LIQUID: A DETECTION RESISTANT COVERT TIMING CHANNEL BASED ON IPD SHAPING

by

ROBERT J. WALLS

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2009

To my father, who made all of this possible.

ACKNOWLEDGEMENTS

ABSTRACT

LIQUID: A DETECTION RESISTANT COVERT TIMING CHANNEL

BASED ON IPD SHAPING

ROBERT J. WALLS, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Matthew Wright

Covert timing channels provide a way to surreptitiously leak information from an entity in a higher-security level to an entity in a lower level. The difficulty of detecting or eliminating such channels makes them a desirable choice for adversaries that value stealth over throughput. When one considers the possibility of such channels transmitting information across network boundaries, the threat becomes even more acute. A promising technique for detecting covert timing channels focuses on using entropy-based tests. This method is able to reliably detect known covert timing channels by using a combination of entropy and conditional entropy to detect anomalies in shape and regularity, respectively. This dual approach is intended to make entropy-based detection robust against both current and future channels. In this work, we show that entropy-based detection can be defeated by a channel that intelligently manipulates the metrics used for detection. Specifically, we propose a new covert channel that uses a portion of the inter-packet delays in a compromised stream to smooth out the distortions detected by the entropy test. Our experimental results suggest that this channel can successfully evade entropy-based detection and other known tests while maintaining reasonable throughput. Furthermore,

we investigate the effects of parameter selection on the channel. We introduce a model for analyzing the effect of our techniques on the entropy of the channel and empirically investigate the accuracy of the model.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Covert timing channels provide a way to surreptitiously leak information from an entity in a higher-security level to an entity in a lower level. The difficulty of detecting or eliminating such channels makes them a desirable choice for adversaries that value stealth over throughput. When one considers the possibility of such channels transmitting information across network boundaries, the threat becomes even more acute.

For example, imagine the computer network inside of a military research laboratory. The internal machines in such a scenario would likely hold confidential information. In order to protect this sensitive data, the lab would employ an array of sophisticated defense mechanisms to monitor and limit all traffic entering and leaving the network. Now imagine that an attacker (potentially with the assistance of a malicious insider) has managed to infiltrate the lab and gain control of a machine. The question then becomes, how does the attacker remove the data without being detected? Any attempts to make an unauthorized connection with an external server would quickly be detected. However, an attacker could hide the stolen information in the packet timing of a legitimate network stream. This avenue of exfiltration is referred to as a covert timing channel.

Covert timing channels are especially hard to detect because the actual content of the packets is not modified. Furthermore, the destination of the compromised stream may not necessarily be the receiver of the covert information. The receiver could be positioned anywhere along the path as long as it is able to observe the timing information.

Previous work on defending against covert timing channels has focused on either eliminating the possibility of a channel or detecting the presence of a channel. Eliminating

a covert timing channel typically involves removing all of the timing information for a network stream by buffering the packets. Such schemes penalize all traffic, including those streams without an embedded covert timing channel. As a result, detection may be a more preferable alternative. Additionally, it may be useful to know when a covert channel is present, as this is a sign that a machine has been compromised. However, detecting covert timing channels is a very difficult problem. Most detection tests fail against one or more known timing channels.

Gianvecchio et. al. [1] propose a promising detection technique that focuses on using the entropy and conditional entropy of a stream. They show that by using a combination of tests for channel regularity and shape, they can reliably detect all known timing channels. This dual approach is intended to make entropy-based detection robust against both current and future channels.

In this thesis, we further demonstrate the difficulty of detecting covert timing channels. We show that even entropy-based detection can be defeated by a channel that intelligently manipulates the metrics used for detection. Specifically, we propose a new covert channel, Liquid, that uses a portion of the inter-packet delays in a compromised stream to smooth out the distortions detected by the entropy test. We demonstrate that our channel is able to evade detection by current tests while still maintaining acceptable capacity.

Furthermore, we investigate the effects of parameter selection on the channel. The number of inter-packet delays used for shaping versus the number used for transmitting the message is an important consideration when trying to balance between the detection resistance and capacity of Liquid. To better understand this tradeoff, we introduce a model for analyzing the effect of our techniques on the entropy of the channel and investigate the accuracy of our analysis. We also use our model to estimate the channel

parameters needed to obtain certain levels of detection resistance and empirically validate the results.

The rest of this thesis is organized as follows. The next chapter reviews existing covert channels and defenses. In Chapter 3, we present the design of Liquid and the parameter estimation framework. Chapter 4 details our experimental setup. Chapter 5 evaluates the detection resistance of our proposed channel. Chapter 6 concludes the paper and discusses future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter we describe the different types of covert channels and provide examples of each. We also discuss proposed defenses against covert channels.

2.1 Background

A covert channel is defined as a communications medium, unintended by the system designer, that an attacker can utilize to transmit hidden messages from an entity in a higher security zone to an entity in a lower security zone [2]. In this context, we define a hidden message as a sequence of symbols embedded into the channel.

Covert channels can be divided into two broad categories, timing and storage. Covert channels that modulate timing information for a shared resource or process are known as timing channels, while covert channels that alter the content of a resource are referred to as storage channels [2]. Both of these types of channels can be used in either a single-system or networked environment. We define a covert channel in a single-system environment as any channel that passes hidden messages between entities on the same machine, whereas a channel in a networked environment is one that transmits the message between machines – potentially across network boundaries.

Possible storage channels in a single-system environment include the alteration of a filename to contain specific symbols, or the presence of a file in a predetermined directory [2]. On the other hand, an example of a storage channel in a network environment is the use of a packet header field to hide a message [3].

This thesis focuses on timing channels used in a network environment. From this point forward, any reference to a covert timing channel (CTC), unless otherwise stated, will refer to those channels that operate in a network environment.

In general, CTCs modulate the time period between two consecutive packets in a network stream in order to encode a symbol. We refer to this period as the inter-packet delay (IPD). Since the IPD is the only part of the stream that is modified, CTCs are effective regardless of the actual packet payload, even if it is encrypted. However, this also means that at least one packet must be sent for each symbol. As such, the capacity of a CTC is significantly lower than standard communication protocols such as FTP.

We refer to the node that encodes the message into the target network stream as the sender and the node that decodes the message as the receiver. In order for the transmission of the message to be successful, the receiver must be positioned so that it is able to observe the IPDs of the stream at some point during its transit through the network. It is important to note that the receiver is not necessarily the final destination of the stream. In addition, the receiver must also have a means to identify which stream contains the covert channel.

Since the CTC is being used to move data from a higher security zone to a lower security zone, it is logical to assume that only simplex communication is available due to the presence of firewalls, or the inability of the receiver to modify the stream. This provides us with a worst case scenario as the receiver cannot communicate directly with the sender. As a result, synchronization and error correction become more difficult since the receiver cannot inform the sender of its current status.

2.1.1 Examples of Covert Timing Channels

We can further classify Covert Timing Channels as either passive or active [1]. Passive CTCs only modify the IPDs of existing network streams to encode the message,

i.e., they do not generate any additional traffic. Conversely, active channels generate new traffic with IPDs that match the symbols of the message. Intuitively, passive channels are harder to detect since they use legitimate streams, and can thus evade intrusion detection systems and monitoring; however, they are also dependent on a process that the attacker may not control. This means they sacrifice capacity in exchange for increased detection resistance. Examples of both active and passive CTCs are given below.

2.1.1.1 Storage IP Simple Covert Channel

The Storage IP Simple Covert Channel (Storage IP SCC) proposed in [4] uses a simple binary encoding scheme in which the channel transmits a bit by sending (or not sending) a packet in a given time interval. This interval is known by both the sender and receiver. The specific choice of timing interval must be made so as to balance channel capacity with the frequency of bit errors. If the interval is too small, network jitter could cause bits to be flipped, corrupting the message. Conversely, if the interval is too large, the capacity of the channel may be too small to be considered practical.

The timing interval may be a static value, or it could be dynamic and transmitted by the sender using a storage channel. In the latter case, the storage IP SCC can adjust the timing interval based on current network conditions. Furthermore, to increase its detection resistance, the storage IP SCC can rotate the interval and inject a predetermined level of noise by sending legitimate IPDs.

2.1.1.2 Timing IP Simple Covert Channel

The Timing IP Simple Covert Channel (Timing IP SCC) maps an arbitrary number of symbols to specific IPD values [4]. Similar to the timing interval of the storage IP SCC, this mapping must be known beforehand. In the simple case of a binary scheme, a distinct IPD value must be assigned to represent both the 0 and 1 bits. We can denote

these values as IPD_0 and IPD_1 respectively. In order to transmit a 1 bit, the channel sends a packet such that the inter-packet delay between the current and previous packets will be equal to IPD_1. A similar process is used encode a 0 bit. However, since only two different values are used to encode the bits, the timing IP SCC has low detection resistance [4].

2.1.1.3 Model Based Covert Timing Channel

Gianvecchio et al. proposed a framework for creating a CTC that attempts to mimic the statistical properties of legitimate network streams [5]. The framework is used to create a model of legitimate traffic, which in turn helps determine the properties of the CTC. We refer to this as a Model Based CTC (MBCTC).

In order to construct the MBCTC, the framework is used to first analyze a target type of traffic and fit that traffic to a distribution. The message is then split into symbols that are mapped to IPDs based on the inverse distribution function of the chosen distribution. Finally, packets are sent using the calculated IPDs. Decoding is performed using the cumulative distribution function. The distribution can be changed over time to reflect any changes in the target traffic.

2.1.1.4 Jitterbug Covert Timing Channel

Shah et al. proposed using a small hardware device to create a covert timing channel [6]. This device, referred to as a Jitterbug, sits like an adapter between a machine and its keyboard. This placement allows it to selectively capture and delay each key stroke made by the machine's user. In interactive network applications, the timing information of key strokes could potentially be used to affect the timing information of the actual network stream. Jitterbug was designed to exploit this property in order to embed CTCs into these streams. One such network application is SSH. In the interactive mode of SSH,

every keystroke in a terminal causes a packet to be sent immediately [7]. An interesting advantage of using a Jitterbug is that the actual machine is never compromised and, as a result, scanning the machine would not reveal its presence.

Jitterbug uses a timing window w to determine the delay required for encoding each symbol. The timing window is set so as to balance the number of errors that are caused by network jitter and channel capacity. In this respect, w is similar to the timing interval in the storage IP SCC. Furthermore, Jitterbug uses a pseudo-random sequence, s, so that the IPDs do not cluster around multiples of w. Each value in s is in the range $[0, w-1]$.

In the case of a binary code, a Jitterbug encodes the symbols by delaying a key stroke such that the resulting IPD satisfies the following equation:

$$(IPD_i - s_i) \bmod w = \begin{cases} 0 & \pm \frac{w}{4} & \text{for 0 bit} \\ \frac{w}{2} & \pm \frac{w}{4} & \text{for 1 bit} \end{cases}$$

2.1.1.5 Watermarking

Watermarking is used to associate a target network stream with a sender by embedding a recognizable pattern into the IPDs of the stream [8, 9, 10]. It is different from traditional timing channels in that the objective is not to leak captured information, but to correlate network streams across multiple hops. Watermarking has been used, both, to violate the confidentially of users across anonymous networks [8] and to trace back attackers through stepping stone connections[9].

2.1.2 Defense Against Covert Channels

Defenses against covert timing channels can be categorized as either prevention or detection techniques. Prevention is concerned with either eliminating the possibility of

a channel, or reducing the capacity and thus rendering the channel impractical. On the other hand, detection techniques attempt to identify active covert channels.

Kemmerer proposed the use of a Shared Resource Matrix to help identify resources and entities that could potentially be utilized by covert channels [2, 11]. This matrix is intended to be used at design time to assist system architects when creating multi-level secure systems. Hu [12] suggested the use of adding noise to system clocks to reduce the capacity of a timing channel. A more aggressive prevention technique, the Pump, was proposed in [13, 14]. The Pump is placed between two processes so that any communication between them can be intercepted and re-sent based on a randomization scheme to perturb timing information.

Active Wardens, proposed by Fisk et al. [3], are designed to remove storage channels from objects that have strict format definitions. These definitions allow the content of the objects to be objectively verified. A practical example of such an object would be the packet header definition of a network protocol. Similar to the Pump, an Active Warden intercepts each of these objects as they are being transmitted. Before the objects are forwarded to their destination, the Warden applies specific rules that may alter packet data in order to make the object's content more consistent. For example, a rule may specify that an unused header field must contain all zeros. When applying this rule, the Warden would zero all of the bits in that field.

Burke et al.[15, 16] investigated a simple statistical method for detecting covert timing channels. This method assumes that the IPDs for a network stream roughly fit a normal distribution. Therefore, if we create a histogram of the IPDs with each bin having an equal range, we would expect the bin that contains the mean IPD value to have the largest count of IPDs. It follows that the presence of a bimodal or multimodal distribution would suggest the existence of a covert timing channel. For example, in the histogram of an IP Simple Covert Channel, the bins for values IPD_0 and IPD_1 will have

the largest count. The mean IPD value, however, will lie somewhere in between these peaks. Using this method, the probability that a stream contains a covert channel can be defined as:

$$P_{CovChan} = 1 - \left(\frac{C_\mu}{C_{max}} \right)$$

where C_μ is the count of the bin with the mean delay, and C_{max} is the count of the largest bin.

Another detection method is based on the variance of IPDs in a network stream [17]. For a legitimate stream, the variation would be expected to change over time. In this method, the sequential IPDs for the stream are split into a series of fixed-size, non-overlapping windows, with a standard deviation calculated for each. The regularity of the variance is then calculated using the following equation:

$$Regularity = STDEV \left(\frac{|\sigma_i - \sigma_j|}{\sigma_i}, \quad i < j, \quad \forall i, j \right)$$

where σ_I is the standard deviation for the ith window.

The Sigma Similarity method sorts the IPDs for a network stream in order of increasing size and attempts to find large relative differences between them [17]. A large percentage of relative differences below some value Sigma suggests the use of a CTC, since we would expect the IPDs to cluster around multiples of the timing interval.

Stillman [18] suggested finding correlations between the IPDs of a network stream and the memory content on a compromised machine in order to detect CTCs. The proposed method creates a histogram of the IPDs, that is then used to estimate the values used to encode each symbol in a binary scheme. If the histogram is bimodal, the system will assume that the first mode is used to encode one symbol, while the second mode is used to encode the other. Otherwise, the system will assume all values below the mean represent one symbol and all values above the mean represent the other. A binary sequence is then generated using the suspected IPD values of the symbols and

the stream's IPDs. Correlation between the calculated binary sequence and the memory content of currently running processes indicates a covert timing channel.

2.1.2.1 Kolmogorov-Smirnov Test

Gianvecchio et al. [1] used the Kolmogorov-Smirnov test (KS Test) to check if the distribution of a sample set of IPDs matched that of a legitimate set of IPDs. A difference in the distributions would suggest the presence of a CTC in the sample. The KS Test is a non-parametric test that can be used to determine if two samples are from the same distribution. One major advantage of this test is that it does not rely on any assumptions in regards to the actual distribution of the samples.

The Kolmogorov-Smirnov test statistic, D, quantifies the maximum distance between the empirical distribution functions of the two samples.

$$D = Max|F_1(i) - F_2(i)|$$

Since the KS Test directly compares the empirical distribution functions, the samples do not need to be the same size.

CHAPTER 3

SYSTEM DESIGN

In this chapter we discuss a promising detection technique based on entropy and conditional entropy. We then propose a new covert channel that can evade this and all other known detection tests. Finally, we present a model for analyzing the effect of parameter selection on the entropy of the channel.

3.1 Entropy Based Detection

Gianvecchio et al.[1] propose using the entropy and conditional entropy of a network stream to detect covert timing channels. Entropy could be used to detect timing channels that caused differences in shape, whereas conditional entropy could be used to detect the regularity of a channel. They postulated that a combination of these two metrics would be effective against known timing channels and robust against future channels. They showed that entropy-based detection could be used to detect well-known timing channels such as IPCTC and the Jitterbug CTC.

3.1.1 Entropy

Given a sample of sequential IPDs from a network stream, each IPD can be mapped to one of a finite set of M possible symbols, with the probability of the ith symbol given as P_i. The entropy, H, of the sample is then calculated as [19]:

$$H = -\sum_{i=1}^{M} P_i log_2 P_i.$$

(3.1)

It follows from this equation that the entropy is maximized when all of the symbols are equally likely, i.e. $P_i = \frac{1}{M}$ for all i. Conversely, the entropy is minimized (zero) when only symbol is possible.

In order to use entropy to detect covert timing channels, there must be a difference in symbol probabilities between the covert and legitimate streams such that the difference results in decreased entropy for the covert stream. This is clearly true in the case of timing IP SCC, because only two IPD values are used to encode the covert message. Since each IPD value maps to only one symbol, only two symbols are used for the hidden message — much fewer than would be expected in a legitimate stream. Consequently, the probability for all the other symbols in the timing IP SCC stream is near zero.

The only way to ensure that this probability difference results in a loss of entropy for the covert stream is by mapping IPDs to symbols in such a way that a legitimate stream's entropy is maximal. Gianvecchio et al. accomplish this by constructing a histogram of IPDs with M bins corresponding to the M possible symbols. They set the ranges of the bins using a large training set of IPDs such that each bin has an equal number of training IPDs, resulting in each symbol having equal probability. If the IPDs of a legitimate stream are binned using the histogram, they would be expected to have uniform distribution among the bins. Since all symbols are equally likely, the entropy is maximal.

In order to detect the presence of a covert timing channel, samples of IPDs are tested to check if their entropy is lower than that of a legitimate stream. The entropy of each sample is calculated using the equation given above, with the probability of each symbol defined as the number of occurrences of that symbol within the sample divided by the sample size.

3.1.2 Conditional Entropy

The effectiveness of using entropy to detect covert timing channels is limited by the fact that the entropy of a sample of IPDs is the same regardless of the actual order of the IPDs. As a result, entropy does not measure the regularity that may be introduced by a covert channel. To compensate for this deficiency, Gianvecchio et al. propose the use of conditional entropy to measure any abnormal correlations between the IPDs of a stream.

Given a sequence of symbols (bin numbers) X, such that X_i is the ith value in the sequence, the conditional entropy of symbol X_i given $X_1..X_{i-1}$ is

$$H(X_i|X_1..X_{i-1}) = H(X_1..X_i) - H(X_1..X_{i-1}).$$

3.1.3 Corrected Entropy

Due to problems with limited sample sizes, Gianvecchio et al. advance the use of corrected entropy (CEN) instead of entropy. CEN is calculated as follows:

$$CEN = H + perc(X_1) * H, \tag{3.2}$$

where $perc(X_1)$ is the percentage of training bins that contain exactly one IPD from the test sample. As we will see in the following sections and in Chapter 5, this value plays a significant role in detecting covert channels.

They similarly use corrected conditional entropy (CCE) instead of conditional entropy:

$$CCE(X_i|X_{i-1}) = H(X_i|X_{i-1}) + perc(X_i) * H, \tag{3.3}$$

where $perc(X_i)$ is the percentage of unique subsequences of symbols of length i.

For the remainder of this thesis, we shorten the notation $perc(X_1)$ to $perc$ for simplicity.

3.2 Liquid

The design goal for Liquid was to create a timing channel that could successfully evade detection by all of the currently known tests, with a focus on defeating entropy-based detection. We decided to use the Jitterbug CTC as a base for Liquid due to its unique attack vector and because, to the best of our knowledge, it successfully evades all detection methods except the corrected entropy test.

The intuition behind Liquid is to use some portion of the IPDs in a compromised stream to smooth out the symbol probability distortions detected by the corrected entropy test. These IPDs would be generated such that they would increase the probability of symbols that are not used by Jitterbug for the transmitting the hidden message. The intended result is that the probability of each symbol present in the stream will be approximately equal, maximizing the entropy. Furthermore, we would like to increase the number of bins that contain exactly one sample IPD, thereby increasing the value of *perc*. We can see from Equation 3.2 that the CEN value is proportional to the value of *perc*. We refer to any IPDs used to meet these criteria as shaping IPDs. However, Liquid, like Jitterbug, is a passive covert timing channel, and as such does not generate any additional traffic. This means that the shaping IPDs can only be created by adding additional delay to outgoing packets.

Intuitively, delaying a single packet P_i changes two different IPDs; the IPD between packets P_{i-1} and P_i increases, while the IPD between packets P_i and P_{i+1} decreases. We denote these delays as IPD_{i-1} and IPD_i, respectively. This means that it is possible for Liquid to send both small and large shaping IPDs by only delaying packets. We can illustrate this property with a simple example. Given the unmodified legitimate stream depicted in Figure 3.1(a), we can see that the inter-packet delays between P_1, P_2, and P_3, are $IPD_1 = IPD_2 = 3$. If we delay P_2 by one time unit such that it is now sent at time $t = 4$(see Figure 3.1(b)) we can see that IPD_1 is increased by one time unit whereas
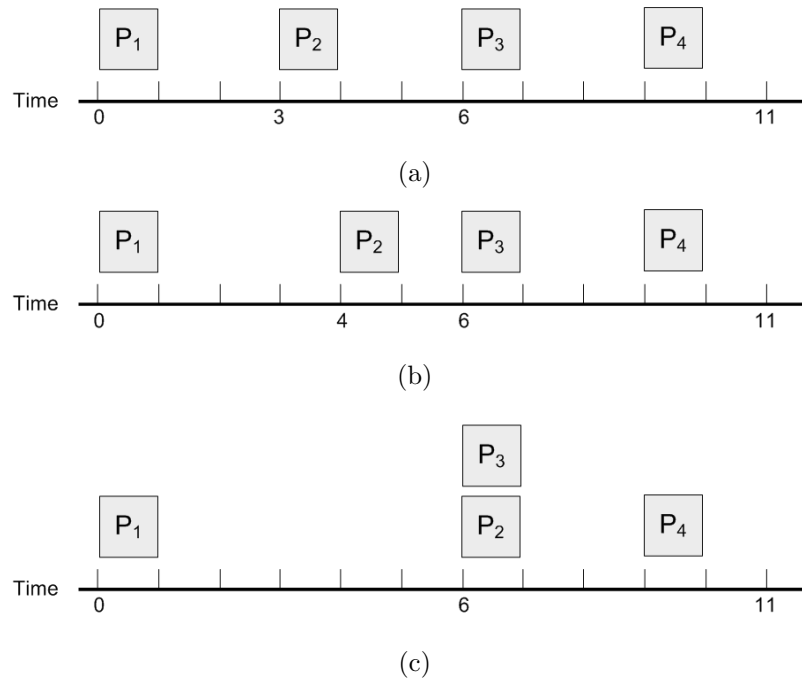
Figure 3.1. Inter-packet Delays. (a) Original Packet Stream. (b) Delaying Packet 2. (c) Causing Packet 3 To Be Buffered.

IPD_2 is decreased by one time unit. The new IPD values are $IPD_1 = 4$ and $IPD_2 = 2$. Furthermore, large delays can cause succeeding packets to be buffered, reducing the inter-packet delay to near zero. This is shown in Figure 3.1(c) where $IPD_2 = 0$.

Liquid splits the network stream IPDs into two distinct types: the transmit IPDs used for sending the hidden message and the shaping IPDs defined above. The transmit IPDs are used to encode the hidden message in a manner similar to a Jitterbug CTC, with a timing window, rotating sequence, and binary code. The difference being the presence, or absence, of transmit noise. In order to provide even greater detection resistance, we can add random amounts of sub-millisecond noise to each transmit IPD. For example, rather than sending out a transmit IPD with a value of 15.000ms, Liquid would send it out with a value of 15.398ms. When noise is not used, the transmit IPDs are encoded exactly like Jitterbug.

$$(IPD_i - s_i) \bmod w = \begin{cases} 0 & \pm\frac{w}{4} & \text{for 0 bit} \\[2ex] \frac{w}{2} & \pm\frac{w}{4} & \text{for 1 bit} \end{cases}$$

The additional delays added for both transmit and shaping IPDs are bounded by the size of the timing window.

As mentioned above, the value of the each shaping IPD is dependent on the current symbol probabilities for the channel. In order to estimate these probabilities, Liquid uses a set of equally probable bins similar to those used in entropy-based detection. The number of bins is set to match that of the entropy test and the bin ranges are determined using a large set of legitimate training IPDs. Although not implemented in our current design, Liquid could also modify the bin ranges over time to better model the target stream. Liquid then determines the bin for each IPD (both transmit and shaping) and keeps track of the count for each bin. Furthermore, it will periodically reset the count for each bin to be more sensitive to the current conditions of the channel. The shaping IPDs are determined such that they minimize the following cost function:

$$Penalty_{ipd} = Penalty_{distance} + Penalty_{bin},$$

where the $Penalty_{distance}$ is the calculated as the additional delay multiplied by a constant, α:

$$Penalty_{distance} = (IPD_{covert} - IPD_{original}) * \alpha,$$

and $Penalty_{bin}$ is calculated as the count of the covert IPD's bin multiplied by a constant, β:

$$Penalty_{bin} = CountOf(BinOf(IPD_{covert})) * \beta.$$

The values of α and β are used to determine the weights of the distance and bin penalties respectively. The higher the value of α with respect to β, the stronger the emphasis on small additional delays. Conversely, the higher the value of β with respect to α, the stronger the emphasis on placing the shaping IPD in bins with relatively few IPDs.

We can define a cycle in Liquid a sequence of N_t transmit IPDs followed by a sequence of N_s shaping IPDs. The values of N_t and N_s are important in determining detection resistance and channel capacity and will be discussed in a later section. N_t and N_s can be rotated among a range of possible values to provide further detection resistance. This would make it more difficult for a detection test to only sample the transmit IPDs. Since the transmit IPDs are modeled after Jitterbug, a channel that the corrected entropy test can detect (see Section 3.1), it follows that Liquid could potentially be detected if the defender could reasonably estimate which packets are used for transmission and only test those.

3.2.1 Estimation Framework

Now that we have a design for Liquid, we would like to be able to estimate its corrected entropy given different values of N_t and N_s. If we have an accurate estimate of the corrected entropy, CEN, and the percentage of unique patterns of length one, $perc$, for a Jitterbug CTC, we can estimate the entropy of a sequence of Liquid transmit IPDs by solving for H_t in Equation 3.2:

$$H_t = \frac{CEN}{1 + perc}.$$

Using H_t, we would like to derive an equation that estimates the entropy of an entire Liquid cycle, H_l. Liquid attempts to add entropy by using a sequence of shaping IPDs. We refer to this additional entropy as H_s. However, adding the shaping IPDs changes the probabilities of all the symbols generated in the transmit sequence, which consequently affects the value of H_t. We denote this new value as H_t'. The entropy of a Liquid cycle can written as the sum of the shaping and transmit entropies:

$$H_l = H_s + H_t'. \tag{3.4}$$

If we assume that Liquid will always be able to find an empty bin to place each of the shaping IPDs, we can then directly calculate H_s and H_t' to find an upper bound on the amount of entropy we can add to the channel. This upper bound is a reasonable estimate given that the corrected entropy test uses a large number of bins and a relatively small sample size. The default implementation of the corrected entropy test uses 2^{16} bins, but a sample size of only 2000 IPDs. We test the accuracy of this estimate in chapter 5.

Since each shaping IPD is placed in an empty bin, we can say that each will result in a new symbol i with the probability:

$$P_i = \frac{1}{N_t + N_s}. \tag{3.5}$$

Substituting the equation for the symbol probabilities (3.5), into the equation for entropy (3.1), H_s can be calculated as follows:

$$
\begin{aligned}
H_s &= -\sum_{i=1}^{N_s} P_i log_2 P_i \\
&= -\sum_{i=1}^{N_s} \left[\frac{1}{N_t + N_s} log_2 \left(\frac{1}{N_t + N_s} \right) \right] \\
&= -N_s * \frac{1}{N_t + N_s} log_2 \left(\frac{1}{N_t + N_s} \right) \\
&= -\frac{N_s}{N_t + N_s} log_2 \left(\frac{1}{N_t + N_s} \right) \\
&= -\frac{N_s}{N_t + N_s} [log_2(1) - log_2(N_t + N_s)] \\
&= \frac{N_s}{N_t + N_s} log_2(N_t + N_s).
\end{aligned}
$$

Due to the assumption that none of the shaping IPDs will be placed bins already containing transmit IPDs, the probability of each symbol in the transmit sequence will be reduced by a factor of x, where:

$$x = \frac{N_t}{N_t + N_s}.$$

We can use x along with Equation 3.1 to calculate H'_t:

$$
\begin{aligned}
H'_t &= -\sum x P_i log_2(x P_i) \\
&= -x \sum [P_i log_2 x + P_i log_2(P_i)] \\
&= -x \left(\sum P_i log_2(x) + \sum P_i log_2(P_i) \right) \\
&= -x \left(log_2(x) \sum P_i + (-H_t) \right) \\
&= -x log_2(x) + x H_t \\
&= -\frac{N_t}{N_t + N_s} log_2 \left(\frac{N_t}{N_t + N_s} \right) + \frac{N_t}{N_t + N_s} H_t.
\end{aligned}
$$

$$(3.6)$$

We can substitute the equations for H_s and H'_t into Equation 3.4 to calculate the entropy of the Liquid cycle:

$$
\begin{aligned}
H_l &= H_s + H'_t \\
&= \frac{N_s}{N_t + N_s} log_2(N_t + N_s) - \frac{N_t}{N_t + N_s} log_2 \frac{N_t}{N_t + N_s} + \frac{N_t}{N_t + N_s} H_t \\
&= \frac{N_s}{N_t + N_s} log_2(N_t + N_s) - \frac{N_t}{N_t + N_s} [log_2(N_t) - log_2(N_t + N_s)] + \frac{N_t}{N_t + N_s} H_t \\
&= \frac{N_s}{N_t + N_s} log_2(N_t + N_s) + \frac{N_t}{N_t + N_s} log_2(N_t + N_s) - \frac{N_t}{N_t + N_s} log_2(N_t) + \frac{N_t}{N_t + N_s} H_t \\
&= log_2(N_t + N_s) + \frac{N_t}{N_t + N_s} (H_t - log_2(N_t)).
\end{aligned}
$$

$$(3.7)$$

Even though we have Equation 3.7 to describe H_l, in order to calculate the corrected entropy we still need to calculate the value of $perc$ for the Liquid cycle. This is done as follows:

$$
perc_l = \frac{perc_t * N_t + N_s}{N_t + N_s}.
$$

Finally, using Equation 3.2 with H_l and $perc_l$ we can calculate the corrected entropy of the Liquid cycle as:

$$
CEN_l = H_l + perc_l * H_l.
$$

CHAPTER 4

EXPERIMENTAL SETUP

In this chapter, we detail the experimental setup we used to validate the detection resistance of Liquid. First we describe how we selected the inter-packet delays used for creating the Jitterbug, Liquid, and Legitimate test samples. We then cover the method we used to create network streams using those samples. Finally, we describe how we performed detection.

4.1 Data Selection

All of the IPDs used in our experiments were collected using network traces acquired from the University of North Carolina at Chapel Hill. We chose to use SSH IPDs since SSH was the protocol used by Shah et al. for Jitterbug and Gianvecchio et al. for entropy-based detection [6, 1]. In order to select the SSH IPDs from the network traces, we isolated the SSH streams using the destination port number. Since Jitterbug modulates keystroke timing, we only used streams where the SSH server was the destination. The source and destination IP addresses were used to disaggregate overlapping streams and create a sequence of correlated inter-packet delays for each individual stream. We concatenated these sequences to create two disjoint sets of data. The first, referred to as the $TrainingSet$, was used as training data for the detection tests. The second, referred to as the $SampleSet$, was used to create multiple test subsets and to create the Liquid training bins. Liquid used the $SampleSet$ for creating its own training bins because, in an actual attack scenario, it is unrealistic to assume that it would have access to the training data used by the detection tests. The size of each set is shown in Table 4.1.

Table 4.1. Experimental Set Sizes. Sizes of the *TrainingSet* and the *SampleSet*.

Type	Number of IPDs
TrainingSet	7,523,526
SampleSet	2,426,264

Most of the IPDs in the sample set are smaller than 5 seconds; however, a small percentage of the IPDs are significantly larger. As such, we set the maximum value for each IPD in the test sets at 120 seconds to allow the experiments to finish in a reasonable amount of time.

4.2   Creating the Test Traffic

In order to simulate a user typing, we created a software application which would send keystrokes directly to an SSH session. The time between each successive keystroke was based a sequence of input IPDs. We refer to this sequence of input IPDs as a test sample. Each test sample was comprised of correlated IPDs taken from the *SampleSet*. To directly compare the differences between legitimate, Jitterbug, and Liquid traffic, each test sample was used as the base input sequence for all three types of traffic. In the case of legitimate traffic, the test sample was sent unchanged. For Jitterbug and Liquid, the test sample was first modified to embed a covert message. We used multiple sets of samples to explore the effect of different parameters and scenarios on the detection scores. Each test set, unless otherwise specified, was created using 100 samples consisting of 2000 IPDs each.

We set the Jitterbug and Liquid parameters to match what was used by Gianvecchio et al. for their study of entropy-based detection [1]. The timing window w was set to be 20ms and the length of the rotate sequence s was set to be equal to the sample size

of 2000 IPDs. For Liquid, we tested various values of N_s and N_t and their effect on achieving different levels of detection resistance.

We also ran Liquid test sets both with and without sub-millisecond noise injected into the transmit IPDs. Recall from Section 3.2 that the Liquid transmit IPDs are encoded exactly like Jitterbug when transmit noise is not used. Removing the transmit noise allows us to more directly examine the effect of the shaping IPDs.

We set the constant α equal to zero so that the distance penalty for shaping IPDs would be zero. Conversely, we set the constant β equal to one, so that the bin penalty for every shaping IPD would be equal to the number of IPDs in the bin. Effectively, this means that the delay added to each shaping IPD was determined solely by the bin count. We eliminated the distance penalty because each shaping IPD could only be delayed a maximum of w milliseconds — a relatively small amount.

4.3 Detection

The sender was located inside of our network and the receiver was an external SSH server. The IPDs were captured at two locations. One was at the sender and the other was four hops away at the edge of the network. The IPDs captured at the first location are only perturbed by the processes on the sending machine itself. This serves as a worst case detection scenario for the covert channel since there is no network jitter to distort the intended IPDs. The second location was chosen so as to mimic a likely detection scenario in a real-world situation. We refer to these locations as local and remote, respectively.

We ran the corrected entropy, corrected conditional entropy, and Kolmogorov-Smirnov tests on each captured sample of legitimate, Jitterbug, and Liquid IPDs. Each test was performed, offline, i.e. after a complete sample of IPDs had been captured and parsed. The resulting test scores for the legitimate IPDs were used to create a cutoff score for each of the detection tests. To achieve a one percent false positive rate, the

1st percentile was used for the corrected entropy test, while the 99th percentile was used for the corrected conditional entropy and Kolmogorov-Smirnov tests. Any sample with a CEN value below the cutoff score would be consider covert. Similarly, any sample with a CCE or KS value above the respective cutoff scores would also be considered covert. For each test, we calculated the detection rate for both Jitterbug and Liquid. This process was performed separately for each detection location.

CHAPTER 5

EXPERIMENTAL RESULTS

In this chapter, we empirically evaluate the accuracy of the estimation equations presented in Chapter 3, as well as the ability of Liquid to evade detection.

5.1 Estimating the Effect of Shaping

We first explore the accuracy of our model of detection accuracy as presented in Chapter 3, and we then use those equations to estimate the values of N_s and N_t needed to achieve certain levels of detection resistance. Liquid, as it is discussed in this section, was implemented without sub-millisecond transmit noise so that the behavior of the Liquid transmit IPDs would exactly match that of original Jitterbug. This provides us with a clearer picture of the effect that shaping IPDs have on the entropy of the channel.

5.1.1 Accuracy of the Estimation Framework

To test the accuracy of our estimation framework, we embedded Liquid into each sample of our test set such that the first 1000 IPDs were for transmission and the following 1000 IPDs were for shaping. For each sample, we calculated the entropy, percentage of singular patterns, and corrected entropy for the transmit IPDs. We then used these values, along with the model presented in Chapter 3, to estimate what the entropy, percentage of singular patterns, and corrected entropy would be for the complete Liquid cycle. We denote the average difference between the estimated and actual values of the corrected entropy for the Liquid cycle (CEN_l) as ϵ. Furthermore, we denote the percentage of singular bins for shaping IPDs as $perc_s$.

Table 5.1. Accuracy of the estimation framework for samples of 2000 Liquid IPDs. The values are averages of the entire test set. This table shows both the corrected entropy score for an entire Liquid cycle as well as the percentage of singular bins for the shaping IPDs.

	Est	Act	ϵ
CEN_l	14.63	14.11	0.52
$perc_s$	1.00	0.91	na

Table 5.1 shows the accuracy of the estimation framework for samples of 2000 Liquid IPDs. The presented value for CEN_l is the average corrected entropy score for each sample in the test set. Similarly, $perc_s$ is also an average. We can see that the actual corrected entropy is 0.52 bits lower than the estimation. This difference can be attributed to the assumption that each shaping IPD will be put into an empty bin, i.e. $perc_s = 1$, whereas the average value of $perc_s$ is actually 0.91. As mentioned in Chapter 3, these equations provide an upper bound on the value of CEN.

5.1.2   Estimating the Number of Shaping IPDs

We used the equations to estimate the values of N_s and N_t needed for Liquid to evade CEN detection 99, 75, and 50 percent of the time. We assumed that detection would be performed using samples of 2000 IPDs. We first ran a test set of legitimate IPDs and found the detection cutoff score, CEN_{cutoff}. We then ran a test set of Liquid transmit IPDs and recorded the entropy H_t and the percentage of singular bins $perc_t$ for each sample. Table 5.2 lists the 1st, 25th, and 50th percentile scores for H_t and $perc_t$.

Table 5.2. Percentile scores for Liquid transmit IPDs. The average entropy and percentage of singular bins for the test set.

Percentile	H_t	$perc_t$
1	4.68	0.00
25	6.42	0.05
50	7.09	0.08

Table 5.3. Accuracy of the detection resistance estimations. Estimated values of N_s and N_t in a 2000 IPD sample needed to obtain a target evasion percentage. Each row contains the expected and actual evasion percentages for each set of parameters.

Exp %	Act %	N_t	N_s
99	98	900	1100
75	75	1074	926
50	62	1168	832

The 1st, 25th, and 50th percentile scores were substituted into the following system of equations, which was used to solve for N_t and N_s:

$$
\begin{aligned}
N_t + N_s &= 2000 \\
H_l &= log_2(N_t + N_s) + \frac{N_t}{N_t + N_s}(H_t - log_2(N_t)) \\
perc_l &= \frac{perc_t * N_t + N_s}{N_t + N_s} \\
CEN_{cutoff} + \epsilon &= H_l + perc_l * H_l
\end{aligned}
$$

$$(5.1)$$

We added ϵ to the corrected entropy cutoff score to account for any error in our estimation equations. The estimated values for N_t and N_s are listed in Table 5.3.

To test the accuracy of our estimations, we ran the corrected entropy test on 150 samples of Liquid and calculated the actual detection rates. Table 5.3 shows the accuracy of the estimation for the values of N_s and N_t in a 2000 IPD sample. Each row contains the expected and actual evasion percentages for each set of parameters. From the table

we can see that Liquid was able meet or exceed each target detection rate using our estimates.

5.2   Detection Resistance

In this section, we explore the detection resistance of Liquid against the corrected entropy, corrected conditional entropy, and Kolmogorov-Smirnov tests.

For the Liquid samples, we rotated the values of N_t and N_s in the range [1..3] such that $\frac{N_t}{N_s} \approx 1$. In this way, the Liquid cycle would consist of as few as 2, or as many as 6 IPDS, with the number changing over time. We rotated between small values of N_t and N_s so that a detection scheme would be unable to easily distinguish between the IPDs used for transmission and those used for shaping. Otherwise, such a scheme could ignore all IPDs used for shaping and completely negate their effect. In order to further increase detection resistance of Liquid, we implemented transmit noise.

We ran 100 samples of legitimate, Jitterbug, and Liquid traffic, capturing the IPDs at the local and remote detection locations discussed in Chapter 4. The corrected entropy, corrected conditional entropy, and Kolmogorov-Smirnov tests were performed on each sample. Using the test results for the legitimate samples, we calculated the cutoff scores for each test so as to achieve a false positive rate of 0.01 or 0.10. We used a 0.10 false positive rate for the corrected entropy test at the remote site so that we could achieve a decent detection rate for Jitterbug.

Cutoff scores for the corrected entropy, corrected conditional entropy, and Kolmogorov-Smirnov tests (calculated for both the local and remote detection sites) are shown in Table 5.4. Any sample with a corrected entropy value less than or equal to the CEN cutoff score would be considered covert. Conversely, any sample with a corrected conditional entropy or Kolmogorov-Smirnov test score greater than or equal to the CCE and KS cutoff scores, respectively, would be considered covert.

Table 5.4. Cutoff scores for the CEN, CCE, and KS tests. Scores are calculated for both the local and remote detection sites. Any sample with a score above or below the respective cutoff score would be considered covert. Each test's cutoff score was chosen to achieve a target false positive rate.

Test	Score		False Positive
	Local		
CEN	$\leq$	11.74	0.01
CCE	$\geq$	1.71	0.01
KS	$\geq$	0.65	0.01
	Remote		
CEN	$\leq$	15.56	0.10
CCE	$\geq$	1.77	0.01
KS	$\geq$	0.64	0.01

The test scores and detection rates for the Legitimate, Jitterbug and Liquid samples are shown in Tables 5.5, 5.6, 5.7. We can see that the KS Test is unable to detect either Jitterbug or Liquid. This is because the amount of additional delay added by both channels is relatively small when compared to the average SSH IPD [1]. Furthermore, CCE detection also fails to identify either of the covert channels. There is a slight difference between the average CCE for the legitimate samples and the average CCE scores for Liquid. This difference is not detectable without significantly increasing the false positive rate, rendering the test useless. On the other hand, the corrected entropy test is able to consistently detect the presence of Jitterbug. It is interesting to note the substantial difference between the average CEN value of the Jitterbug samples at the two detection locations. The average CEN for the remote site is 5.37 bits higher than that of the local site. We attributed this increase to network jitter. In order to achieve a reasonable detection rate for Jitterbug at the remote site, we had to increase the false positive rate to 0.10. Both the legitimate and Liquid samples also showed a modest increase in the average CEN scores.

Table 5.5. Legitimate test scores. Average test scores and false positive rates for the legitimate samples at both the local and remote sites.

Test	Mean	Standard Deviation	False Positive
Local			
CEN	18.24	2.19	0.01
CCE	1.06	0.38	0.01
KS	0.37	0.09	0.01
Remote			
CEN	18.25	2.20	0.10
CCE	1.05	0.36	0.01
KS	0.37	0.08	0.01

Table 5.6. Jitterbug test scores. Average test scores and detection rates for the Jitterbug samples at both the local and remote sites.

| Test	Mean	Standard Deviation	Detection Rate
Local			
CEN	8.55	1.18	1.00
CCE	1.19	0.27	0.02
KS	0.39	0.07	0.01
Remote			
CEN	13.92	1.34	0.91
CCE	1.16	0.28	0.00
KS	0.39	0.07	0.01

Table 5.7. Liquid test scores. Average test scores and detection rates for the Liquid samples at both the local and remote sites.

| Test	Mean	Standard Deviation	Detection Rate
Local			
CEN	19.18	1.61	0.00
CCE	1.26	0.29	0.02
KS	0.35	0.08	0.01
Remote			
CEN	20.05	1.03	0.02
CCE	1.22	0.29	0.02
KS	0.36	0.08	0.01

Using this data, we see that none of the detection tests were able to effectively distinguish Liquid from legitimate traffic. Since we set N_t and N_s such that approximately half of the IPDs in each sample were used for shaping, we can say that our channel achieved complete detection resistance while only reducing the maximum channel capacity by half.

5.3 Bin Frequency

As we detailed in Chapter 3, the entropy of a stream is maximized when all of the possible symbols are equally likely. The corrected entropy test uses a histogram of IPDs with bins corresponding to the possible symbols. The bin ranges are constructed in such a way that, for legitimate samples, each symbol would be equally likely, maximizing the entropy. Figure 5.3 shows the count of each bin for 200,000 IPDs of different types. As we can see from Figure 5.1(a), the count of the bins for the legitimate IPDs is basically uniform. If we look at Figure 5.1(b) we can see that the Jitterbug IPDs heavily cluster around a small set of bins. This clustering is easily detected by the corrected entropy test. Interestingly, if we look at the results for Liquid in Figure 5.1(c), the histogram appears to be roughly uniform, with a few high count bins. We can attribute these minor bin discrepancies to the transmit IPDs of Liquid. However, they are much less pronounced than we see in Jitterbug, and consequently undetectable for smaller samples.
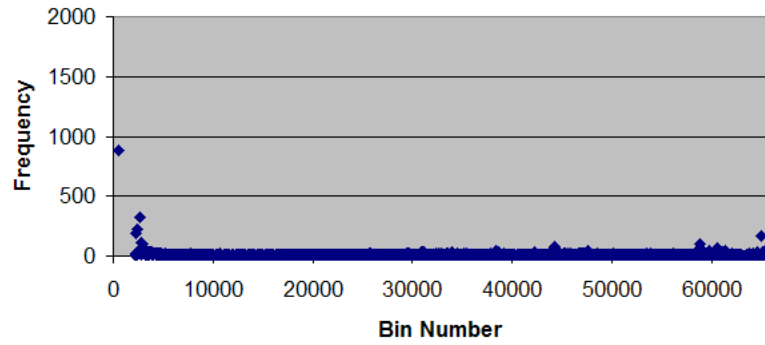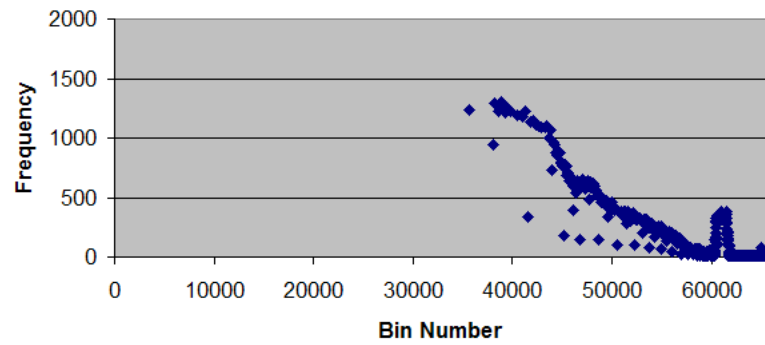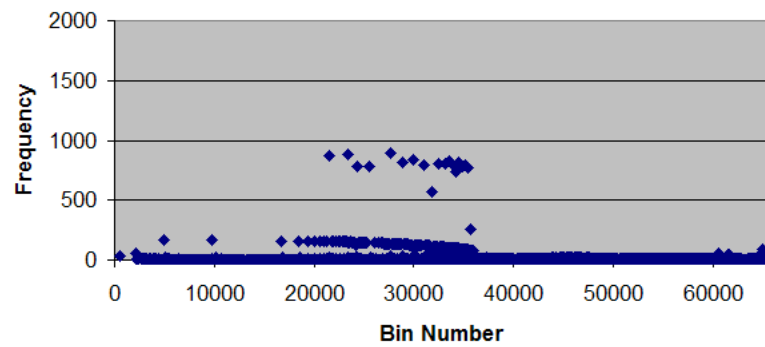
Figure 5.1. Histograms of the training bins used by the corrected entropy test for different sets of traffic. (a) Legitimate. (b) Jitterbug. (c) Liquid.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We proposed Liquid, a new covert timing channel that uses a portion of the inter-packet delays in a compromised stream to smooth out the symbol probability distortions detected by the entropy test. We split the implementation of our channel into two main parts. The first part consisted of the techniques used for embedding the hidden message, which we based on Jitterbug. We chose Jitterbug because of its unique attack vector and its ability to evade all but the corrected entropy test. We also explored the effect of adding small amounts of noise to help disguise the regularity of these transmit IPDs. The second part was comprised of the methods used for shaping the channel such that it would evade the corrected entropy test. We introduced the idea of adding additional shaping delay to a subset of IPDs in the channel. The additional delay for each shaping IPD would be chosen so as to minimize a cost function.

We found that the number of inter-packet delays used for shaping versus the number used for transmitting the message is an important consideration when trying to balance detection resistance and capacity. To better understand this trade-off, we introduced a set of equations to estimate the effects of our techniques on the entropy of the channel. We then used those equations to estimate the channel parameters needed to obtain certain levels of detection resistance. We empirically showed that when Liquid used the estimated parameters, it was able to meet or exceed the target levels of detection resistance.

We then evaluated the detection rates of various tests against both Liquid and Jitterbug. We showed that none of the tests were able to distinguish Liquid from legitimate samples.

Future work could focus on detecting Liquid, potentially by modifying the corrected conditional entropy test. Another potential avenue of exploration includes the development of techniques for dynamic adjustment of the shaping bins. Furthermore, we could analyze methods for maintaining synchronization between the covert sender and receiver when the values of N_s and N_t are rotated. Finally, we plan on extending Liquid to other network protocols.

REFERENCES

[1] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 307–316.

[2] R. A. Kemmerer, "Shared resource matrix methodology: an approach to identifying storage and timing channels," *ACM Trans. Comput. Syst.*, vol. 1, no. 3, pp. 256–277, 1983.

[3] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in internet traffic with active wardens," in *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*. London, UK: Springer-Verlag, 2003, pp. 18–35.

[4] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert channel detection," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 4, pp. 1–29, 2009.

[5] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia, "Model-based covert timing channels: Automated modeling and evasion," in *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 211–230.

[6] G. Shah, A. Molina, and M. Blaze, "Keyboards and covert channels," in *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2006.

[7] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh," in *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2001, pp. 25–25.

[8] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 116–130.

[9] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2003, pp. 20–29.

[10] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows," in *Network and Distributed Systems Security Symposium*, G. Vigna, Ed. Internet Society, February 2009.

[11] R. Kemmerer, "A practical approach to identifying storage and timing channels: twenty years later," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, 2002, pp. 109–118.

[12] W.-M. Hu, "Reducing timing channels with fuzzy time," in *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, May 1991, pp. 8–20.

[13] M. H. Kang and I. S. Moskowitz, "A pump for rapid, reliable, secure communication," in *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*. New York, NY, USA: ACM, 1993, pp. 119–129.

[14] M. H. Kang, I. S. Moskowitz, and S. Chincheck, "The pump: A decade of covert fun," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 352–360.

[15] V. Berk, A. Giani, and G. Cybenko, "Covert channel detection using process query systems," in *Proceedings of FLOCON 2005*, September 2005.

[16] ——, "Detection of covert channel encoding in network packet delays," Dartmouth College, Tech. Rep. TR536.

[17] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: design and detection," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 2004, pp. 178–187.

[18] R. Stillman, "Detecting ip covert timing channels by correlating packet timing with memory content," in *Southeastcon, 2008. IEEE*, April 2008, pp. 204–209.

[19] C. E. Shannon, "A mathematical theory of communication," in *Bell System Technical Journal*, vol. 27, July and October 1948.

BIOGRAPHICAL STATEMENT

Robert J. Walls was born in Dallas, Texas. He received his B.S. and M.S. degrees in Computer Science from The University of Texas at Arlington, in 2007 and 2009 respectively. From 2006 to 2009 he worked with the software development group at Tenaska Power Services.