A First Step Toward Live Botmaster Traceback

Daniel Ramsbrock, Xinyuan Wang
Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
{dramsbro, xwangc}@gmu.edu

Xuxian Jiang
Department of Computer Science
North Carolina State University
Raleigh, NC 27606, USA
jiang@cs.ncsu.edu

Abstract. Despite the increasing botnet threat, research in the area of botmaster traceback is limited. The four main obstacles are 1) the low-traffic nature of the bot-to-botmaster link; 2) chains of "stepping stones;" 3) the use of encryption along these chains; and 4) mixing with traffic from other bots. Most existing traceback approaches can address one or two of these issues, but no single approach can overcome all of them. We present a novel flow watermarking technique to address all four obstacles simultaneously. Our approach allows us to uniquely identify and trace any IRC-based botnet flow even if 1) it is encrypted (e.g., via SSL/TLS); 2) it passes multiple intermediate stepping stones (e.g., IRC server, SOCKs); and 3) it is mixed with other botnet traffic. Our watermarking scheme relies on adding padding characters to outgoing botnet C&C messages at the application layer. This produces specific differences in lengths between randomly chosen pairs of messages in a network flow. As a result, our watermarking technique can be used to trace any interactive botnet C&C traffic and it only requires a few dozen packets to be effective. To the best of our knowledge, this is the first approach that has the potential to allow real-time botmaster traceback across the Internet.
We have empirically validated the effectiveness of our botnet flow watermarking approach with live experiments on PlanetLab nodes and public IRC servers on different continents. We achieved virtually a 100% detection rate of watermarked (encrypted and unencrypted) IRC traffic with a false positive rate on the order of 10^{-5}. Due to the message queuing and throttling functionality of IRC servers, mixing chaff with the watermarked flow does not significantly impact the effectiveness of our watermarking approach.

1 Introduction

Botnets are currently one of the most serious threats to computers connected to the Internet. Recent media coverage has revealed many large-scale botnets worldwide. One botnet [22, 23] has reportedly compromised and controlled over 400,000 computers – including computers at the Weapons Division of the U.S. Naval Air Warfare Center, U.S. Department of Defense Information Systems Agency. Another recently discovered botnet is suspected to have controlled 1.5 million computers around the globe [9]. It has been estimated [20] that more than

5 percent of all computers connected to the Internet have been compromised and used as bots. Currently, botnets are responsible for most spam, adware, spyware, phishing, identity theft, online fraud and DDoS attacks on the Internet.

The botnet problem has recently received significant attention from the research community. Most existing work on botnet defense [1–3, 6, 11, 14, 15, 18] has focused on the detection and removal of command and control (C&C) servers and individual bots. While such a capability is a useful start in mitigating the botnet problem, it does not address the root cause: the botmaster. For example, existing botnet defense mechanisms can detect and dismantle botnets, but they usually cannot determine the identity and location of the botmaster. As a result, the botmaster is free to create and operate another botnet by compromising other vulnerable hosts. Botmasters can currently operate with impunity due to a lack of reliable traceback mechanisms. However, if the botmaster's risk of being caught is increased, he would be hesitant to create and operate botnets. Therefore, even an imperfect botmaster traceback capability could effectively deter botmasters. Unfortunately, current botmasters have all the potential gains from operating botnets with minimum risk of being caught. Therefore, the botnet problem cannot be solved until we develop a reliable method for identifying and locating botmasters across the Internet. This paper presents a substantial first step towards achieving the goal of botmaster traceback.

Tracking and locating the botmaster of a discovered botnet is very challenging. First, the botmaster only needs to be online briefly to issue commands or check the bots' status. As a result, any botmaster traceback has to occur in real-time. Second, the botmaster usually does not directly connect to the botnet C&C server and he can easily launder his connection through various stepping stones. Third, the botmaster can protect his C&C traffic with strong encryption. For example, Agobot has built-in SSL/TLS support. Finally, the C&C traffic from the botmaster is typically low-volume. As a result, a successful botmaster traceback approach must be effective on low-volume, encrypted traffic across multiple stepping stones.

To the best of our knowledge, no existing traceback methods can effectively track a botmaster across the Internet in real-time. For example, methods [33, 32, 8, 31, 4, 29, 30] have been shown to be able to trace encrypted traffic across various stepping stones and proxies, but they need a large amount of traffic (at least hundreds of packets) to be effective. During a typical session, each bot exchanges only a few dozen packets with the botmaster. Due to this low traffic volume, the above techniques are not suitable for botmaster traceback.

In this paper, we address the botmaster traceback problem with a novel packet flow watermarking technique. Our goal is to develop a practical solution that can be used to trace low-volume botnet C&C traffic in real-time even if it is encrypted and laundered through multiple intermediate hosts (e.g., IRC servers, stepping stones, proxies). We assume that the tracer has control of a single rogue bot in the target botnet, and this bot can send messages in response to a the query from the botmaster. To trace the response traffic back to the botmaster, the rogue bot transparently injects a unique watermark into its

response. If the injected watermark can survive the various transformations (e.g., encryption/decryption, proxying) of the botnet C&C traffic, we can trace the watermark and locate the botmaster via monitoring nodes across the Internet. To embed the watermark, we adjust the lengths of randomly selected pairs of packets such that the length difference between each packet pair will fall within a certain range. To track encrypted botnet traffic that mixes messages from multiple bots, we developed a hybrid length-timing watermarking method. Compared to previous approaches [31, 29, 30], our two proposed methods require far less traffic volume to encode high-entropy watermarks. We empirically validated the effectiveness of our watermarking algorithms using real-time experiments on live IRC traffic through PlanetLab nodes and public IRC servers across different continents. Both of our watermarking approaches achieved a virtually 100% watermark detection rate and a 10^{-5} false positive rate with only a few dozen packets. To the best of our knowledge, this is the first approach that has the potential to allow real-time botmaster traceback across the Internet.

The remainder of the paper is structured as follows: Section 2 introduces the botmaster traceback model. Section 3 presents the design and analysis of our flow watermarking schemes. Section 4 describes our experiments and their results, while section 5 discusses limitations and future work. Finally, Section 6 surveys related literature and Section 7 summarizes our findings.

2 Botmaster Traceback Model

According to [17, 21, 28], most botnets currently in the wild are IRC-based. Therefore, we will focus on tracing the botmaster in the context of IRC-based botnets. Nevertheless, our flow watermarking trace approach is applicable to any interactive botnet traffic.

2.1 Botnets and Stepping Stones

Bots have been covered extensively in the existing literature, for example [2, 6, 7, 16, 21] provide good overviews. The typical bot lifecycle starts with exploitation, followed by download and installation of the bot software. At this point, the bot contacts the central C&C server run by the botmaster, where he can execute commands and receive responses from his botnet.

Botmasters rarely connect directly to their C&C servers since this would reveal their true IP address and approximate location. Instead, they use a chain of stepping stone proxies that anonymously relay traffic. Popular proxy software used for this purpose is SSH, SOCKS, and IRC BNCs (such as psyBNC). Since the stepping stones are controlled by the attacker, they do not have an audit trail in place or other means of tracing the true source of traffic. However, there are two properties of stepping stones that can be exploited for tracing purposes: 1) the content of the message (the application-layer payload) is never modified and 2) messages are passed on immediately due to the interactive nature of IRC. Consequently, the relative lengths of messages and their timings are preserved,

even if encryption is used. In the case of encryption, the message lengths are rounded up to the nearest multiple of the block size. This inherent length and timing preservation is the foundation of our botmaster traceback approach.

2.2 Tracking the Botmaster by Watermarking Botnet Traffic

Our botmaster traceback approach exploits the fact that the communication between the IRC-based bots and the botmaster is bidirectional and interactive. Whenever the botmaster issues commands to a bot, the response traffic will eventually return to the botmaster after being laundered and possibly transformed. Therefore, if we can watermark the response traffic from a bot to the botmaster, we can eventually trace and locate the botmaster. Since the response traffic we are tracking may be mixed with other IRC traffic, we need to be able to isolate the target traffic. With unencrypted traffic, this can be achieved by content inspection, but encrypted traffic presents a challenge which we address with our hybrid length-timing algorithm.

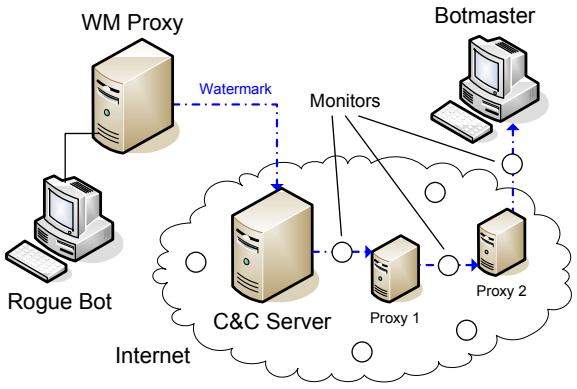

Fig. 1. Botmaster traceback by watermarking the botnet response traffic.

Figure 1 shows the overall watermarking traceback model. We assume that we control a rogue bot, which could be a honeypot host that has been compromised and has joined a botnet. The rogue bot watermarks its outgoing PRIVMSG traffic in response to commands from the botmaster. As with any traceback approach, our watermark tracing scheme needs support from the network. Specifically, we assume there are cooperating monitor nodes across the Internet, which will inspect the passing traffic for the specified watermark and report back to us whenever they find it. Note that our approach does not require a global monitoring capability. If there are uncooperative or unmonitored areas, we would lose one or more links along the traceback path. However, we can pick up the trail again once the watermarked traffic re-enters a monitored area. In general, this appears to be the best possible approach in the absence of a global monitoring

capability. We assume that the tracer can securely share the desired watermark with all monitor nodes prior to sending the watermarked traffic. This enables the monitors to report 'sightings' of the watermark in real-time and requires only a single watermarked flow to complete the trace.

3 Length-Based Watermarking Scheme

Our watermarking scheme was specifically designed for a low-traffic, text-based channel such as the one between a bot and its botmaster. This section describes the design and analysis of both the length-only (unencrypted traffic) and the length-timing hybrid algorithms (encrypted traffic). We describe the encoding and decoding formulas for both algorithms and address the issue of false positives and false negatives.

The terms 'message' and 'packet' are used interchangeably since a typical botnet C&C message is usually small (less than 512 bytes) and fits into a single packet).

3.1 Basic Length-Based Watermarking Scheme

Watermark Bit Encoding Given a packet flow f of n packets $P_1, \ldots, P_n$, we want to encode an l-bit watermark $W = w_0, \ldots, w_{l-1}$ using $2l \leq n$ packets. We first use a pseudo-random number generator (PRNG) with seed s to randomly choose $2l$ distinct packets from $P_1, \ldots, P_n$, we then use them to randomly form l packet pairs: $\langle P_{r_i}, P_{e_i} \rangle$ $(i = 0, \ldots, l-1)$ such that $r_i \leq e_i$. We call packet P_{r_i} a *reference packet* and packet P_{e_i} an *encoding packet*. We further use the PRNG to randomly assign watermark bit w_k $(0 \leq k \leq l-1)$ to packet pair $\langle P_{r_i}, P_{e_i} \rangle$, and we use $\langle r_i, e_i, k \rangle$ to represent that packet pair $\langle P_{r_i}, P_{e_i} \rangle$ is assigned to encode watermark bit w_k.

To encode the watermark bit w_k into packet pair $\langle P_{r_i}, P_{e_i} \rangle$, we modify the length of the encoding packet P_{e_i} by adding padding characters to achieve a specific length difference to its corresponding reference packet P_{r_i}. The padding characters could be invisible (such as whitespace) or visible characters and they can be inserted in random locations within the message. This would make it difficult for the adversary to detect the existence of the padding. Let l_e and l_r be the packet lengths of the watermark encoding and reference packets respectively, $Z = l_e - l_r$ be the length difference, and $L > 0$ be the bucket size. We define the *watermark bit encoding function* as

$$e(l_r, l_e, L, w) = l_e + [(0.5 + w)L - (l_e - l_r)] \bmod 2L \tag{1}$$

which returns the increased length of watermark encoding packet given the length of the reference packet l_r, the length of the encoding packet l_e, the bucket size L, and the watermark bit to be encoded w.

Therefore,

$$
\begin{aligned}
& (e(l_r, l_e, L, w) - l_r) \bmod 2L \qquad (2)\\
&= \{(l_e - l_r) + [(0.5 + w)L - (l_e - l_r)] \bmod 2L\} \bmod 2L\\
&= \{(0.5 + w)L\} \bmod 2L\\
&= (w + 0.5)L
\end{aligned}
$$

This indicates that the packet length difference $Z = l_e - l_r$, after l_e is adjusted by the watermark bit encoding function $e(l_r, l_e, L, w)$, falls within the middle of either an even or odd numbered bucket depending on whether the watermark bit w is even or odd.

Watermark Bit Decoding Assuming the decoder knows the watermarking parameters: PRNG, s, n, l, W and L, the watermark decoder can obtain the exact pseudo-random mapping $\langle r_i, e_i, k\rangle$ as that used by the watermark encoder. We use the following *watermark bit decoding function* to decode watermark bit w_k from the packet lengths of packets P_{r_i} and P_{e_i}

$$
d(l_r, l_e, L) = \lfloor \frac{l_e - l_r}{L} \rfloor \bmod 2 \qquad (3)
$$

The equation below proves that any watermark bit w encoded by the encoding function defined in equation (1) will be correctly decoded by the decoding function defined in equation (3).

$$
\begin{aligned}
& d(l_r, e(l_r, l_e, L, w), L) \qquad (4)\\
&= \lfloor \frac{e(l_r, l_e, L, w) - l_r}{L} \rfloor \bmod 2\\
&= \lfloor \frac{(l_e - l_r) \bmod 2L + [(0.5 + w)L - (l_e - l_r)] \bmod 2L}{L} \rfloor \bmod 2\\
&= \lfloor \frac{(0.5 + w)L}{L} \rfloor \bmod 2\\
&= w
\end{aligned}
$$

Assume the lengths of packets P_r and P_e (l_r and l_e) have been increased for $x_r \geq 0$ and $x_e \geq 0$ bytes respectively when they are transmitted over the network (e.g., due to padding of encryption), then $x_e - x_r$ is the distortion over the packet length difference $l_e - l_r$. Then the decoding with such distortion is

$$
\begin{aligned}
& d(l_r + x_r, e(l_r, l_e, L, w) + x_e, L) \qquad (5)\\
&= \lfloor \frac{e(l_r, l_e, L, w) - l_r + (x_e - x_r)}{L} \rfloor \bmod 2\\
&= w + \lfloor 0.5 + \frac{x_e - x_r}{L} \rfloor \bmod 2
\end{aligned}
$$

Therefore, the decoding with distortion will be correct if and only if

$$(-0.5 + 2i)L \leq x_e - x_r < (0.5 + 2i)L \tag{6}$$

Specifically, when the magnitude of the distortion $|x_e - x_r| < 0.5L$, the decoding is guaranteed to be correct.

Watermark Decoding and Error Tolerance Given a packet flow f and appropriate watermarking parameters (PRNG, s, n, l, W and L) used by the watermark encoder, the watermark decoder can obtain a l-bit decoded watermark W' using the watermark bit decoding function defined in equation (3). Due to potential distortion of the packet lengths in the packet flow f, the decoded W' could have a few bits different from the encoded watermark W. We introduce a Hamming distance threshold $h \geq 0$ to accommodate such partial corruption of the embedded watermark. Specifically, we will consider that packet flow f contains watermark W if the Hamming distance between W and W': $H(W, W')$ is no bigger than h.

Watermark Collision Probability (False Positive Rate) No matter what watermark W and Hamming distance threshold h we choose, there is always a non-zero possibility that the decoding W' of a random unwatermarked flow happens to have no more than h Hamming distance to the random watermark W we have chosen. In other words, watermark W is reported found in an unwatermarked flow; we refer to this case as a *watermark collision*.

Intuitively, the longer the watermark and the smaller the Hamming distance threshold, the smaller the probability of a watermark collision. Assume we have randomly chosen a l-bit watermark, and we are decoding l-bits from random unwatermarked flows. Any particular bit decoded from a random unwatermarked flow should have 0.5 probability to match the corresponding bit of the random watermark we have chosen. Therefore, the collision probability of l-bit watermark from random unwatermarked flows with Hamming distance threshold h is

$$\sum_{i=0}^{h} \binom{l}{i} (\frac{1}{2})^l \tag{7}$$

We have empirically validated the watermark collision probability distribution with the following experiment. We first use a PRNG and a random seed number s to generate 32 packet pairs $\langle r_i, e_i \rangle$ and pseudo-randomly assign each bit of a 32-bit watermark W to the 32 packet pairs, we then encode the 32 bit watermark W into a random packet flow f. Now we try to decode the watermarked flow f' with 1,000 wrong seed numbers. Given the pseudo-random nature of our selection of the packet pairs, decoding a watermarked flow with the wrong seed is equivalent of decoding an unwatermarked flow, which can be used to measure the watermark collision probability.

The left side of Figure 2 illustrates the number of matched bits from the decoding with each of the 1,000 wrong seed numbers. It shows that the numbers

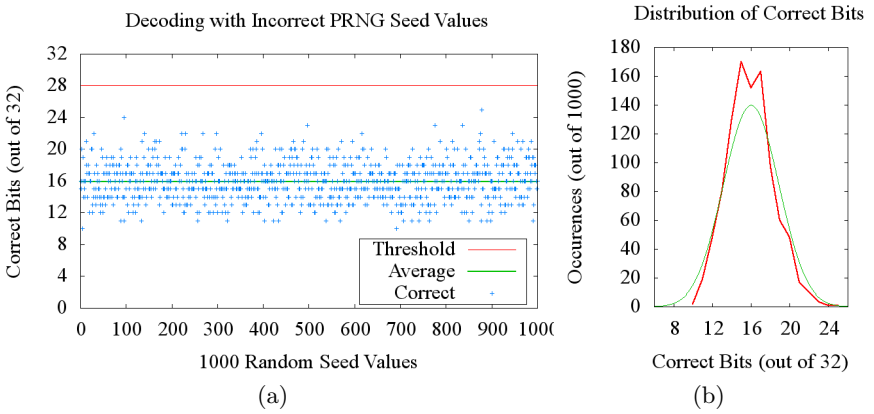

Fig. 2. 32-bit watermark collision probability and distribution

of matched bits are centered around the expected value of 16 bits, which is half of the watermark length. Based on these results and the experimental data in Section 4.2, we can choose a Hamming distance threshold of $h = 4$ (28 bits) as shown on the graph, yielding an expected false positive rate (FPR) of 9.64×10^{-6} according to equation (7). The right side of Figure 2 shows the distributions of the measured and the expected number of matched bits. It illustrates that the distribution of the measured number of matched bits is close to the expected binomial distribution with $p = 0.5$ and $n = 32$.

Watermark Loss (False Negative) Our length-only encoding scheme (without the hybrid timing approach) is highly sensitive to having the correct sequence of messages. If any messages are added or deleted in transit, the watermark will be lost in that flow. However, the chance of this happening is very remote since the encoding takes place at the application layer, on top of TCP. By its nature, TCP guarantees in-order delivery of all packets and their contents, so a non-intentional watermark loss is very unlikely.

In the case of active countermeasures, our scheme can tolerate distortion as long as $|x_e - x_r| < 0.5L$, as described by inequality (6). This property is the result of aiming for the center of each bucket when encoding. However, if an active adversary drops, adds, or reorders messages, the watermark will be lost unless additional redundancy is in place or the length-timing algorithm is used.

3.2 Hybrid Length-Timing Watermarking for Encrypted Traffic

By their nature, IRC-based botnets have many bots on one channel at once, many of them joining, parting, or sending data to the botmaster simultaneously. In this case, the watermarked messages from our rogue bot will be mixed with

unwatermarked messages from other bots. We call these unwatermarked messages from others *chaff* messages. In order to reliably decode the embedded watermark, we need to filter out chaff messages as much as possible.

When the C&C traffic is unencrypted, it is easy for the watermark decoder to filter out chaff based on the sender nicks in the messages. However, if the traffic is encrypted (e.g., using SSL/TLS), we cannot rely on content inspection to identify chaff messages. To address this new challenge in filtering out chaff, we propose to use another dimension of information – the packet timing – to filter out chaff.

The basic idea is to send the watermark encoding packets at a specific time (e.g., t_i). Assuming the network jitter δ is limited, we can narrow the range of potential packets used for decoding to $[t_{e_i} - \frac{\delta}{2}, t_{e_i} + \frac{\delta}{2}]$. If $\delta > 0$ is small, then the chances that some chaff packet happens to fall within the range $[t_{e_i} - \frac{\delta}{2}, t_{e_i} + \frac{\delta}{2}]$ is small. This means we can decode the watermark correctly even if there are substantial encrypted chaff packets.

Watermark Encoding The watermark bit encoding process is exactly the same as that of the basic length-based watermarking scheme. The difference is that now we send out each watermarked packet P_{e_i} at a precise time. Specifically, we use the watermark bit encoding function defined in equation (1) to adjust the length of the watermark encoding packet P_{e_i}. We use a pseudo-random number generator PRNG and seed s_t to generate the random time t_{e_i} at which P_{e_i} will be sent out.

An implicit requirement for the hybrid length-timing watermarking scheme is that we need to know when each watermark encoding packet P_{e_i} will be available. In our watermark tracing model, the tracer owns a rogue bot who can determine what to send out and when to send it. Since we have full control over the outgoing traffic, we can use the hybrid length-timing scheme to watermark the traffic in real-time.

Watermark Decoding When we decode the encrypted botnet traffic, we do not know which packet is a watermark encoding packet P_{e_i}. However, given the PRNG and s_t we do know the approximate time t_{e_i} at which the watermark encoding packet P_{e_i} should arrive. We then use all packets in the time interval $[t_{e_i} - \frac{\delta}{2}, t_{e_i} + \frac{\delta}{2}]$ to decode. Specifically, we use the sum of the lengths of all the packets in the time interval $[t_{e_i} - \frac{\delta}{2}, t_{e_i} + \frac{\delta}{2}]$ as the length of the watermark encoding packet and apply that to the watermark bit decoding function (3).

Due to network delay jitter and/or active timing perturbation by the adversary, the exact arrival time of watermark encoding packet P_{e_i} may be different from t_{e_i}. Fortunately, the decoding can self-synchronize with the encoding by leveraging an intrinsic property of our hybrid length-timing watermarking scheme. Specifically, if the decoding of a watermarked flow uses the wrong offset or wrong seeds (s and s_t), then the decoded l-bit watermark W' will almost always have about $\frac{l}{2}$ bits matched with the true watermark W. This gives us an

easy way to determine if we are using the correct offset, and we can try a range of possible offsets and pick the best decoding result.

4 Implementation and Experiment

To validate the practicality of our watermarking scheme, we implemented both the length-only algorithm (unencrypted traffic) and the length-timing hybrid algorithm (encrypted traffic). To let our watermarking proxy interact with a realistic but benign IRC bot, we obtained a sanitized version of Agobot from its source code, containing only benign IRC communication features. We ran the sanitized Agobot on a local machine to generate benign IRC traffic to test the effectiveness of our watermarking scheme across public IRC servers and PlanetLab nodes. At no time did we send malicious traffic to anyone in the course of our experiments.

4.1 Length-Only Algorithm (Unencrypted Traffic)

We implemented the length-only algorithm in a modified open-source IRC proxy server and ran a series of experiments using the sanitized Agobot and public Internet IRC servers. We were able to recover the watermark successfully from unencrypted traffic in all ten of our trials.

Modified IRC Bouncer To achieve greater flexibility, we added our watermarking functionality to an existing IRC bouncer (BNC) package, psyBNC. Having the watermarking implemented on a proxy server allows us to use it on all bots conforming to the standard IRC protocol. It eliminates the need to have access to a bot's source code to add the watermarking functionality: outgoing traffic is modified by the BNC after the bot sends it.

In order for psyBNC to act as a transparent proxy, it needs to be configured identically to the bot. The information required consists of the C&C server's hostname, the port, and an IRC nick consistent with the bot's naming scheme. This information can be gathered by running the bot and monitoring the outgoing network traffic. In order to trick the bot into connecting to the BNC rather than to the real C&C host, we also need to update our local DNS cache so that a lookup of the C&C server's hostname resolves to the IP of our BNC.

Once it has been configured with this information, the BNC is completely transparent to the bot: when it starts up, the bot is automatically signed into the real C&C server by the BNC. The bot now joins the botnet channel as if it were directly connected and then waits for the botmaster's instructions. All PRIVMSG traffic from the bot to the C&C server (and by extension, to the botmaster) is watermarked by the transparent BNC in between.

Experiment and Results To test our watermarking scheme, we devised an experiment that emulates the conditions of an Internet-wide botnet as closely

as possible. To simulate the botmaster and stepping stones, we used PlanetLab nodes in California and Germany. We used a live, public IRC server in Arizona to act as a C&C host, creating a uniquely-named channel for our experiments. Our channel consisted of two IRC users: the Test Bot was running a copy of the sanitized Agobot and the Botmaster was acting as the botmaster (see Figure 3). As the diagram indicates, all traffic sent by the Test Bot passes through the psyBNC server (WM Proxy) where the watermark is injected. The distances involved in this setup are considerable: the watermarked traffic traverses literally half the globe (12 time zones) before reaching its ultimate destination in Germany, with a combined round-trip time of 292 milliseconds on average (at the time of our experiment).

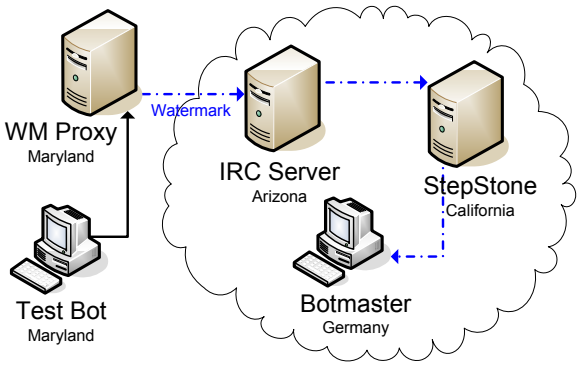

Fig. 3. Experiment setup for unencrypted traffic.

The objective is to be able to decode the full watermark in the traffic captured at the Stepping Stone and Botmaster. Since only PRIVMSG traffic from the Test Bot is watermarked, all other traffic (chaff) must be filtered out before decoding. Most of this chaff consists of messages from other users on the channel, PING/PONG exchanges, and JOIN/PART notifications from the channel. There could be additional chaff on the same connection if the botmaster is logged into multiple channels on the same IRC server. However, filtering out the chaff is trivial in the absence of encryption since all IRC messages contain the sender's nick. Therefore, we can easily isolate the watermarked packets based on the Test Bot's nick.

During our experiments, the psyBNC proxy was configured to inject a 32-bit watermark into a 64-packet stream. To generate traffic from the Test Bot, the Botmaster logged in and issued the `commands.list` command, causing the bot to send a list of all valid bot commands and their descriptions. We captured all traffic leaving the WM Proxy, arriving at the Stepping Stone, and arriving at the Botmaster. In ten trials with different (random) 32-bit watermarks, we were able to correct decode the full 32-bit watermark at all three monitoring

locations: the WM Proxy in Maryland, the Stepping Stone in California, and Botmaster in Germany.

4.2 Hybrid Length-Timing Algorithm (Encrypted Traffic)

To test the hybrid length-timing algorithm, we implemented a simple IRC bot that sends length-watermarked messages out at specific intervals. We used a "chaff bot" on the channel to generate controlled amounts of chaff. We were able to recover the watermark with a high success rate, even when high amounts of chaff were present.

Hybrid Length-Timing Encoder We implemented the hybrid encoding algorithm as a Perl program which reads in a previously length-only watermarked stream of messages and sends them out at specific times. To achieve highly precise timing, we used the `Time::HiRes` Perl package, which provides microsecond-resolution timers. At startup, the program uses the Mersenne Twister PRNG (via the `Math::Random::MT` package) to generate a list of departure times for all messages to be sent. Each message is sent at a randomly chosen time between 2 and 2.35 seconds after the previous message. The 2-second minimum spacing avoids IRC server packet throttling (more details are discussed in Section 4.2).

Hybrid Length-Timing Decoder The hybrid decoding script was also written in Perl, relying on the PCAP library to provide a standardized network traffic capture mechanism (via the `Net::Pcap` module). The program reads in a stream of packets (either from a live interface or from a PCAP file), then performs a sliding-window offset self-synchronization process to determine the time $t1$ of the first watermarked packet. To find the correct $t1$, the program steps through a range of possible values determined by the `offset`, `max`, and `step` parameters. It starts with $t1 =$`offset`, incrementing $t1$ by `step` until $t1 =$`(offset + max)`. It decodes the full watermark sequence for each $t1$, recording the number of bits matching the sought watermark W. It then chooses the $t1$ that produced the highest number of matching bits. If there are multiple $t1$ values resulting in the same number of matching bits, it uses the lowest value for $t1$. Figure 4 illustrates the synchronization process, showing that the correct $t1$ is near 6 seconds: 5.92 sec has 32 correct bits. For all incorrect $t1$ values, the decoding rate was significantly lower, averaging 14.84 correct bits.

Experiment and Results The experiment setup in this case was similar to the unencrypted experiment described in Section 4.1. The three main differences were: 1) a single Source computer producing watermarked traffic on its own replaced the Test Bot and WM Proxy; 2) the connection between the Botmaster and the IRC server (via StepStone) was encrypted using SSL/TLS; and 3) we used a different IRC server because the one in Arizona does not support SSL/TLS connections. The IRC server in this case happens to be located in Germany, but

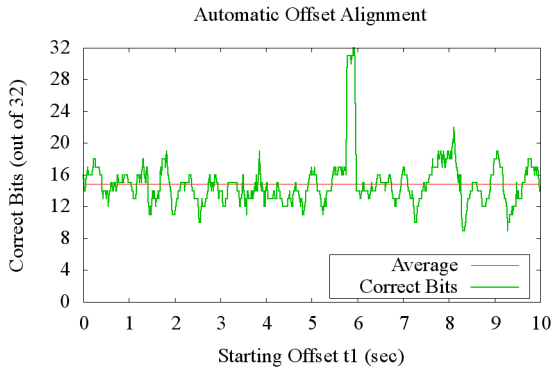

Fig. 4. Offset Self-Synchronization via Offset Sliding-Window

not in the same place as the Botmaster. Please refer to Figure 5 for the full experiment setup. In this configuration, the distances involved are even greater, with the watermarked traffic traversing the equivalent of the entire globe (24 time zones). The combined round-trip time from Source to Botmaster was 482 milliseconds (on average) at the time of our experiment.

To handle encryption, the parameters for the length-only algorithm were adjusted to ensure that the bucket size matched or exceeded the encryption block size. Most SSL/TLS connections use a block size of 128 bits (16 bytes), though 192 and 256 bits are also common. To ensure that each added bucket also causes another encrypted block to be added to the message, the bucket size has to be greater than or equal to the block size. For our experiment, we used a bucket size of 16 bytes, which was sufficient for the 128-bit block size used in the SSL/TLS connection. For compatibility with the larger block sizes (192 and 256 bits), a bucket size of 32 bytes can be used.

For the experiments, the Source produced a stream of 64 packets, containing a randomly generated 32-bit watermark. The Chaff Bot produced a controlled amount of background traffic, spacing the packets at random intervals between 1 and 6 seconds (at least 1 second to avoid throttling). In addition to our Control run (no chaff), we ran five different chaff levels (Chaff 1 to 5). The number refers to the maximum time between packets (not including the minimum 1-second spacing). For example, for the Chaff 1 run, packets were sent at a random time between 1 and 2 seconds. Thus, one packet was sent on average every 1.5 seconds, resulting in a chaff rate of approximately $1/1.5 = 0.667$ packets/sec.

We captured network traffic in three places: 1) traffic from Source and Chaff Bot to IRC Server; 2) traffic arriving at StepStone from IRC Server; and 3) traffic arriving at Botmaster from StepStone. Traffic in all three locations includes both watermark and chaff packets. We decoded the traffic at each location, recording the number of matching bits. For decoding, we used a value of 200 milliseconds for the timing window size δ and a sliding offset range from 0 to 10 seconds. This δ value was large enough to account for possible jitter along the stepping stone

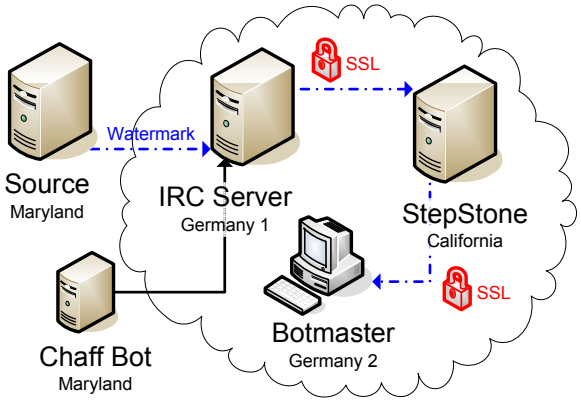

Fig. 5. Experiment setup for encrypted traffic.

chain but small enough to make it unlikely that a chaff packet appears within δ of an encoding packet. We also measured the actual chaff rate based on the departure times of each chaff packet, and these were very close to the expected rates based on an even distribution of random departure times. We repeated this process three times for each chaff level, resulting in a total of 18 runs. Our experiment results are summarized in Table 1, with each column representing the average values from three trials.

Monitoring Location	**Chaff 1**	**Chaff 2**	**Chaff 3**	**Chaff 4**	**Chaff 5**	**Control**
Chaff Rate (packets/sec)	0.6719	0.4976	0.4274	0.3236	0.2872	no chaff
Source - Maryland	29.67	30.33	29.67	30.33	30.33	32
StepStone - California	31	32	31.67	31.67	32	32
Botmaster - Germany	31	31.67	32	31.67	31.67	32

Table 1. Experiment results for encrypted traffic: Recovered watermark bits (out of 32) at each monitoring station along the watermark's path (averaged from three trials).

We had near-perfect decoding along the stepping-stone chain for all chaff rates of 0.5 packets/sec and below. Only when the chaff rate rose above 0.5 packets/sec did the chaff start having a slight impact, bringing the decoding rate down to an average of 31 bits. The overall average decoding rate at the StepStone and Botmaster was 31.69 bits, or 99.05 percent. The lowest recorded decoding rate during our experiments was 28 bits, so we can use a Hamming distance threshold of $h = 4$ to obtain a 100 percent true positive rate (TPR) and a false positive rate (FPR) of 9.64×10^{-6}.

The most surprising result is that in all cases where chaff was present, the decoding rate was worse at the Source than downstream at the StepStone and Botmaster. After examining the network traces in detail, we realized that this

behavior was due to the presence of traffic queuing and throttling on the IRC Server. To avoid flooding, IRC servers are configured to enforce minimum packet spacings, and most will throttle traffic at 0.5 to 1 packets/sec. To confirm this behavior, we sent packets to the IRC Server in Germany at random intervals of 100 to 300 milliseconds. For the first 5 seconds, packets were passed on immediately, but after that the throttling kicked in, limiting the server's outgoing rate to 1 packet/sec. After about 2 minutes, the server's packet queue became full with backlogged packets, and it disconnected our client. Figure 6 illustrates the effect of throttling on the packet arrival times, including the 5-second "grace period" at the beginning.

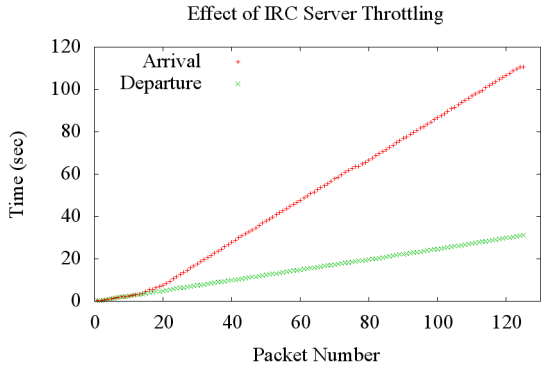

Fig. 6. IRC server throttling causes packets to be spaced apart further upon arrival.

In the context of our hybrid encoding scheme, IRC message queuing is highly beneficial because it dramatically reduces the chances that chaff and encoding packets will appear close to each other. At the Source, packets appear at the exact intervals they are sent, which could be less than δ and therefore affect decoding. However, this interval will be increased due to queuing by the IRC server. By the time the packets reach the StepStone and Botmaster, they no longer affect decoding because they are more than δ apart. In our experiments, we observed that the IRC server introduced a distance of at about 130 milliseconds between packets due to queuing. Since our δ value was 200 milliseconds, this made it unlikely that two packets would arrive in the same slot.

5 Discussion and Future Work

Our experiments show that our watermarking scheme is effective in tracing the botmaster of IRC-based botnets, which are still the predominant type in the wild [17, 21, 28]. Our watermark can be recovered with a high degree of accuracy even when the watermarked botnet C&C traffic is encrypted across multiple stepping stones and mixed with other flows.

In theory, our flow watermarking technique could be applied to trace any real-time and interactive botnet C&C traffic. Therefore, it could be used to track the botmaster of peer-to-peer (P2P) botnets which have started appearing recently [13]. However, HTTP-based botnets present a much higher level of traceback difficulty: the messages do not get passed from the bot to the botmaster in real-time. They are typically stored on the C&C server until the botmaster retrieves them in bulk, usually over an encrypted connection such as SSH. Due to this, any approach that relies on properties of individual packets (such as length and timing) will be unsuccessful.

When SSH is used as the final hop in a chain of stepping stones, it presents unique challenges. In this case, the botmaster uses SSH to log into a stepping stone, launches a commandline-based IRC client on that host, and uses this IRC client to connect to his botnet (possibly via more stepping stones). In this capacity, SSH is not acting as a proxy, passing on messages verbatim like psyBNC or SOCKS. Instead, it transfers the "graphical" screen updates of the running IRC client, which is not necessarily correlated to the incoming IRC messages. This situation is challenging for our approach because the application-layer content is transformed, altering the relative lengths of packets. We are working on this problem, but we have been unable to explore it in detail. Notice that if SSH is used in a tunnelling capacity (such as port forwarding or a SOCKS proxy) in the middle of a stepping stone chain, this limitation does not apply.

Once the botmaster become aware of the flow watermarking tracing approach, he may want to corrupt the embedded watermark from intermediate stepping stones. However, since the padding characters could be almost any character and they are inserted randomly in the botnet message, it would be difficult for any intermediate stepping stone to identify and remove the padding characters without knowing the original unwatermarked message. The botmaster may be able to detect and identify the padding if he knows exactly what he is expecting for. However, once he receives the watermarked message, the watermarked message has already left the complete trail toward the botmaster. The botmaster could have intermediate stepping stones to perturb the length of the passing botnet messages by adding random padding such as white space. Since the watermark is embedded in the length difference between randomly chosen packets, the negative impact of the padding by the adversary tends to cancel each other. We can further mitigate the negative impact by using redundant pairs of packets to encode the watermark. However, this would increase the number of packets needed. So this is essentially a tradeoff between the robustness and the efficiency.

As previously discussed in Section 2.2, our approach requires at least partial network coverage of distributed monitoring stations. This is a common requirement for network traceback approaches, especially since the coverage does not need to be global. The accuracy of the trace is directly proportional to the number and placement of monitoring nodes.

Our work is a significant step in the direction of live botmaster traceback, but as the title implies, it is indeed a first step. Our future work in this area includes

the exploration of several topics, including optimal deployment of monitoring nodes, SSH traffic on the last hop, further data collection with longer stepping stone chains, and traceback experiments on in-the-wild botnets.

6 Related Work

The botnet research field is relatively new, but many papers have been published in the last few years as the botnet threat has accelerated. As one of the first in the botnet arena, the Honeynet Project [1] provided a starting point for future exploration of the problem. A comprehensive study at Johns Hopkins University [21] constructed a honeypot-based framework for acquiring and analyzing bot binaries. The framework can automatically generate rogue bots (drones) to actively infiltrate botnets, which is the first step in injecting a watermark and tracing the botmaster.

Most early botnet work focused on defining, understanding, and classifying botnets. Some examples are papers by Cooke et al. [6], Dagon et al. [7], Ianelli and Hackworth [17], Barford and Yegneswaran [2], and Holz's summary in *Security & Privacy* [16]. Since then, bot detection has become more of a focal point and many techniques have been proposed. Binkley and Singh [3] presented an anomaly-based detection algorithm for IRC-based botnets. Goebel and Holz [11] reported success with their Rishi tool, which evaluates IRC nicknames for likely botnet membership. Karasaridis et al. [18] described an ISP-level algorithm for detecting botnet traffic based on analysis of transport-layer summary statistics. Gu et al. [15] detailed their BotHunter approach, which is based on IDS dialog correlation techniques. They also published a related paper in 2008 [14] where they introduce BotSniffer, a tool for detecting C&C traffic in network traces.

Despite a large amount of literature regarding botnet detection and removal, relatively little work has been done on finding and eliminating the root cause: the botmaster himself. An earlier paper by Freiling et al. [10] describes a manual method of infiltrating a botnet and attempting to locate the botmaster, but the approach does not scale well due to lack of automation.

In the general traceback field, there are two main areas of interest: 1) network-layer (IP) traceback and 2) tracing approaches resilient to stepping stones. The advent of the first category dates back to the era of fast-spreading worms, when no stepping stones were used and IP-level traceback was sufficient. A leading paper in this area is Savage et al. [25], which introduced the probabilistic packet marking technique, embedding tracing information an IP header field. Two years later, Goodrich [12] expounded on this approach, introducing "randomize-and-link" with better scalability. A different technique for IP traceback is the log/hash-based one introduced by Snoeren et al. [26], and enhanced by Li et al. [19].

There are a number of works on how to trace attack traffic across stepping stones under various conditions. For example, [33, 34, 8, 32, 31, 4, 29, 30] used inter-packet timing to correlate encrypted traffic across the stepping stones and/or low-latency anonymity systems. Most timing-based correlation schemes

are passive, with the exception of the three active methods [31, 29, 30]. Our proposed method is based on the same active watermarking principle used in these three works. However, our method differs from them in that it uses the packet length, in addition to the packet timing, to encode the watermark. As a result, our method requires much fewer packets than methods [31, 29, 30] to be effective.

7 Conclusion

The key contribution of our work is that it addresses the four major obstacles in botmaster traceback: 1) stepping stones, 2) encryption, 3) flow mixing and 4) a low traffic volume between bot and botmaster. Our watermarking traceback approach is resilient to stepping stones and encryption, and it requires only a small number of packets in order to embed a high-entropy watermark into a network flow. The watermarked flow can be tracked even when it has been mixed with randomized chaff traffic. Due to these characteristics, our approach is uniquely suited for real-time tracing of the interactive, low-traffic botnet C&C communication between a bot and its botmaster. We believe that this is the first viable technique for performing live botmaster traceback on the Internet.

We validated our watermarking traceback algorithm both analytically and experimentally. In trials on public Internet IRC servers, we were able to achieve virtually a 100 percent TPR with an FPR of less than 10^{-5}. Our method can successfully trace a watermarked IRC flow from an IRC botnet member to the botmaster's true location, even if the watermarked flow 1) is encrypted with SSL/TLS; 2) passes through several stepping stones; and 3) travels tens of thousands of miles around the world.

8 Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments that helped to improve the presentation of this paper. This work was partially supported by NSF Grants CNS-0524286, CCF-0728771 and CNS-0716376.

References

1. P. Bächer, T. Holz, M. Kötter, and G. Wicherski. Know Your Enemy: Tracking Botnets. March 13, 2005. See http://www.honeynet.org/papers/bots/.
2. P. Barford and V. Yegneswaran. An Inside Look at Botnets. In *Proc. Special Workshop on Malware Detection, Advances in Info. Security.* Springer, 2006.
3. J. Binkley and S. Singh An Algorithm for Anomaly-based Botnet Detection. In *Proc. 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI).* San Jose, CA, July 7, 2006, pp. 43–48.
4. A. Blum, D. Song, and S. Venkataraman. Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds. In *Proc. 7th Symposium on Recent Advances in Intrusion Detection (RAID 2004).* Springer, October 2004.

5. Z. Chi, Z. Zhao. Detecting and Blocking Malicious Traffic Caused by IRC Protocol Based Botnets. In *Proc. Network and Parallel Computing (NPC 2007)*. Dalian, China, September 2007, pp. 485–489.
6. E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disturbing Botnets. In *Proc. Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*. Cambridge, MA, July 7, 2005, pp. 39–44.
7. D. Dagon, G. Gu, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A Taxonomy of Botnets. unpublished paper, 2005.
8. D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit and S. Staniford. Multiscale Stepping Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *Proc. 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002): LNCS-2516*, pp. 17–35. Springer, October 2002.
9. J. Evers. 'Bot herders' may have controlled 1.5 million PCs. http://news.com.com/2102-7350_3-5906896.html?tag=st.util.print
10. F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent DoS Attacks. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS)*. Milan, Italy, Sept. 2005.
11. J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *Proc. First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA, April 10, 2007.
12. M. T. Goodrich. Efficient Packet Marking for Large-scale IP Traceback. In *Proc. 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pp. 117–126. ACM, October 2002.
13. J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-Peer Botnets: Overview and Case Study. In *Proc. First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA, April 2007.
14. G. Gu, J. Zhang, and W. Lee BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic In *Proc. 15th Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2008.
15. G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation In *Proc. 16th USENIX Security Symposium*. Boston, MA, August 2007.
16. T. Holz. A Short Visit to the Bot Zoo. *Sec. and Privacy*, 3(3), 2005, pp. 76–79.
17. N. Ianelli and A. Hackworth. Botnets as a Vehicle for Online Crime. In *Proc. 18th Annual Forum of Incident Response and Security Teams (FIRST)*. Baltimore, MD, June 25-30, 2006.
18. A. Karasaridis, B. Rexroad, and D. Hoein. Wide-Scale Botnet Detection and Characterization. In *Proc. First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA, April 10, 2007.
19. J. Li, M. Sung, J. Xu and L. Li. Large Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proc. 2004 IEEE Symposium on Security and Privacy*, IEEE, 2004.
20. R. Naraine. Is the Botnet Battle Already Lost? http://www.eweek.com/article2/0,1895,2029720,00.asp
21. M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proc. 6th ACM SIGCOMM on Internet Measurement*. Rio de Janeiro, Brazil, October 25-27, 2006.
22. P. F. Roberts. California Man Charged with Botnet Offenses. http://www.eweek.com/article2/0,1759,1881621,00.asp

23. P. F. Roberts. Botnet Operator Pleads Guilty. http://www.eweek.com/article2/0,1759,1914833,00.asp
24. P. F. Roberts. DOJ Indicts Hacker for Hospital Botnet Attack. http://www.eweek.com/article2/0,1759,1925456,00.asp
25. S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. ACM SIGCOMM 2000*, pp. 295–306. Sept. 2000.
26. A. Snoeren, C. Patridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP Traceback. In *Proc. ACM SIGCOMM 2001*, pp. 3–14. ACM, September 2001.
27. Symantec. Symantec Internet Security Threat Report – Trends for January 06 - June 06. Volume X, September 2006.
28. Trend Micro. Taxonomy of Botnet Threats. Trend Micro Enterprise Security Library, November 2006.
29. X. Wang, S. Chen, and S. Jajodia. Tracking Anonymous, Peer-to-Peer VoIP Calls on the Internet. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS 2005)*, October 2007.
30. X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In *Proc. 2007 IEEE Symposium on Security and Privacy (S&P 2007)*, May 2007.
31. X. Wang and D. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *Proc. 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pp. 20–29. ACM, October 2003.
32. X. Wang, D. Reeves, and S. Wu. Inter-packet Delay Based Correlation for Tracing Encrypted Connections Through Stepping Stones. In *Proc. 7th European Symposium on Research in Computer Security (ESORICS 2002), LNCS-2502*, pp. 244–263. Springer-Verlag, October 2002.
33. K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *Proc. 6th European Symposium on Research in Computer Security (ESORICS 2000), LNCS-1895*, pp. 191–205. Springer-Verlag, October 2002.
34. Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proc. 9th USENIX Security Symposium*, pp. 171–184. USENIX, 2000.