millisecond timestamps.

4.2.2 Network events and timing

Wireshark was used on Marie to capture traffic through the TAP. netsh, a native MS command-line utility which allows configuring and displaying the status of various network communications [36] was used to capture the Windows clients', Alex and Bullock, relative

network traffic. Using a native utility removed the need to install Wireshark in an effort to reduce unknowns with regard to operating system resource requirements on the clients. Collecting traffic at the clients, as well as at the TAP, enabled correlation of network traffic events and ease of isolation of client initiated traffic.

The netsh standard output in ETL format [36] was converted to PCAP format[9] using the program in appendix A.2. Utilizing the MS Windows environment variable SSLKEYLOGFILE, which was configured in A.1 to collect the TLS keys used in the network communication, a decrypted version of the PCAP files were created. Editcap, a standalone application that comes with Wireshark, was used to facilitate TLS decryption without requiring additional configuration to Wireshark's protocol preferences [37].

4.3 Isolating Network Events of Interest

Network traffic between either client, Alex or Bullock, and any server the Internet Protocol (IP) addresses associated with the IP range(s) assigned to the IM server's parent company was isolated. The TCP Acknowledgement (ACK) packets were also filtered out as seen here in Fig. 4.1.

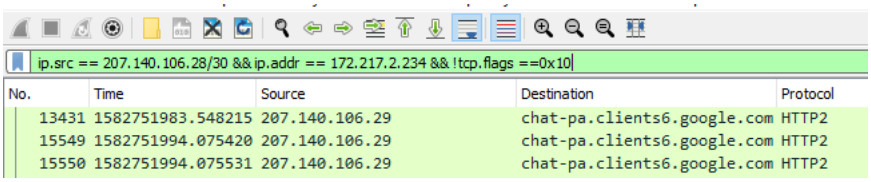


Figure 4.1: Wireshark Filter Used to Isolate Hangouts Network Traffic.

This left only the network traffic containing either message content or control data in their payload to be used for analysis. Using Wireshark's Packet Dissection functionality, a CSV file containing only the metadata of these TCP streams was created for use during analysis of correlations in timing between network events and keyboard events.

[9]The use of an executable found in the MS Tech Community forum [35] replaced the script in the final tests due to being able to convert PCAP with much better time granularity.

4.4 Challenge: Time Synchronization

A challenge that became apparent during the initial data collection was Marie's inability to connect to an Network Time Protocol (NTP) service. Marie was not setup to have direct network or internet access, so NTP could not be relied upon to synchronize the time settings. This time synchronization discrepancy made itself apparent in the PCAP files logging the network events. Since logs recorded by the different systems did not have a shared time, it was a task to correlate the corresponding network traffic with the keystroke logs.

This issue was resolved by having the clients ping the gateway during each session setup process. The times of these pings was recorded by the clients. When compared to the ping times recorded by Marie, a time offset was calculated. Editcap was then used to adjust the timestamps and reconcile the difference [37].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results

This chapter will present the results of analysing the data collected using the general methods described in Chapter 4 and discuss what, if any, vulnerabilities were found for each of the messaging applications discussed in Chapter 2.

5.1 Data Analysis

The decrypted PCAP files were used to search for the network traffic directly associated with the various aspects of message traffic. More specifically, viewing the decrypted traffic allowed pinpointing network traffic with payloads directly pertaining to message content. This same general procedure was performed for the network traffic collected during the different sessions focusing on each IM server.

With regards to message traffic, all three platforms[10] followed the same basic operating procedure. The platforms would establish an individual TCP sessions between the IM server and each client. These sessions were used for all IM based traffic, no side C2 channels were found to carry IM meta-data. The message content data and session control data are transmitted through the same encrypted TCP stream.

In some instances, multiple TCP sessions were observed between the clients and platform servers. Based on analysis of the decrypted traffic within, these other streams were found to be unrelated to the IM session. These extraneous sessions were instead associated with traffic related to other applications[11] associated with the platform. The clients at no point attempted to create sessions directly with each other, all traffic was transmitted through the platform's server first.

5.1.1 Facebook Messenger Data Analysis

Analysis of the decrypted TCP stream showed that any network traffic, originating from the client, with a TLS record length of 95 bytes indicated either the client had recently begun

[10]Facebook Messenger, Google Hangouts, and Kiwi IRC.
[11]For example Facebook and Google login authentication.

typing a message or that they had just sent one (pressed *Enter*). The following frame with a record length of either exactly 636 bytes or one within the range of 962 and 1010 bytes, respectively, determined which of these events occurred. In the latter case, the TLS record of the second packet contained the transmission of the full message. In the case of all the traffic collected during this study, the encapsulating Ethernet frames are 59 bytes longer than TLS records in the payload. This size difference accounts for the Ethernet, IP, and TCP combined header lengths. The graphics throughout this chapter reference the Ethernet frame length.

Analysis of the decrypted TLS record showed only the `body` and `__req` keys varying in size, with the latter by at-most one character. The `__req` appeared to be used as a method of maintaining synchronicity with its value being the hexadecimal value of the total conversation length. The `body` key contained the full message. Subtracting the length of this key (one byte per character) from the length of of its encapsulating frame containing resulted in value of 1021 (+/- 1 depending on the length of the `__req` key). It follows that any properly filtered frame's length over 1021 bytes (TLS record length of 962 bytes) corresponds to the message's character length. Extending the filter shown in Fig. 4.1 to account for frame length (e.g., `frame.len in {1021 .. 1100}`) isolated all of the frames containing message content. No other Ethernet frames, within the PCAP files corpus used for this project, met all of these criteria. With these frames, the length of each message (and the duration of the entire conversation) can be determined as shown in Fig. 5.1. The exception to this rule was when the message contained a non-character, such as an emoji, as seen in Fig. 5.2 with the second occurrence of a frame size of 1051 bytes.

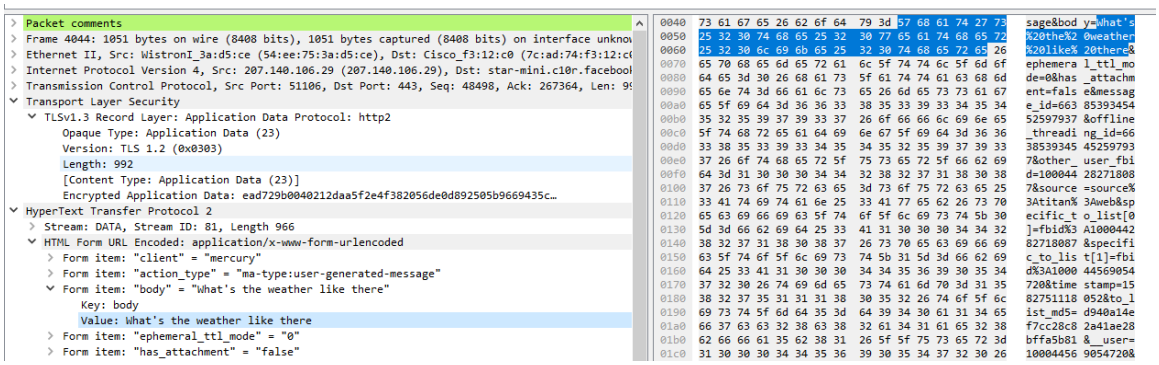


Figure 5.1: The Ethernet Frame (1051 bytes) and TLS Record (992 bytes) of a Single Message (30 bytes) from a Wireshark Capture.

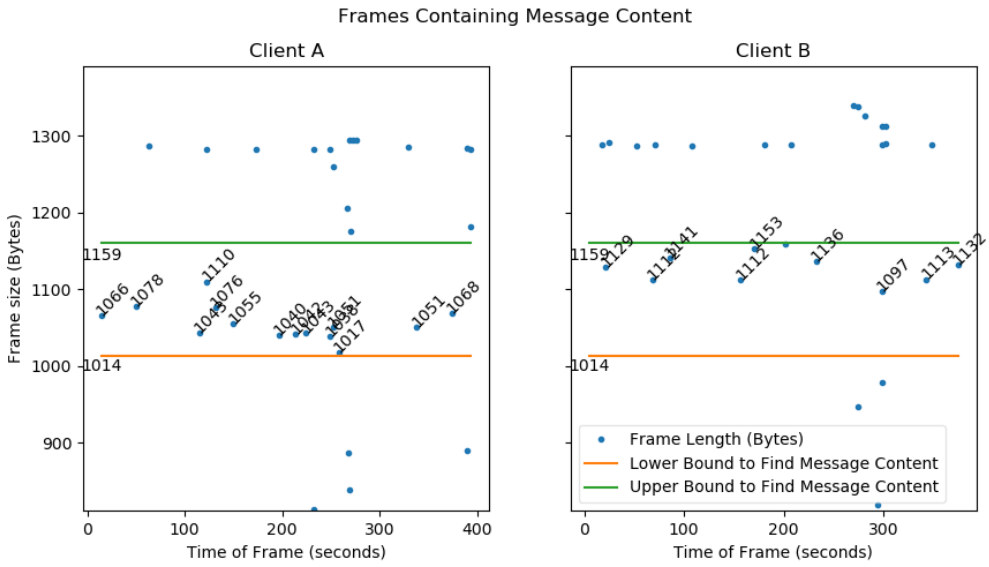


Figure 5.2: Frames containing Facebook message content.

5.1.2 Google Hangouts Data Analysis

Google Hangouts messages containing the transmission of a full message were found to be those with TLS record lengths within the range of 330 to 420 bytes which originate from the client. The size of the TLS record above the base number of 330 bytes corresponded to the number of characters in the transmitted message. The value of 330 bytes as the lower bound was found by subtracting the contained message length from the TLS record length. The upper bound of 420 was used as it corresponded to the length of the longest message sent within the chat message script used in this study. No other network traffic was found to fall within this size range. This range allows for traffic containing messages up to 90 characters to be isolated from any other network traffic.

From this we could consistently detect and identify which encrypted network packets contained message traffic and closely approximate the number of characters contained in the actual message. Little additional information could be determined without a much broader statistical approach. When unencrypted, the traffic shows shows various `null` and `false` values within the TLS containing the message content. These other values could be altered possibly by adjusting font color or style. Changing these values were not attempted during our testing. These kind of changes would have an effect on the overall length of the

TLS record, therefor the encapsulating Ethernet frame's base length.

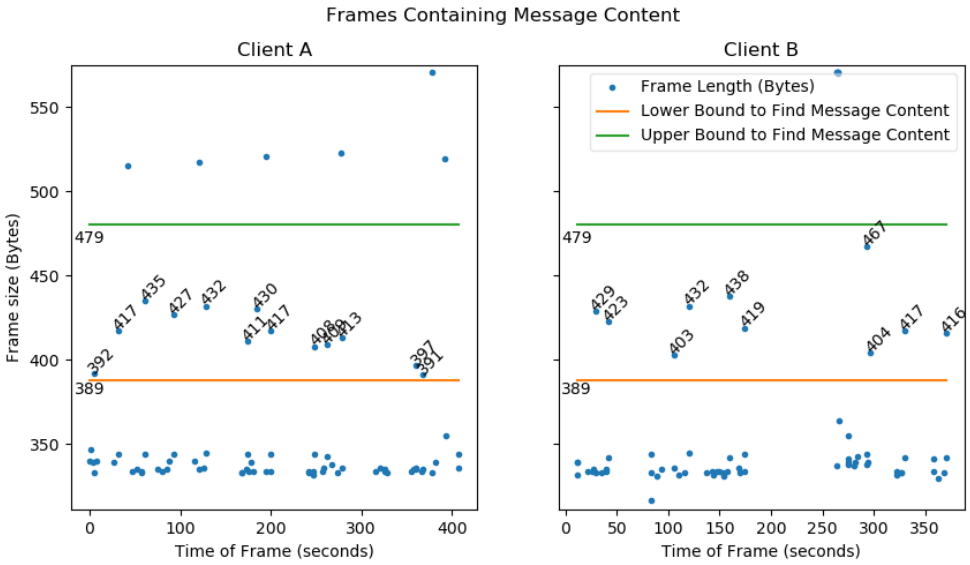


Figure 5.3: Frames Containing Hangouts Message Content.

5.1.3 Kiwi IRC Data Analysis

KiwiIRC has two TLS channels. The sparseness of communications between Client A and Server 2 (45.55.225.44, www.kiwiirc.com) as shown in 5.4 indicate that it merely maintains a connection with the website. Server 1 (104.26.6.99, do-a.clients.kiwiirc.com) manages the message traffic. Kiwi uses so little overhead communications and that the total number of frames is hardly three times the number of messages transmitted. Of those frames, any originating from the client containing TLS record length greater than 74 bytes contain message content. The value of 74 bytes corresponded with the difference in length of the message contained within the TLS record and that of the record itself. As with the other servers, the number of bytes over that base line corresponds to the length of the message, one to one. With the except of the one frame at 175 bytes, no frames larger than those containing message content were transmitted by the client to the messaging server, an upper limit is yet to be shown.

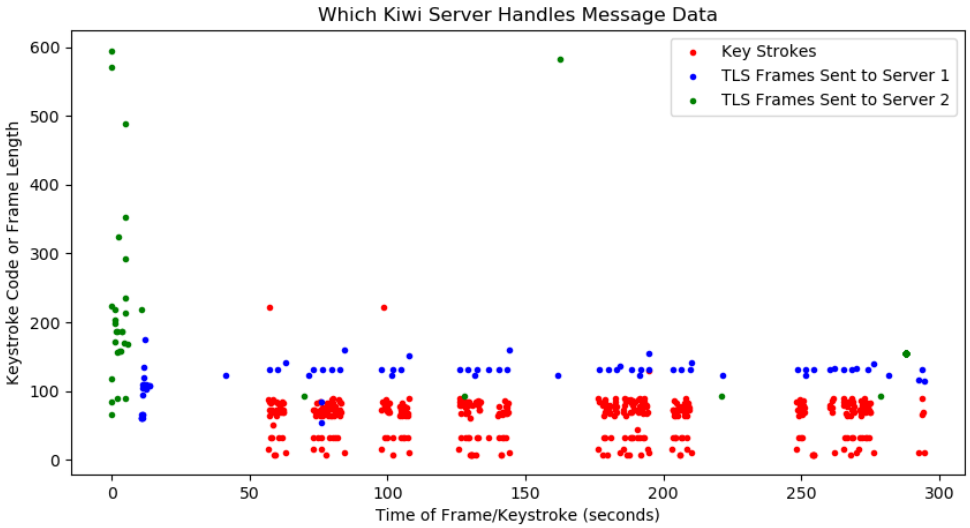


Figure 5.4: Visualizing Keystroke and TLS Traffic Times between Client A and both KiwiIRC servers.

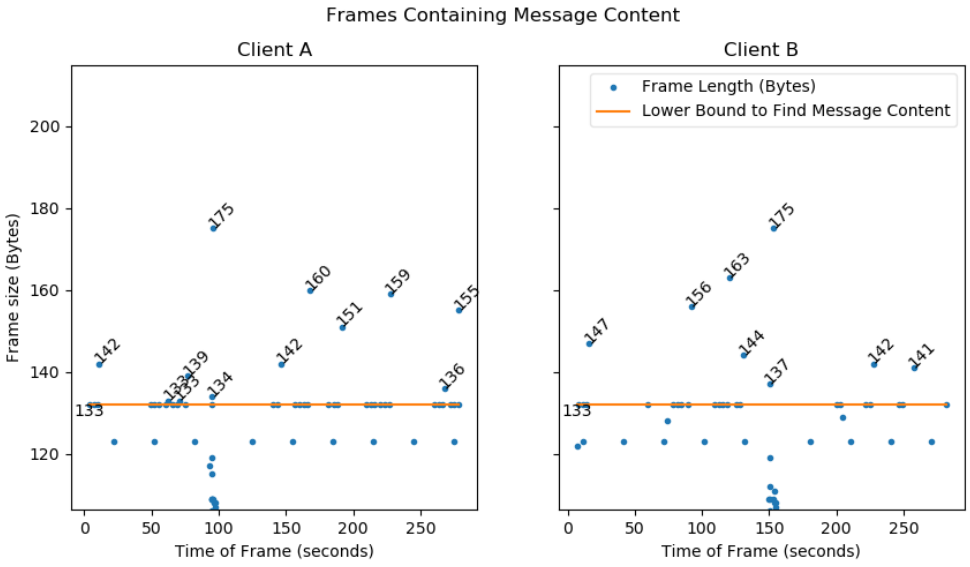


Figure 5.5: Frames Containing Kiwi Message Content.

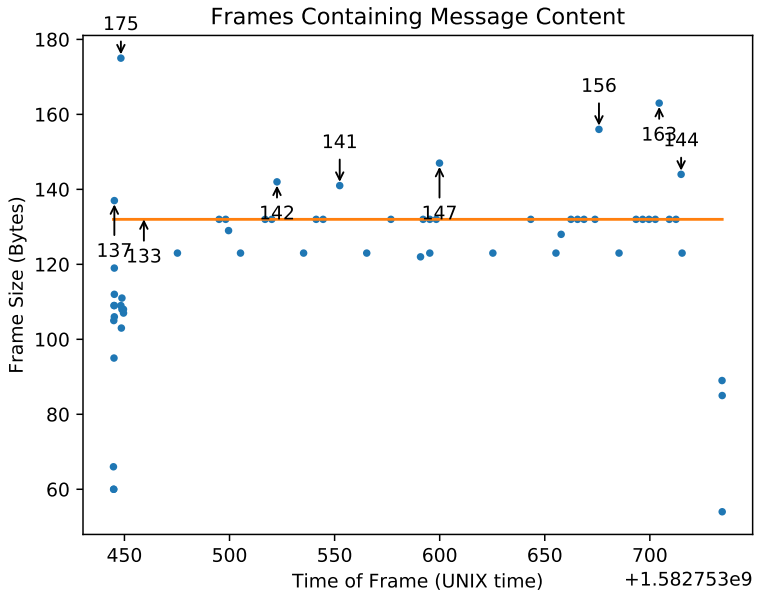


Figure 5.6: Frames Containing Kiwi Message Content from Client B.

5.2 Timing analysis

Comparing the timing of network traffic events with the timing of keyboard events was done by combining the CSV file containing the keyboard events created by the Biologger [33] with the CSV file created from Wireshark's Packet Dissection functionality. Sorting this data by timestamp, it was concluded that there is very minimal network activity correlation to individual keystrokes. Message meta-information (e.g. first keystroke, pausing beyond the server's time-out period, pressing *Enter*) were the only events found to directly cause network traffic. This negates the possibility of direct keystroke analysis based on timing of network traffic. It does not, however, remove the possibility of other potential threats such as user identification (using typing speed comparisons), message length analysis, or device fingerprinting.

On top of this, the periodicity of the network traffic packets was compared to that of the keystrokes. The Python script in A.4 was used to extract the timestamps of key press events as collected by the Biologger [33] and those of related IP packet transmissions. The section below shows the general lack of correlation between the majority of keyboard/client events

and the messaging servers. The only notable exceptions tend to coincide with the first character typed and enter being pressed to send the message.

5.2.1 Facebook Messenger Timing analysis

Fig. 5.7 and Fig. 5.8 compare the periodicity of the keystrokes of Clients A and B, respectively, and that of the traffic sent to the Facebook messaging server from each client. Fig. 5.9 shows the keystroke and TLS traffic times with regard to one message from the collected conversation.

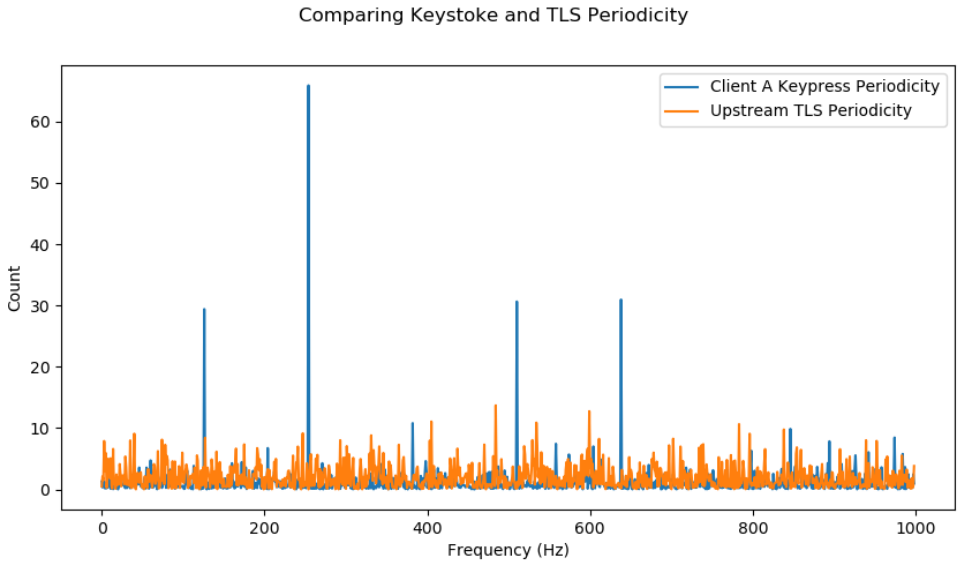


Figure 5.7: Client A keystroke periodicity compared to that of TLS traffic sent to the Facebook messaging server.

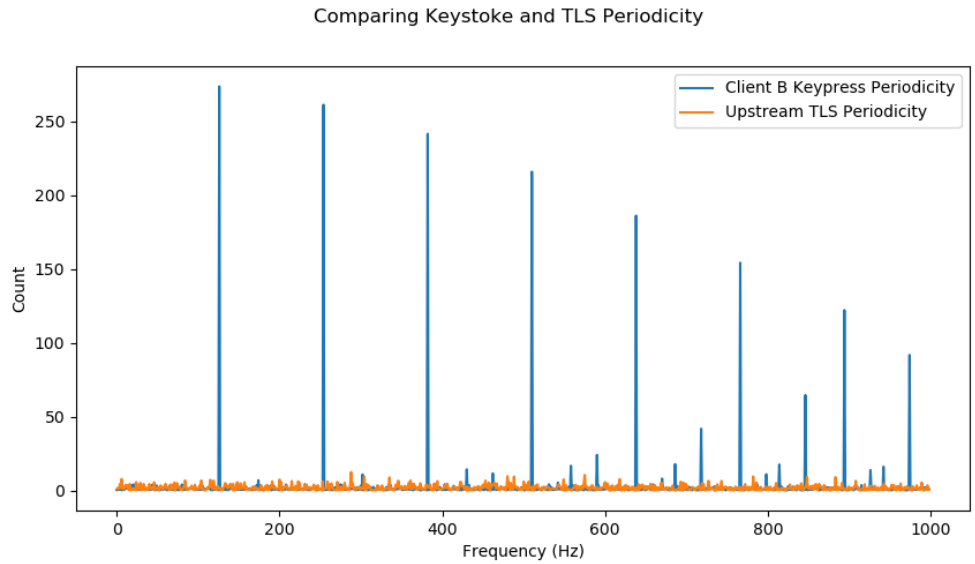


Figure 5.8: Client B keystroke periodicity compared to that of TLS traffic sent to the Facebook messaging server.

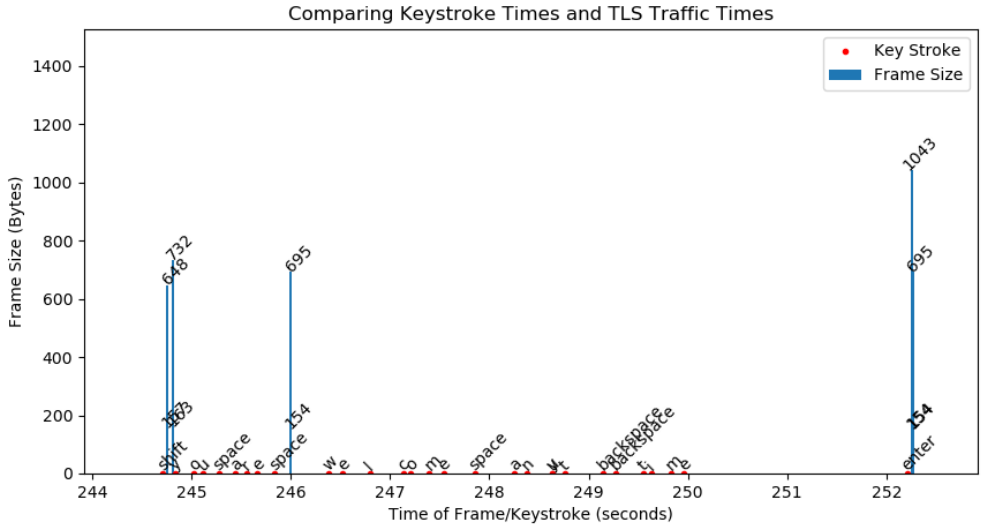
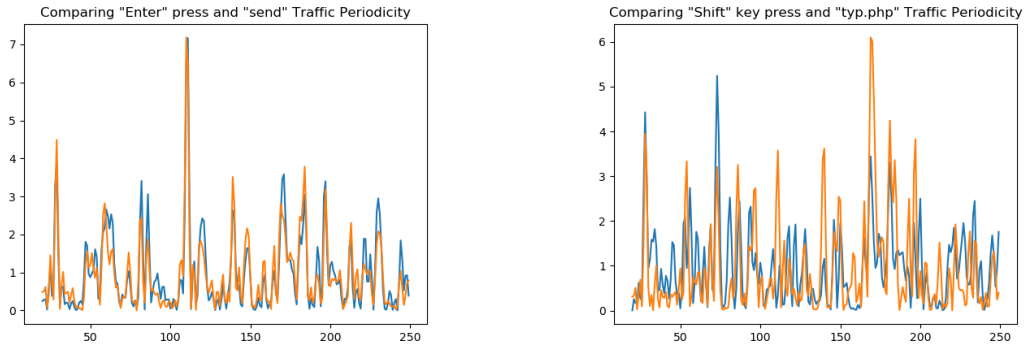


Figure 5.9: Timing of Client A keystrokes compared to the timing and frame size of TLS traffic sent to the Facebook messaging server for one message in the collected conversation.

Reviewing Fig. 5.9, a relationship between the "typing" network packets and *Shift* key. Another trend can be noted with the "send" *Enter* key. These relationships are shown in

Fig. 5.10, the blue lines represents the network traffic, while orange represents the keystroke periodicity.



(a) Enter (b) Shift

Figure 5.10: Facebook client A traffic periodogram charts when filtered for a specific key and message type combination.

5.2.2 Google Hangouts Timing analysis

Fig. 5.11 and Fig. 5.12 compare the periodicity of the keystrokes of Clients A and B, respectively, and that of the traffic sent to the Facebook messaging server from each client. Fig. 5.13 shows the keystroke and TLS traffic times with regard to one message from the collected conversation.

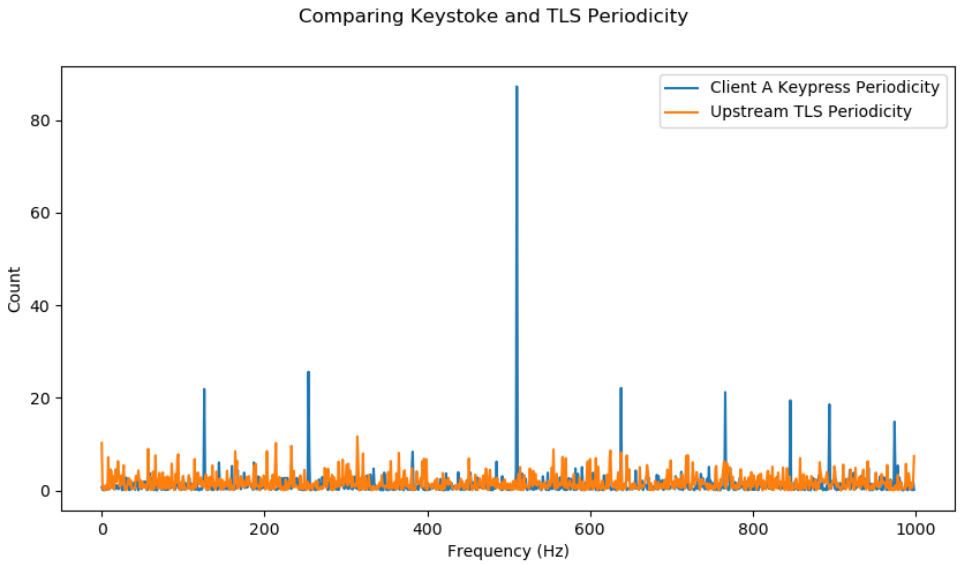


Figure 5.11: Client A keystroke periodicity compared to that of TLS traffic sent to the Google Hangouts server.

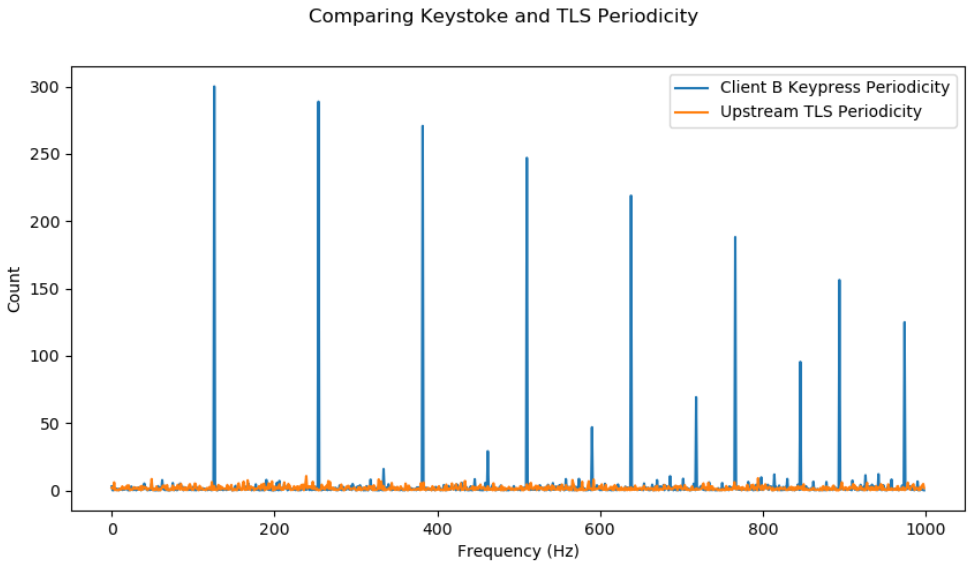


Figure 5.12: Client B keystroke periodicity compared to that of TLS traffic sent to the Google Hangouts server.

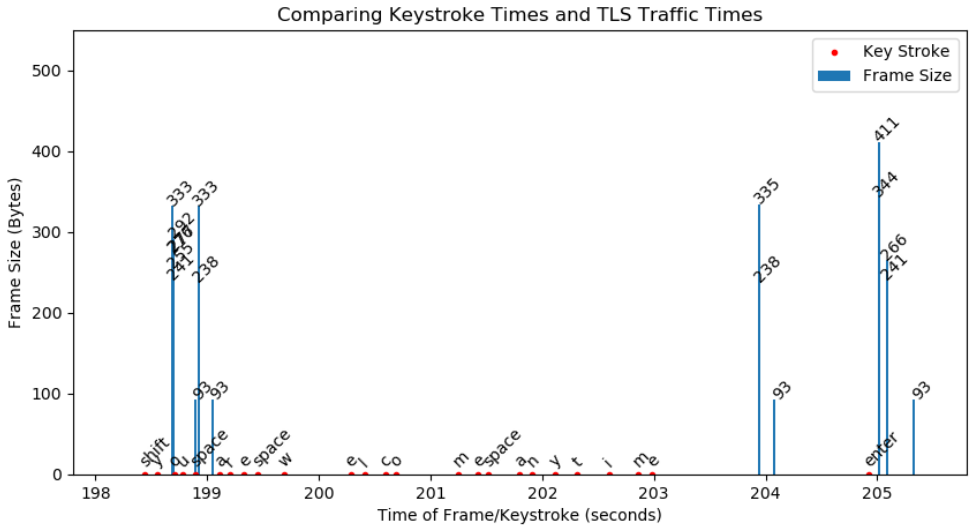
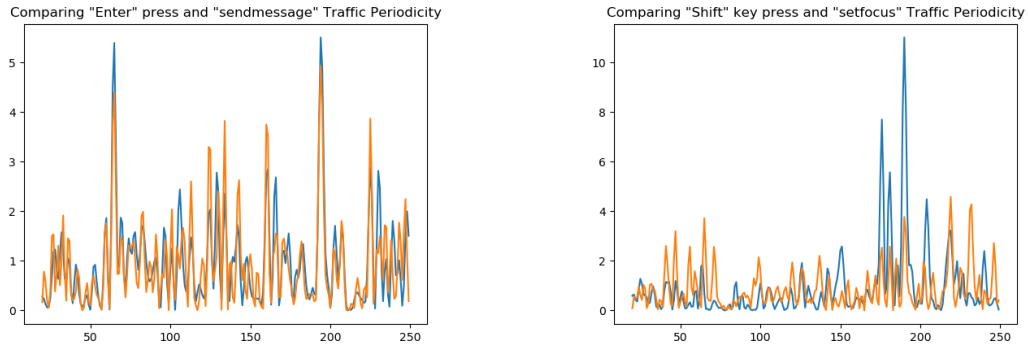


Figure 5.13: Timing of Client A keystrokes compared to the timing and frame size of TLS traffic sent to the Google Hangouts server for one message in the collected conversation.

Reviewing Fig. 5.13, a general relationship between certain keystrokes, such as *Enter* and network traffic. As with Facebook, a Power Spectral Density (PSD) analysis was run between the "sendmessage" packets and the *Enter* keystroke. Reviewing Fig. 5.14 a strong relationship can be noted. Repeating the same analysis between the "setfocus" packets and the *Shift* keystroke provided an interesting set of clusters around the 196hz range for network traffic. However, this cluster does not seem to be closely correlated to the the *Shift* keystroke, at least not as can be seen in the "sendmessage" and *Enter* chart.



(a) Enter

(b) Shift

Figure 5.14: Google Hangouts client A traffic periodogram charts when filtered for a specific key and message type combination.

5.2.3 Kiwi IRC Timing analysis

Fig. 5.15 and Fig. 5.16 compare the periodicity of the keystrokes of Clients A and B, respectively, and that of the traffic sent to the Facebook messaging server from each client. Fig. 5.17 shows the keystroke and TLS traffic times with regard to one message from the collected conversation.

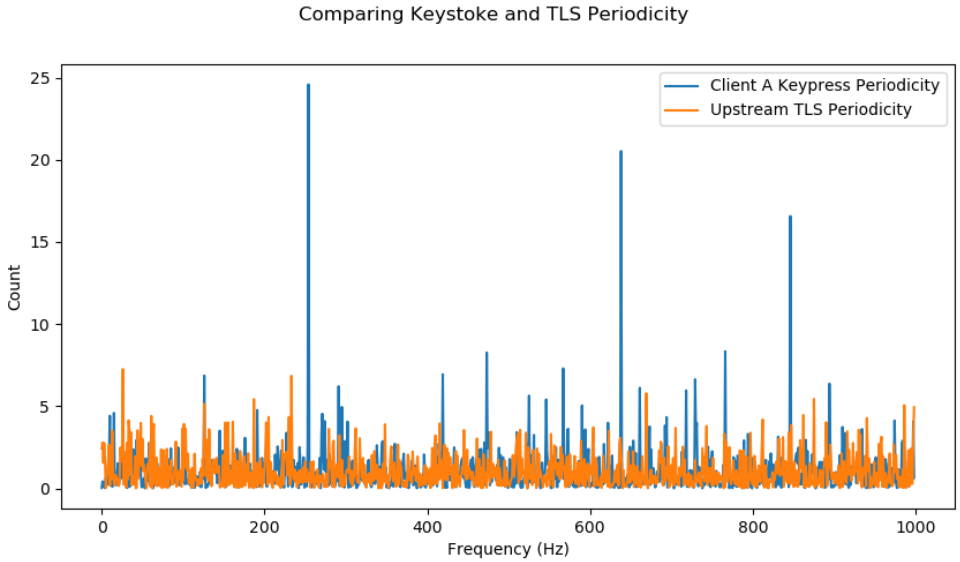


Figure 5.15: Client A keystroke periodicity compared to that of TLS traffic sent to the Kiwi IRC messaging server.

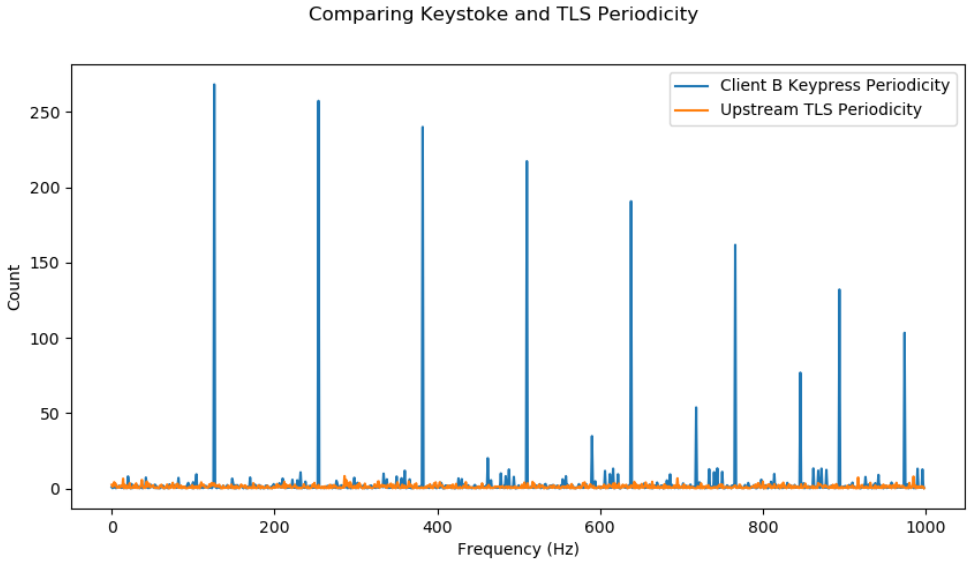


Figure 5.16: Client B keystroke periodicity compared to that of TLS traffic sent to the Kiwi IRC messaging server.

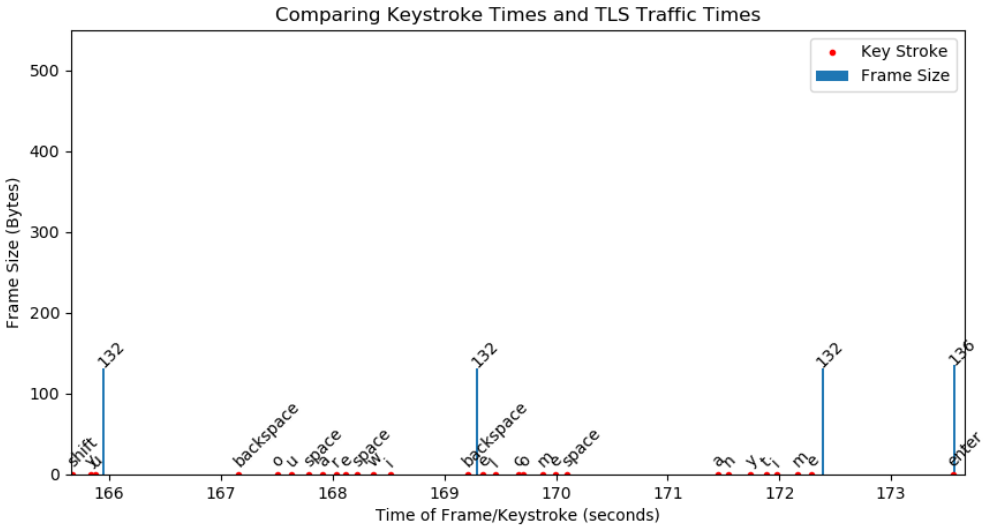


Figure 5.17: Timing of Client A keystrokes compared to the timing and frame size of TLS traffic sent to the Kiwi IRC messaging server for one message in the collected conversation.

When reviewing the network traffic from Kiwi IRC as shown in Fig. 5.17, Only two types of message traffic became apparent. The first is a 132 byte keep-alive message occurring approximately every 3 seconds. The second type of packet was a packet larger then 132 bytes that correlated to the *Enter* keystroke. The relationship between all packets larger then 132 bytes and the *Enter* keystroke can be noted in Fig. 5.18 for client A.

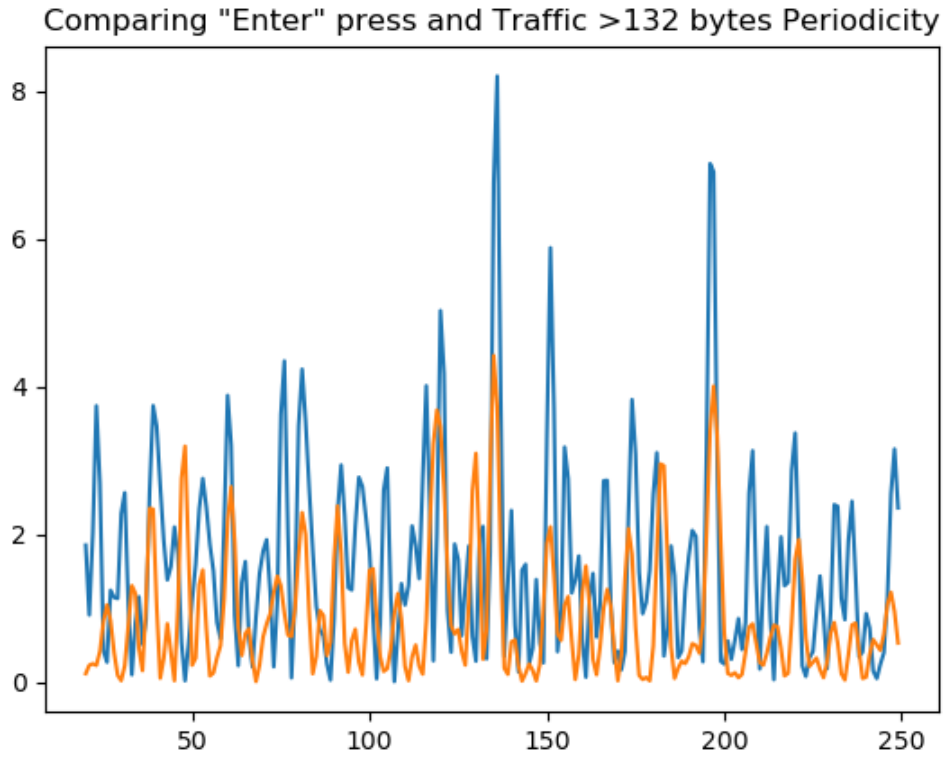


Figure 5.18: Kiwi IRC client A traffic periodogram charts when filtered for a specific key and message type combination.

5.3 Hardware Differences

When reviewing and comparing client keystroke periodicity, such as can be seen in Fig. 5.7 and Fig. 5.8, a distinct difference between the two clients can be noted More specifically, a difference in how the keystrokes are clustered in time frequency. In Fig. 5.12 the keystroke periodicity peaks approximately every 125Hz, converting that to milliseconds that is a Universal Serial Bus (USB) poll rate of approximately 8ms[12]. Reviewing Fig. 5.11 keystroke periodicity appeared to cluster around 500Hz, so the polling frequency was around 2ms.

[12] $Time(s) = \frac{1}{Frequency(Hz)}$

This is an unfortunate side-effect of hardware differences between the two clients. USB peripheral devices can provide input to the processor bus at different frequencies [38]. In addition, different Central Processing Units (CPUs) poll the USB bus at different rates . This results in the actual keystroke being processed by the CPU at different frequencies.

Based on reviewing the periodicity between Alex and Bullock, it would appear that Alex's hardware was able to maintain a closer to real-time processing of keystrokes as they were typed. This is shown by the relatively lower peaks in the graphs, Fig. 5.11 having the highest peak with a count around 90 keystrokes. On the other hand, Bullock's hardware had a bottleneck in processing the keystrokes, resulting in a very defined pattern of clustered keystrokes at approximately 100Hz intervals. The peak count on keystrokes also was on average much higher, Fig. 5.12 having clusters as high as approximately 300 keystrokes.

As can be seen in Fig. 5.13, the difference between keystroke polling and packet transmit times were uncorrelated to the final result. If a correlation between keystrokes and packet timing had been found, further research into how the hardware keystroke polling might impact keystroke timing to user typing might have become necessary. However, it is likely that this may not be an issue. User overall keystroke speeds generally takes longer then 10ms[13], so granularity under 10ms is likely not required.

To verify that under 10ms granularity was not pertinent, we calculated the harmonic mean using $\frac{1}{H_y} = \frac{1}{n}\sum\frac{1}{Y_i}$ and the simple time delta in ms between all the key press and release events during the Hangouts trial for each client. Alex resulted in an average of approximately 97ms and Bullock in 70ms[14]. This is well in excess of 10ms granularity and further shows that a polling frequency of 100Hz or 10ms is more then adequate for our purposes.

[13] A quick demonstration of personal keystroke response times can be found at https://blog.seethis.link/scan-rate-estimator/.

[14] It should be noted that the results from Bullock had 12 cases where the delta was 0ms, and some cases in which the delta was over 1000ms, one case being 74094ms. The 0ms deltas were assumed to be errors and removed from calculation of the harmonic mean.

CHAPTER 6:
Conclusion and Future Work

From our findings we were able to determine a few characteristics with regard to the timing of message traffic sent, but not enough to make it susceptible to keystroke timing based attack vectors.

6.1 Primary Takeaway

1. ***Is encryption used?*** Yes, all three web based IM applications made use of TLS. It is worth noting that all three web based IM applications were solely reliant on TLS. Many modern browsers will enforce the use of HTTPS, so TLS would be used, this feature was not considered, nor disabled during these chat sessions.

2. ***Does the IM web application provide keystroke interaction based traffic?*** No, none of the three IM web applications had direct keystroke to network event correlations for all keystrokes. All three applications did have predictable network traffic associated with the first keystroke and the use of enter for sending the message[15]. Both Facebook and Google made use of timed intervals based polling to check if a user was actively typing. Based on that timing Facebook and Google would send an activity update notification. Kiwi would only send a status notification change at the start of typing then after a message was sent or if the user canceled the message.

3. ***Is the meta-data in-band or out-of-band?*** The meta-data was maintained in-band to the web socket connection used for all parts of the chat session. The IM web clients created an encrypted channel between the clients and the server specific to that chat session, then all traffic pertaining to that session was maintained inside that channel. It was noted that these channels were unique to the chat session between the two clients, not part of the TLS session used for logging into the sites.

[15]No specific tests were made to check if any there were any differences between using *Enter* to send a message compared to using the send button.

4. ***Is there any other element that lends this application to or against this form of attack?*** Since all three applications made use of time based padding, they are not likely to be susceptible to direct keystroke timing attacks. However, encrypted message traffic was still able to provide some useful information.

This study did illuminate a method of consistently separating packets which contained text from those that did not, and detection of some key events. The method of using packet size ranges will need to be further validated against a variety of conditions to confirm wide-scale applicability. This finding was based on backward analysis and predictable behavior under a controlled environment that was not designed to simulate the multitude of conditions found in the wild.

That being said, determining the type of message[16] is possible and could be used to provide filtering, fingerprinting or user identification characteristics. Further, review of the encrypted packets provided a direct or indirect[17] correspondence between packet size the number of characters in a message. This can be used to ascertain the characters length of each message sent, providing a possible method of user fingerprinting. These results are worth future investigation.

6.2 Future Work

The results of our work was limited in scope to accommodate for a focus on a specific genre of application. There are many avenues of future work that would expand our findings[18] for a more accurate representation of the feasibility of testing keystroke timing based attacks against web or cloud based environments.

A wider range of web application testing. For our work, we focused on web based IM applications that did not require a private server or registered account. Many web based IM services that exist either require some form of registered account to use, such as Microsoft Teams. Other services require the ability to host a server, such as Mako Chat.

[16]Messages containing user text vs. C2 based messages.

[17]We found that Facebook Messenger and Kiwi IRC used compression, Google Hangouts did not.

[18]A digital copy of the raw data and associated code can be found published at https://gitlab.nps.edu/maco2020/Chat-Metadata

Fingerprinting of devices by reviewing the jitter between their keystroke and network traffic. The focus of this project centered on what a third party is capable of capturing without access to the device. For this reason limited focus was placed on the finding related to keystroke to network traffic periodicity.

Yet, there does appear to be some evidence relating specific keystroke events with network traffic that might be used to help fingerprint an OS. Indicators like peaks in keystrokes along certain frequencies may indicated a certain type of operating system or hardware is being used. Research into the clock frequencies of various OS and hardware combined with a larger test corpus may reveal a method of device identification.

Mobile devices and locally installed applications. Many IM applications require the installation of a local client onto the device. This is true of most mobile applications and many IM applications must be used over a locally installed client. Additionally, all of the web based applications we tested have equivalent local installation based clients that may function differently then their web based variant. These in particular may be more susceptible to keystroke timing since they are possibly not built with the same bandwidth limitations that a web based client would be.

Cloud based applications that use real-time update features for collaboration. As was eluded to during chapter 1, cloud based applications must send out keystroke based updates that would contain at least some amount of data. We did not focus on these applications, yet concern around these tools security can be easily found with a simple web search. Applying the methodologies used during our experiments to cloud based collaborative tools like Overleaf.com and Microsoft 365 may provide a better assessment of keystroke timing attacks than IM based applications did.

Use of alternate environments. We maintained an environment that was as free of virtualization as possible[19]. However, comparing results between a virtual machine environment (VMWare or similar) to an equivalent non-virtual environment may show an impact on packet forwarding and keystroke polling.

[19]Most modern operating systems make use of some level of virtualization, i.e. Unified Extensible Firmware Interface (UEFI) virtualization and Microsoft Hyper-V

The use of an automated text input and response application might also be implemented to produce more consistent keystroke timing throughout the dialogue. For our experiments, the use of human agents for chat dialogue did not impact our results. However, automating this process would help in trying to find inconsistencies the various applications might have in how much time they take in responding to user input in future studies.

6.3 Conclusion

The use of keystroke timing analysis as a vector for interpreting web based IM client traffic does not appear to be a viable option. This is based heavily on the fact that network events were not being produced on an individual keystroke basis. From a security perspective, the lack of adherence to real-time traffic is what made the difference. More importantly, the methods used and the base findings provide a wide range of future research options and some insight into how encrypted real-time traffic might still be vulnerable.

APPENDIX A:
Scripts

A.1 Auto Capture Setup Script

```
@ECHO OFF
pushd "%~dp0"
::  ========  Simple Auto Capture Setup Script  ========================
SET                        Ver=1.0
::                  by Jeana M. Verkempinck
::           MACO 2020 - Metadata Analysis Capstone
::  Designed to prep and tear-down Windows user enviorment to packet
::  capture using netsh and capture SSL keylog for Firefox
::  Requires:  etl2pcapng.exe or .ps1 converter script in same folder.
::  ================================================================
SET URL=NA
SET MiURL=N
SET FiOwn=N
SET Tfp=NA
TITLE AutoCapture %Ver

:: Auto check to see if batch ran with admin rights.
openfiles>nul 2>&1
if %errorlevel% EQU 0 GOTO BEGIN
CALL :BYE Certain commands will only work with elevated privlages.

:: Module to allow for a fancy menu
:DspTitle
  CLS
  ECHO ************************************************************
  ECHO            Simple Auto Capture Setup Script v.%Ver
  ECHO ************************************************************
  IF NOT [%1]==[] (
  ECHO                 %*
  ECHO ************************************************************
  )
  ECHO.
GOTO :EOF

:: Choice to set client for naming, not necessary for run.
:FiSelect
  CALL :DspTitle Client Choice
  CHOICE /C ABM /M "Select client currently in use: "
  IF ERRORLEVEL 1 SET FiOwn=A
  IF ERRORLEVEL 2 SET FiOwn=B
```

```
41    IF ERRORLEVEL 3 SET FiOwn=M
42  GOTO :EOF
43
44  :: Select which site to initiate test with on Firefox.
45  :URLCHOICE
46    CALL :DspTitle Choose URL
47    ECHO F - Facebook
48    ECHO K - Kiwi (IRC)
49    ECHO G - Google
50    CHOICE /C FKG /M "Select the site to use: "
51    IF ERRORLEVEL 1 SET MiURL=F&& SET URL="https://www.facebook.com/messages/"
52    IF ERRORLEVEL 2 SET MiURL=K&& SET URL="https://kiwiirc.com/nextclient/"
53    IF ERRORLEVEL 3 SET MiURL=G&& SET URL="https://hangouts.google.com/"
54  GOTO :EOF
55
56  :: Module to set and verify the log/folder naming convention.
57  :SetOut
58    CALL :DspTitle Verify Logfile
59    SET Tfp=%MiURL%%FiOwn%-%date:~10,4%%date:~4,2%%date:~7,2%
60    ECHO Directory to store: %userprofile%\Desktop\%Tfp%\ && ECHO.
61    SET /P Tfp=Filename is currently %Tfp%, type in new filename or [ENTER] to continue: ||
        SET Tfp=%Tfp%
62  GOTO :EOF
63
64  :: Short script to add formatted lines to the log file.
65  :LogBreak
66    ECHO ==== Initiated %* At %TIME% on the %date% ====>> %Tfp%-Log.txt
67  GOTO :EOF
68
69  :: Begin setting up the capture environment
70  :BEGIN
71    :: Select the client ID and the URL
72    CALL :FiSelect
73    CALL :URLCHOICE
74
75    :MakeDir
76    :: Sub-lable to initiate the naming convention
77    CALL :SetOut
78
79    :: Make folder if it dosen't already exist, and jump into it.
80    IF NOT EXIST %userprofile%\Desktop\%Tfp% (
81    MKDIR %userprofile%\Desktop\%Tfp%
82    PUSHD %userprofile%\Desktop\%Tfp%
83    ) else (
84      CALL :DspTitle "Error - Folder already Exists"
85      ECHO "Change the File Name to continue" && PAUSE
86      GOTO MakeDir
87    )
88
89    :: Initialize the log with some information useful for filtering.
```

```
CALL :LogBreak Capture
ECHO Initialized at %time% on the %date% >> %Tfp%-Log.txt
SYSTEMINFO >> %Tfp%-Log.txt

CALL :LogBreak IP-Config
netstat -ap tcp >> %Tfp%-Log.txt

:: Pause until all systems ready to begin trace.
CALL :DspTitle Trace for %URL%
ECHO Hit [ENTER] to begin trace. && PAUSE

:: ******** Actually Start the trace ********
:: Set SSL key logging
CALL :LogBreak Set SSL Key Log
CALL :DspTitle Set SSL Log
SETX SSLKEYLOGFILE %userprofile%\Desktop\%Tfp%\%Tfp%-SessionKeys.log
ECHO Set SSLKEYLOGFILE at %time% >> %Tfp%-Log.txt
ECHO. >> %Tfp%-Log.txt

:: Start the packet capture
CALL :DspTitle Starting netsh
netsh trace start persistent=no capture=yes report=no tracefile=%Tfp%.etl >> %Tfp%-Log.
  txt
ECHO Trace started at: %TIME% >> %Tfp%-Log.txt
ECHO. >> %Tfp%-Log.txt

:: Send Pings for monitor to help synch up captures (our local gateway: 207.140.106.1).
CALL :DspTitle Ping Synchronization
ECHO Ping ran for self synchronization.  TTL set at 3. >> %Tfp%-Log.txt
ping /n 3 /i 7 207.140.106.1 >> %Tfp%-Log.txt
ECHO. >> %Tfp%-Log.txt

:: Open Firefox to the previously selected URL.
CALL :DspTitle Start Browser and Keylogger
START "" /d "%programfiles%\Mozilla Firefox" Firefox.exe "%URL%"
ECHO Browers started for %URL%. >> %Tfp%-Log.txt
ECHO. >> %Tfp%-Log.txt

:: Opening keylogger will put rest of script on hold, until keylogger closed.
CALL :DspTitle Begin Test
ECHO Test will auto-complete when the keylogger is closed.
ECHO Keylogger started at: %TIME% >> %Tfp%-Log.txt
START "" /min /wait /d "C:\Program Files\Java\jdk-13.0.1\bin" java.exe -jar c:\biologger
  \biologger1.2.jar -im -iw -o %userprofile%\Desktop\%Tfp%\

:: Rename the Biologger output
REN keystroke.csv %Tfp%-keystroke.csv
REN mouseclick.csv %Tfp%-keymouseclick.csv

:: Cleanup the SSL environment variable (stop logging SSL Keys)
```

```
  SETX SSLKEYLOGFILE ""
  ECHO Reverted SSL Keylog variable at %TIME% >> %Tfp%-Log.txt

  :: Stop packet trace, called last due to how long it takes to stop.
  CALL :DspTitle Merging Trace
  ECHO Merging trace and generating data collection takes a few minutes.
  CALL :LogBreak netsh merge
  START "" /i /wait /b netsh trace stop
  ECHO Finished Merge at %TIME% >> %Tfp%-Log.txt

  :: Convert ETL to PCAP
  CALL :DspTitle Converting ETL to PCAP
  POPD
  :: Using PowerShell: START "" /wait /d %userprofile%\Desktop\MACO-Script PowerShell.exe
    "& {".\ConvertEtlToPcap.ps1" -Path "%Tfp%.etl" -Destination "%Tfp%.pcap"}"
  :: Using .EXE version:
  START /wait etl2pcapng.exe %userprofile%\Desktop\%Tfp%\%Tfp%.etl %userprofile%\Desktop\
    %Tfp%\%Tfp%.pcapng
  PUSHD %userprofile%\Desktop\%Tfp%

  :: Delete CAB and ETL files, generally not needed at this point...
  CHOICE /c:YN /m "Delete CAB and ETL files?: "%1
  IF ERRORLEVEL 2 GOTO SkipDelCab
  IF ERRORLEVEL 1 GOTO DelCab

  :DelCab
    DEL %Tfp%.cab
    DEL %Tfp%.etl

  :SkipDelCab
  CALL :LogBreak Exiting
  CALL :DspTitle Exiting
  Call :BYE Summery: Logs created, Capture Completed and Converted to pcap.
GOTO :EOF

:BYE
  :: Check what type of exit to use
  ECHO %*
  ECHO Exiting in:
  timeout /t 5
  exit
GOTO :EOF
```

Listing A.1: Script to set up capture environment.

A.2 Convert ETL to PCAP PowerShell Script

```
# Usage ".\ConvertEtl-ToPcap.ps1 -Path c:\<path\file>.etl -Destination c:\<path\file>.pcap "
[CmdletBinding()]
param(
[Parameter(Position=0)]
[ValidateScript({
    if( -Not ($_ | Test-Path) ){
        throw "File or folder $_ does not exist"
    }

    if($_.Extension -ne ".etl"){
        throw "Source file must be .etl file"
    }
    return $true
})]
[System.IO.FileInfo]$Path,

[Parameter(Position=1)]
[ValidateScript({
    if( -Not ($path.DirectoryName | Test-Path) ){
        throw "File or folder does not exist"
    }

    if($_.Extension -ne ".pcap") {
        throw "Estination file must be .pcap file"
    }
    return $true
})]
[System.IO.FileInfo]$Destination,

[Parameter(Position=2)]
[Uint32]$MaxPacketSizeBytes = 65536)


$csharp_code = @'
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing.Reader;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace chentiangemalc
{
    public static class NetworkRoutines
    {
    public static long ConvertEtlToPcap(string source, string destination, UInt32 maxPacketSize)
```

```
47          {
48              int result = 0;
49              using (BinaryWriter writer = new BinaryWriter(File.Open(destination, FileMode.
    Create)))
50              {
51                  UInt32 magic_number = 0xa1b2c3d4;
52                  UInt16 version_major = 2;
53                  UInt16 version_minor = 4;
54                  Int32 thiszone = 0;
55                  UInt32 sigfigs = 0;
56                  UInt32 snaplen = maxPacketSize;
57                  UInt32 network = 1; // LINKTYPE_ETHERNET
58                  writer.Write(magic_number);
59                  writer.Write(version_major);
60                  writer.Write(version_minor);
61                  writer.Write(thiszone);
62                  writer.Write(sigfigs);
63                  writer.Write(snaplen);
64                  writer.Write(network);
65                  long c = 0;
66                  long t = 0;
67                  using (var reader = new EventLogReader(source, PathType.FilePath))
68                  {
69                      EventRecord record;
70                      while ((record = reader.ReadEvent()) != null)
71                      {
72                          c++;
73                          t++;
74                          if (c == 10000)
75                          {
76                              Console.WriteLine(String.Format("Processed {0} events with {1}
     packets processed",t,result));
77                              c = 0;
78                          }
79                          using (record)
80                          {
81                              if (record.ProviderName == "
    Microsoft-Windows-NDIS-PacketCapture")
82                              {
83                                  result++;
84                                  DateTime timeCreated = (DateTime)record.TimeCreated;
85                                  UInt32 ts_sec = (UInt32)((timeCreated.Subtract(new
    DateTime(1970, 1, 1))).TotalSeconds);
86                                  UInt32 ts_usec = (UInt32)(((timeCreated.Subtract(new
    DateTime(1970, 1, 1))).TotalMilliseconds) - ((UInt32)((timeCreated.Subtract(new
    DateTime(1970, 1, 1))).TotalSeconds * 1000))) * 1000;
87                                  UInt32 incl_len = (UInt32)record.Properties[2].Value;
88                                  if (incl_len > maxPacketSize)
89                                  {
```

```
                                    Console.WriteLine(String.Format("Packet size of {0}
    exceeded max packet size {1}, packet ignored",incl_len,maxPacketSize));
                                }
                                UInt32 orig_len = incl_len;
                                writer.Write(ts_sec);
                                writer.Write(ts_usec);
                                writer.Write(incl_len);
                                writer.Write(orig_len);
                                writer.Write((byte[])record.Properties[3].Value);
                            }
                        }
                    }
                }
            }
            return result;
        }
    }
}
'@

Add-Type -Type $csharp_code

$result = [chentiangemalc.NetworkRoutines]::ConvertEtlToPcap($Path.FullName,$Destination.
    FullName,$MaxPacketSizeBytes)

Write-Host "$result packets converted."
```

Listing A.2: PS script for ETL capture to PCAP conversion. Adapted from [39].

A.3 User Chat Dialogue

```
===== Session [A/B/M]-DDMmmYYYY =======
A and B chat windows open
[Begin collect on A, B, and M]

A: What's the weather like there [ENTER]
B: It's always cold [ENTER]

A: I heard that California always had nice weather [ENTER]
B: It's colder that I would like [ENTER]

A: It's snowing here [PAUSE 3 seconds] and has been all day [ENTER]
B: I wouldn't mind some seasons [PAUSE 5 seconds] [ENTER]

A: You could come out to visit [PAUSE 3 seconds] and be even colder [ENTER]
B: haha [ENTER]
B: I might just take up on that offer [ENTER]

A: You are welcome anytime [ENTER]
A: If you were to visit, when would you come? [ENTER]

[PAUSE 5 seconds]

A: I have a break in early April [ENTER]

B: That could work [PAUSE 7 seconds, or until notification ends on A] [ENTER]

B: I've never been to New England in the Spring [ENTER]

A: That would be [PAUSE 5 seconds] lovely [ENTER]

[B types what A type simultaneously]
B: I'll start looking into getting that time off work [ENTER]
A: You can stay with me [ENTER] I have plenty of room here [ENTER]

[Individually]
B: Sounds good, talk to you later.
A: Ok, bye.

[End collect on A and B]

===== Session A1-DDMmmYYYY =======
A sending to B WITHOUT B's window open
[Begin collect on A]

A: If you were to visit, when would you come? [ENTER]

[PAUSE 5 seconds]

```

```
A: I have a break in early April [ENTER]
[End collect on A]

===== Session B1-DDMmmYYYY =======
B opens chat window to receive A's previous messages
[Begin collect on B]

B: That could work [PAUSE 7 seconds] [ENTER]
B: I've never been to New England in the Spring [ENTER]

===== Session A1a-DDMmmYYYY =======

[Begin collect on A]
[A logs in and receives B's previous messages]
A: That would be [PAUSE 5 seconds] lovely [ENTER]
B: I'll start looking into getting that time off work [ENTER]

A: You can stay with me [ENTER]
A: I have plenty of room here [ENTER]

[End collect on A, B, and M]
```

Listing A.3: Standardized typing script to allow for easier cross-referencing between multiple network capture files

A.4 Periodogram Python Script

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from glob import glob
from pathlib import Path

def periodogram(t):
    f = np.linspace(0, 500, 1001)
    f = f[f>=1]
    P = np.zeros(len(f), dtype=np.complex)
    for k in range(len(f)):
        P[k] = np.exp(2j * np.pi * f[k] * t).sum()
    P = (np.abs(P)**2)/len(t)
    return P

def get_file(title="Choose a file", type="get",
             filetypes="", ifile="", ext="", keyword=None):
    from tkinter import Tk
    from tkinter.filedialog import askopenfilename, asksaveasfilename
    from pathlib import Path
    app = Tk()
    app.withdraw() itshape#itshape itshapeNoitshape itshapeTkitshape itshapeGUI
    filetypes = [
        ("csv files","*.csv"),
        ("excel files","*.xls*"),
        ("all files","*.*"),
        ]
    if keyword:
        filetypes.insert(0, ("keyword", "*"+keyword+"*"))

    if type == "get":
        file = askopenfilename(
                title = title,
                filetypes = filetypes,
                initialfile = ifile,
                )

    elif type == "save":
        file = asksaveasfilename(
                title = title,
                filetypes = (
                    ("all files","*.*"),
                    ("pdf files","*.pdf"),
                    ),
                initialfile = ifile,
                defaultextension = ext,
                )

```

```
    if not len(file):
        from sys import exit
        exit()

    app.destroy()
    return Path(file)

def get_data(file, columns):
    if file.suffix == ".csv":
        data = pd.read_csv(file)
    elif file.suffix[:4] == '.xls':
        data = pd.read_excel(file)
    else:
        return []
    if isinstance(columns, (list, tuple)):
        cdata = []
        for column in columns:
            cdata.append(data[column])
    else:
        cdata = data[columns]
    return cdata

def main():
    show = True
    save = False

    for client in ["A", "B"]:
        files = [
            Path(glob(r"*"+client+r"*keystroke*.csv")[0]),
            Path(glob(r"*"+client+r"*tls*.csv")[0])
            ]

        if save:
            ext = 'png'
            outfile_name = files[0].stem[:12] + "compare_periods"
            outfile = get_file("Save as...", type="save", ifile=outfile_name, ext=ext)

        keystimes = periodogram(get_data(files[0], "press_time")/1000)
        tlsdata = periodogram(get_data(files[1], "Time"))

        fig = plt.figure(figsize=(10, 5))
        plt.xlabel("Frequency (Hz)")
        plt.ylabel("Count")
        fig.suptitle('Comparing Keystoke and TLS Periodicity')

        plt.plot(keystimes, label="Client A Keypress Periodicity")
        plt.plot(tlsdata, label="Upstream TLS Periodicity")
        plt.legend()

        if save:
```

```
99              plt.savefig(outfile, format=ext)
100
101         if show:
102             plt.show()
103
104 if __name__ == "__main__":
105     main()
```

Listing A.4: Python script for creating graphs of time intervals of network packets and keystroke.

APPENDIX B:
Schematics

B.1 Throwing Star LAN Tap Schematic

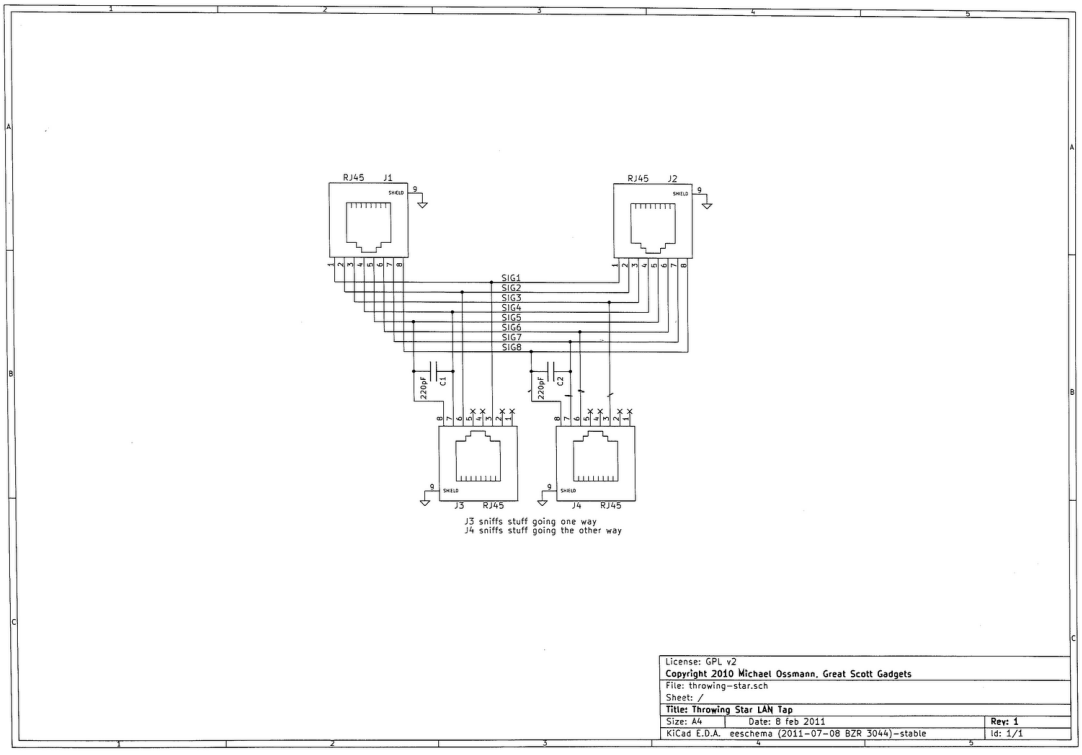


Figure B.1: Throwing Star LAN Tap Schematic. Adapted from [40].

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] S. S. Bender and H. J. Postley, "Key sequence rhythm recognition system and method," U.S. Patent 7 206 938, Apr. 17, 2007. Available: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7206938.PN.&OS=PN/7206938&RS=PN/7206938

[2] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking - MobiCom '15*. ACM Press, 2015, p. 90–102. Available: http://dl.acm.org/citation.cfm?doid=2789168.2790109

[3] J. V. Monaco, "Feasibility of a keystroke timing attack on search engines with autocomplete," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, May 2019, p. 212–217. Available: https://ieeexplore.ieee.org/document/8844619/

[4] N. J. Heacox, R. A. Moore, J. G. Morrison, and R. F. Yturralde, "Real-time online communications: 'chat' use in navy operations," in *2004 Command and Control Research and Technology Symposium*. Pacific Science & Engineering Group Inc, June 2004, p. 12. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a465828.pdf

[5] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock around the clock: Time-based device fingerprinting," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (CCS '18). New York, NY, USA: Association for Computing Machinery, 2018, p. 1502–1514. Available: https://doi.org/10.1145/3243734.3243796

[6] Techopedia. (2017, June). What is a Web-Based Application? - Definition from Techopedia. [Online]. Available: https://www.techopedia.com/definition/26002/web-based-application

[7] H. Fei, L. Rui, H. Liting, and B. Yu, "Analysis and characteristic at the chat session level in instant message traffic," in *2009 First International Conference on Information Science and Engineering*, Apr. 2009, pp. 1666–1669.

[8] K. Shuang and K. Feng, "Research on server push methods in web browser based instant messaging applications," *Journal of Software*, vol. 8, no. 10, pp. 2644–2651, Oct. 2013. Available: http://www.jsoftware.us/vol8/jsw0810-36.pdf

[9] H. T. Chu, L. Hsieeh, and W. S. Chen, "A novel design of instant messaging service extended from short message service with XMPP," *Int'l. Journal of Comp Networks*

and Apps., vol. 2, no. 1, pp. 35–40, Feb. 2015. Available: https://www.ijcna.org/Manuscripts/Volume-2/Issue-1/Vol-2-issue-1-M-05.pdf

[10] S. Saad, M. U. Ilyas, K. Khurshid, and H. Radha, "IM session identification by outlier detection in cross-correlation functions," in *2015 Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2015, pp. 1–5. Available: https://www.researchgate.net/publication/274635819_IM_Session_Identification_by_Outlier_Detection_in_Cross-correlation_Functions

[11] Z. Liu, G. Shu, and D. Lee, "Instant messaging security," in *Network Security, Administration and Management: Advancing Technology and Practice*, D. C. Kar and M. R. Syed, Eds. Hershey, PA: IGI Global, June 2011, ch. 15, pp. 288–323. Available: https://www.igi-global.com/viewtitlesample.aspx?id=54211&ptid=49582&t=Instant%20Messaging%20Security&isxn=9781609607777

[12] A. Wacker, G. Schiele, S. Holzapfel, and W. T., "A nat traversal mechanism for peer-to-peer networks," in *2008 Eighth International Conference on Peer-to-Peer Computing*, Sept. 2008, pp. 81–83.

[13] Computer Weekly News. (2015, Jan.). Instant Messaging Activity Notification. *Computer Weekly News*. [Online]. Available: http://libproxy.nps.edu/login?url=https://search-proquest-com.libproxy.nps.edu/docview/1645257434?accountid=12702

[14] N. A. Barghouthy and H. E. Said, "Social networks IM forensics: Encryption analysis," *Journal of Communication*, vol. 8, no. 11, pp. 708–715, Nov. 2013. Available: https://www.semanticscholar.org/paper/Social-Networks-IM-Forensics%3A-Encryption-Analysis-Barghouthy-Said/605fe9611a653eb6db2e74466ad833edce906f95

[15] M. Husák, M. Cermák, T. Jirsík, and P. Celeda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," *EURASIP Journal on Information Security*, Feb. 2016. Available: https://doi.org/10.1186/s13635-016-0030-7

[16] Cisco, "Collaboration instant messaging and presence," in *Cisco Collaboration System 12.x Solution Reference Network Designs*, 12th ed., Mar. 2018, pp. 1–72. Available: https://www.cisco.com/c/en/us/td/docs/voice_ip_comm/cucm/srnd/collab12/collab12/presence.html

[17] V. Wang, P. Moskovits, and F. Salim, *The Definitive Guide to HTML5 WebSocket*, 2013. Available: https://b-ok.org/book/2072127/9d6c32

[18] Facebook. Privacy & safety. [Online]. Available: https://www.messenger.com/privacy

[19] K. Wong, A. C. T. Lai, J. C. K. Yeung, W. L. Lee, and P. H. Chan, "Facebook forensics," Valkyrie-X Security Research Group (VXRL), Tech. Rep., July 2011. Available: https://www.fbiic.gov/public/2011/jul/Facebook_Forensics-Finalized.pdf

[20] Facebook, *Send API - Messenger Platform - Documentation*. Available: https://developers.facebook.com/docs/messenger-platform/reference/send-api

[21] N. A. Mutawa, I. A. Awadhi, I. M. Baggili, and A. Marrington, "Forensic artifacts of facebook's instant messaging service," in *6th International Conference on Internet Technology and Secured Transactions*, Dec. 2011, pp. 771–776. Available: https://www.semanticscholar.org/paper/Forensic-artifacts-of-Facebook's-instant-messaging-Mutawa-Awadhi/3050a040be5fb3a9095c3753a2cb27695dc7ecf2

[22] T. C. Sottek. (2013, May). Google unveils Hangouts: a unified messaging system for Android, iOS, and Chrome. [Online]. Available: https://www.theverge.com/2013/5/15/4332556/google-hangouts-unified-messaging-google-io-2013

[23] P. Higgins. Google abandons open standards for Instant Messaging. [Online]. Available: https://www.eff.org/deeplinks/2013/05/google-abandons-open-standards-instant-messaging

[24] C. de Looper. Google will begin shutting down the classic Hangouts app in October. [Online]. Available: https://www.digitaltrends.com/social-media/google-hangouts-shut-down-2020/

[25] Kiwi IRC. Powerful features built directly in your Web IRC client. [Online]. Available: https://kiwiirc.com/

[26] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: A metric for measuring information leakage," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 106–117. Available: https://ieeexplore.ieee.org/document/6237010

[27] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *2010 IEEE Symposium on Security and Privacy*, May 2010, p. 191–206.

[28] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, R. Spreitzer, and S. Mangard, "Keydrown: Eliminating keystroke timing side-channel attacks," *arXiv.org*, Jun. 2017. Available: http://arxiv.org/abs/1706.06381

[29] G. Shah, A. Molina, and M. Blaze, "Keyboards and covert channels," in *Proceedings of the 15th Conference on USENIX Security Symposium – Volume 15*, July 2006, pp. 59–75.

[30] H. Schut, M. Scanlon, J. Farina, and N. Le-Khac, "Towards the Forensic Identification and Investigation of Cloud Hosted Servers through Non-Invasive Wiretaps," in *2015 10th International Conference on Availability, Reliability and Security*. IEEE, Aug. 2015, pp. 249–257. Available: https://arxiv.org/pdf/1510.00664v1.pdf

[31] J. Karunaratne, "The passive splice network tap," Dec. 2009. Available: https://janitha.com/articles/passive-splice-network-tap

[32] D. G. Gómez. (2003, June). Receive-only UTP cables and network taps. [Online]. Available: https://dgonzalez.net/papers/roc/roc.pdf

[33] J. V. Monaco, *BioLogger*, Jan. 2020, accessed Jan. 20, 2020. Available: https://github.com/vmonaco/biologger

[34] *Debugging TLS issues with Wireshark*, June 2019. Available: https://lekensteyn.nl/files/wireshark-tls-debugging-sharkfest19us.pdf

[35] S. Greenbaum. (2020, Jan.). Converting ETL Files to PCAP Files. [Online]. Available: https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/converting-etl-files-to-pcap-files/ba-p/1133297#

[36] Microsoft, *Netsh Command Syntax, Contexts, and Formatting*. Available: https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts

[37] Wireshark, *editcap – Edit and/or translate the format of capture files*. Available: https://www.wireshark.org/docs/man-pages/editcap.html

[38] D. Luu. (2017, Oct). Keyboard Latency. [Online]. Available: https://danluu.com/keyboard-latency/

[39] M. McCaffery. (2018, Oct.). Convert netsh trace ETL to PCAP with PowerShell. [Online]. Available: https://chentiangemalc.wordpress.com/2018/10/08/convert-netsh-trace-etl-to-pcap-with-powershell/

[40] M. Ossman. (2011, Sept.). Throwing Star LAN Tap schematic. [Online]. Available: https://windywindycitytech.files.wordpress.com/2011/09/tapschematic.gif

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California