On the Secrecy of Spread-Spectrum Flow Watermarks

Xiapu Luo, Junjie Zhang, Roberto Perdisci, and Wenke Lee

College of Computing, Georgia Institute of Technology
{csxpluo,jjzhang,wenke}@cc.gatech.edu, perdisci@gtisc.gatech.edu

Abstract. Spread-spectrum flow watermarks offer an invisible and ready-to-use flow watermarking scheme that can be employed to stealthily correlate the two ends of a network communication. Such technique has wide applications in network security and privacy. Although several methods have been proposed to detect various flow watermarks, few can effectively detect spread-spectrum flow watermarks. Moreover, there is currently no solution that allows end users to eliminate spread-spectrum flow watermarks from their flows *without* the support of a separate network element. In this paper, we propose a novel approach to detect spread-spectrum flow watermarks by leveraging their intrinsic features. Contrary to the common belief that Pseudo-Noise (PN) codes can render flow watermarks invisible, we prove that PN codes actually facilitate their detection. Furthermore, we propose a novel method based on TCP's flow-control mechanism that provides end users with the ability to autonomously remove spread-spectrum flow watermarks. We conducted extensive experiments on traffic flowing both through one-hop proxies in the PlanetLab network, and through Tor. The experimental results show that the proposed detection system can achieve up to 100% detection rate with zero false positives, and confirm that our elimination system can effectively remove spread-spectrum flow watermarks.

1 Introduction

Flow watermarks can be employed to trace end-to-end communications, even when they flow through stepping stones or anonymity networks [27]. By secretly embedding a (sequence of) watermark(s) into network flows at a location close to one end, it is possible to identify the other end of the communication by detecting the presence of the watermark in the traffic without being noticed by either end (see Figure 1). For example, flow watermarks may be used by law enforcement agencies to detect stepping stones used by attackers [20], to determine whether a certain user is accessing a specific (e.g., terrorism-related) web site [27,28], to trace communications among *bot*-compromised machines [21], to correlate anonymous Peer-to-Peer VoIP calls [26], etc.

If an adversary detects that her flows have been watermarked, she may be able to remove the watermarks, or deliberately cause false alarms by embedding the detected watermarks into legitimate flows [19,16]. Therefore, it is important to evaluate the *secrecy* of a flow watermarking scheme before deploying it in a real network. In this paper, we investigate the secrecy of spread-spectrum flow watermarks (SSFW) [28] from the following two aspects: (1) can SSFW be accurately detected? (2) can SSFW be effectively removed from network flows?

Recently, a few methods have been proposed that aim to detect SSFW [16,15]. Kiyavash et al. proposed a *multi-flow* attack, base on the assumption that SSFW is used to simultaneously embed the same watermark sequence into *multiple* flows. The detection approach leverages the length of low-throughput period as a metric to detect SSFW. The authors assume that that normal traffic follows the Markov-Modulated Poisson Process model, while watermarked flows would not fit this model [16]. However, it has been shown that the multi-flow attack can be evaded, if the encoder uses different Pseudo-Noise (PN) codes or if the watermark sequence changes for different flows [14]. The detection approach proposed by Jia et al. [15] leverages the fact that SSFW employs a single m-sequence, a specific PN code with good autocorrelation features, to spread the bits of a watermark sequence along a flow. Using only one m-sequence leads to obvious self-similarity in watermarked traffic. However, Jia et al. indicated that this detection approach may be evaded by using different m-sequences or orthogonal PN codes to spread individual bits of a watermark sequence [15].

To the best of our knowledge, there currently exists no solution that allows end users to remove SSFW from their flows without the support of a middlebox (e.g. a router, proxy, or a relay host within an anonymity network). In addition, although a middlebox may remove SSFW by altering the throughput of each individual network flow crossing it, few middleboxes actually deploy such watermark elimination strategy because of the consequent heavy overhead.

In this paper, we propose a novel detection system that is able to identify the existence of SSFW within a given network flow. In addition, we propose a novel elimination system that enables end users to autonomously remove SSFW from their flows. Our detection system leverages SSFW's intrinsic features. Unlike existing detection approaches (e.g., [16,28]), our approach does not assume that the same watermarks are simultaneously embedded in multiple flows, and does not assume an ideal traffic model. Moreover, we do not assume that SSFW uses only a single PN code. Instead, we assume that any other kind of valid PN codes (e.g. orthogonal PN codes) [12] could be employed.

Our detection approach is based on the following key observations: (1) similar to amplitude modulation in signal processing, SSFW causes alternate low-throughput and high-throughput periods in a watermarked flow; (2) PN codes make the detection of spread-spectrum flow watermarks easier, because they increase the number of low-throughput periods; (3) unlike spread-spectrum radio communications, which spread a radio signal over a wide frequency range, SSFW embeds watermarks separately in each *one* flow, instead of spreading them over a wide set of flows. Therefore, the detection system simply needs to examine individual flows (see Section 3).

Our elimination system leverages TCP's basic flow control mechanism to regulate the throughput of *incoming* traffic. More precisely, our system modifies the

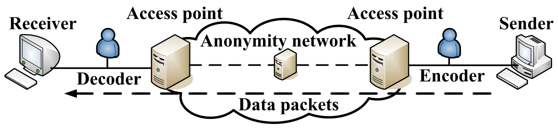


Fig. 1. Watermarks are embedded into the traffic by the Encoder, and identified by the Decoder

advertising window in TCP packets sent by either an end user or a middlebox to its upstream node to modulate the throughput. It is worth noting that our approach is independent from application-layer flow control and congestion control mechanisms. In fact, our approach works even in those cases when application layer flow control and congestion control mechanisms cannot remove SSFW.

In summary, this paper makes the following main contributions:

1. We propose a novel watermark detection system that is able to identify whether a flow has been watermarked using SSFW. Our approach removes many of the assumptions required by existing SSFW detection methods.
2. We propose a novel receiver-based system to remove SSFW. Our system can be deployed at either the end-user or middlebox level. To the best of our knowledge, ours is the first practical system that allows end-users to autonomously remove SSFW from their flows.
3. We performed extensive experiments to evaluate the proposed detection and elimination systems. The experimental results show that our system is able to successfully detect SSFW and remove the watermarks from TCP flows.

The rest of the paper is organized as follows. We describe the threat model and introduce related work in the next section. Section 3 and section 4 present the detection scheme and the elimination scheme respectively. We describe the experiment results in Section 5 and conclude the paper in Section 6.

2 Background

2.1 Threat Model

Figure 1 shows the threat model used in this paper. Assume an entity (e.g., law enforcement) intends to find out whether there exists an end-to-end network communication between a sender S and a receiver R. To this end, a watermark encoder E is placed between S and its neighbor network nodes, and a watermark decoder D is placed at the other end of the communication, between R and its neighbor nodes. E manipulates the throughput of all flows originating from S to embed a sequence of watermarks. On the other hand, all flows received by R are investigated by D to determine whether they carry watermarks previously embedded by E. If that is the case, this means that a communication between S and R is in place. Along the path between S and R there are n ($n \geq 1$) middleboxes. Each middlebox behaves as a proxy or relay host, therefore separating the logical connection between S and R into multiple loosely coupled TCP connections. This scenario is typical of stepping stones [29,8], anonymity networks (e.g. Anonymizer (www.anonymizer.com) or Tor [6]), and HTTP/SOCKS proxies.

Our detection system (see Section 3) can be located at a middlebox, between the encoder and the decoder, to determine whether or not the flows going through the middlebox have been watermarked. If so, the middlebox can use a traffic shaper to remove the watermarks from the *outgoing* traffic sent to the next hop.

We also consider the case of non-cooperative middleboxes, and we assume the end user (the receiver) wants to make sure that her flows cannot be traced back. In this case, R can apply our elimination system (see Section 4) to blindly remove the watermarks from all her *incoming* flows before they can be identified by the decoder.

2.2 Spread-Spectrum Flow Watermarks

A target flow's throughput is the carrier of the spread-spectrum flow watermark. A watermark comprises of a sequence of bits denoted as $W = \{w_1, \ldots, w_M\}$, where M is the length of a watermark. Instead of using w_i ($i = 1, \ldots, M$) to directly modulate a flow's throughput, the encoder first maps w_i to a PN code according to: $w_i \rightarrow \begin{cases} Z \text{ if } & w_i = 1, \\ \overline{Z} \text{ if } & w_i = -1, \end{cases}$ where $Z = \{z_1, \ldots, z_V\}$ is a V-bit PN code and $\overline{Z}$ is the complement of Z. After obtaining the new sequence of MV bits indicated as $W_{DSSS} = \{Z_1, \ldots, Z_M\}$, the encoder uses each bit in W_{DSSS} to modulate a flow's throughput. More precisely, if a bit is -1, the encoder will cause a low-throughput period of T_c (called *chip* duration) by causing many packet losses in the target flow. Otherwise, the encoder will maintain a high-throughput period of T_c by doing nothing or causing less packet loss in the target flow [7].

A PN code is a special binary sequence. Before introducing general features of a PN code, we give the definition of *run* in a binary sequence.

Definition 1. *Given a binary sequence $B = \{b_1, \ldots, b_L\}$, a run is defined as a sequence of $\{b_j, \ldots, b_k\}$ where $b_j = b_{j+1} = \ldots = b_k$ and $b_{j-1} \neq b_j$ and $b_{k+1} \neq b_k$. Its length is equal to $k-j+1$. Note that if $j = 1$, b_1 is the start of a run. Similarly, if $k = L$, b_L is the end of a run.*

Golomb indicated that a PN code may have one or many following properties [12]: (1) the number of 1 is approximately equal to the number of -1. (2) runs of 1 or -1 occur with probability that is inversely proportional to the length of runs. (3) its autocorrelation has the maximal value in the middle and declines quickly at the ends. Yu et. al. employed m-sequence, which has all above properties [12], to implement the spread-spectrum flow watermarks [28].

2.3 Countermeasures

Kiyavash et al. proposed the multi-flow attack to detect SSFW [16]. They assume that the same watermark is embedded into *multiple* flows simultaneously and normal traffic follows the Markov modulated Poisson process [16]. Our detection system does not need such assumptions. The multi-flow attack exploits the observation that SSFW may cause a long low-throughput period on several flows comparing with a trained model. However, they also showed that the multi-flow attack can be evaded when the encoder applies different PN codes or flow watermarks to different flows [14]. Moreover, changing the position of a watermark in a flow may disable the multi-flow attack because the number of flow combinations that need investigation increases exponentially [16,14]. When facing multiple flows, our detection system only needs to investigate them one by one because we exploit the fundamental difference between network flows and radio signals, which is detailed in section 3.1.

The watermarked flow may show self-similarity because Yu et. al. used one m-sequence code to spread every bit in a watermark [28] and all m-sequence codes have excellent autocorrelation feature [7]. Exploiting this observation, Jia et al. proposed a detection approach that employs mean-square autocorrelation to measure the similarity between a modulated traffic segment and the same segment shifted by certain period [15]. However, it is easy to evade this method by using different m-sequence codes or orthogonal PN codes to spread individual bits of a watermark [15]. Our detection system can handle both cases.

3 Detection System

We first explain the traffic anomalies caused by SSFW in Section 3.1 and 3.2, and then elaborate on our detection scheme in section 3.3 and 3.4.

3.1 Basic Idea

Our detection scheme leverages the anomalies steming from SSFW's intrinsic features and takes advantage of fundamental differences between network flows and radio signals. We can first notice that when applied to a network flow SSFW causes an abnormal sequence of low-throughput periods in the flow. This happens because the encoder needs to throttle the flow's throughput to a low value for

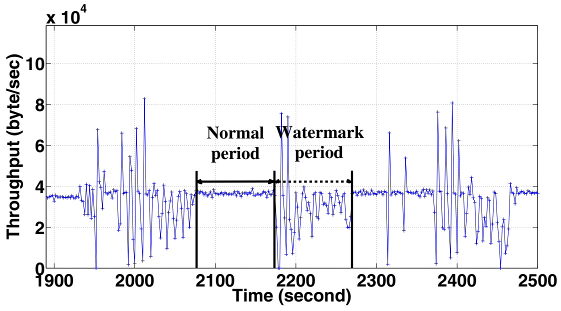


Fig. 2. Throughput of a watermarked flow that went through Tor network

a given period T_c when the bit to be embedded is -1 (notice that SSFW uses a binary encoding with values equal to either -1 or +1). The low-throughput periods caused by -1 bits are noticeably different from throughput degradations caused by network congestion in terms of the throughput level, duration and frequency. The reason is that network congestion is out of the control of the encoder and throughput degradations caused by network congestion may mislead the decoder. Therefore, to correctly decode a watermark bit and distinguish it from noise due to network congestion, Yu et al. have specified in [28] (Equation 12 and Figure 10) that the encoder needs to implement the following strategies:

– Increase the difference between the high-throughput and low-throughput levels. Since the maximum high-throughput is determined by the network, this strategy can only be achieved by decreasing the value of low-throughput.
– Increase the duration of low-throughput periods (i.e. T_c).
– Increase the length of the PN code, thus causing a higher number of low-throughput periods.

Based on this observations, the goal of our detection system is to detect SSFW by identifying the presence of anomalous sequences of low-throughput periods in a network flow. Figure 2, which is based on the data from [28], illustrates the throughput of a watermarked flow crossing the Tor network. Its throughput is computed in every *chip* duration (i.e. 2 seconds) [28]. We highlight two periods: the watermark period in which a watermark was embedded into the flow, and the normal period when the encoder is idle. It is easy to notice the higher number of low-throughput points during the watermark period, compared to the normal period. Since each low-throughput point in Figure 2 indicates the aggregated throughput during a chip duration, it represents a low-throughput period when the throughput is aggregated within a small time unit. We describe the selection of the basic time unit in section 3.3.

Second, we prove in section 3.2 that using PN codes to spread watermarks increases the number of low-throughput periods significantly. This feature allows our detection system to quickly identify SSFW.

Third, spread spectrum was originally designed to spread radio signal from a small frequency range to a wider frequency range [7]. A fundamental difference between radio signals and network flows is that although the spread-spectrum technique can spread the radio signal's energy to a wide range of frequencies and recover the original signal from those frequencies, applying SSFW to a network flow only affects that one flow and not a set of flows. Therefore, just like a decoder that only needs to inspect one flow to identify the embedded watermarks, our detection system only needs to investigate individual flows.

Based on the above observations, our detection scheme consists of two steps:

1. Locate low-throughput periods in a flow (section 3.3).
2. Detect abnormal sequences of low-throughput periods (section 3.4).

3.2 Low-Throughput Periods Resulted from PN Codes

We use R_1 and R_{-1} to denote the number of runs (see Section 2.2) of 1 and the number of runs of -1 in a binary sequence. R_{-1} is equal to the number of low-throughput periods. Without loss of generality, we assume that the flow's throughput is high before and after the watermark period. Since runs of 1 and runs of -1 alternate, the relationship between R_1 and R_{-1} falls into one of the following scenarios: (1) $R_{-1} = R_1$; (2) If $R_{-1} \neq R_1$ and $b_1 = -1$, $R_{-1} = R_1 + 1$; (3) If $R_{-1} \neq R_1$ and $b_1 = 1$, $R_{-1} = R_1 - 1$ [11].

Lemma 1 shows that the expected number of runs has the maximal value $1+\frac{L}{2}$ when the number of 1 is equal to the number of -1. For the ease of explanation, we assume that L is an even number. According to the relationship between R_{-1} and R_1 listed above, we know that the expected number of R_{-1} reaches its maximal value. Since PN codes have similar number of 1 and -1, they possess a large R_{-1}.

Lemma 1. *In a L-bit binary sequence, the expected number of runs reaches the maximal value $1 + \frac{L}{2}$ when the number of 1 is equal to that of -1.*

Proof. The expected number of runs (i.e. $R_{-1} + R_1$) in a L-bit binary sequence is equal to $1 + \frac{2L_{-1}(1-L_{-1})}{L}$ where L_{-1} is the number of -1 [11]. Since $2L_{-1}(L - L_{-1}) \leq \frac{L^2}{2}$ and the inequality becomes equality when $L_{-1} = \frac{L}{2}$, we get the maximal value $1 + \frac{L}{2}$ when the number of 1 is equal to that of -1.

Since SSFW turns an M-bit watermark into an MV-bit binary sequence using a V-bit PN code, R_{-1} is increased significantly. Without loss of generality, we assume that both the original watermark and the spread watermark are random sequences. In this case, we use the Corollary 2.1 in [11] to compute the probability of R_{-1}. Figure 3 illustrates R_{-1}'s PDF in 16-bit watermarks and that in the corresponding watermarks spread by 7-bit PN codes. Obviously, the spread watermarks have much larger R_{-1} than the original watermark. The average R_{-1} has been increased by around 7.

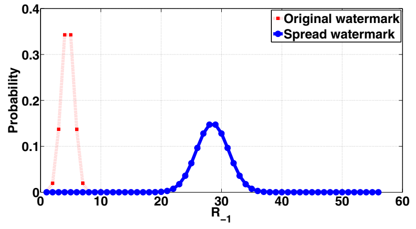


Fig. 3. The PDF of number of runs of -1 (i.e. R_{-1}) in 16-bit watermarks and that in the corresponding watermarks spread by 7-bit PN codes

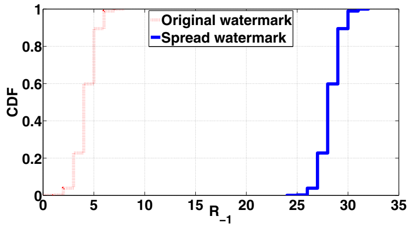


Fig. 4. The CDF of the number of runs of -1 (i.e. R_{-1}) in 16-bit original watermarks and that in the corresponding watermarks spread by a m-sequence PN code $\{1, -1, -1, 1, 1, 1, -1\}$

Lemma 2 calculates the exact number of R_{-1} in a m-sequence that is used as the PN code in [28]. Lemma 2 indicates that when using a L-bit m-sequence to spread one bit, the number of low-throughput periods will increase by around $\frac{L+1}{4}$. Figure 4 illustrates the CDF of R_{-1} in all possible 16-bit watermarks and that in the corresponding watermarks spread by a m-sequence PN code $\{1, -1, -1, 1, 1, 1, -1\}$. Obviously, the spread watermarks have much larger R_{-1} than the original watermarks.

Lemma 2. *In a L-bit m-sequence, $R_{-1} = \frac{L+1}{4}$.*

Proof. The number of k-bit runs of -1 is equal to 2^{J-2-k} ($k = 1, \ldots, J-2$), where $J = \log_2(L+1)$ [7]. Since the maximal length of runs of -1 is $J-1$ and there is only one $(J-1)$-bit run of -1 [7], $R_{-1} = \sum_{k=1}^{J-2} 2^{J-2-k} + 1 = \frac{L+1}{4}$.

3.3 Locating Low-Throughput Periods

Our detection system computes a target flow's throughput in each basic time unit. We call these values as throughput samples. The system is independent of the transport layer protocol. For TCP flows, we let the basic time unit be the round-trip time (RTT), denoted as T_{rtt}, between the host where our detection system is located and its upstream host. The rational is that TCP packets are usually sent in burst within each RTT duration because of TCP's ACK-based self-clocking. Using a smaller period to compute throughput samples may lead to many useless zero values because the time for sending a burst of TCP packets is a small portion of RTT. Using a period larger than the chip duration to calculate throughput samples may blur low-throughput periods caused by the watermark. For UDP flows, the basic time unit could be set to the average inter-packet delay. Since the original SSFW targets on TCP flows [28] and many public proxies and anonymity networks (e.g. Tor) only support TCP connections, we evaluate our detection system using only TCP flows.

Given a sequence of throughput samples $\Pi = \{\pi_1, \pi_2, \ldots\}$, we construct a new binary sequence ($\hat{\Pi} = \{\hat{\pi}_1, \hat{\pi}_2, \ldots\}$) according to Equation (1):

$$\hat{\pi}_i = \begin{cases} 1 & \text{if } \pi_i - (1-\rho)\mu_\Pi > 0, \\ -1 & \text{if } \pi_i - (1-\rho)\mu_\Pi \le 0, \end{cases} \tag{1}$$

where μ_Π is the average value of Π and ρ ($0 < \rho < 1$) is a parameter. We define a low-throughput period as a sequence of throughput samples whose values are not larger than $(1-\rho)\mu_\Pi$ and the duration of such sequence is longer than T_r.

We found that it is proper to let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$, where μ_Π and σ_Π are the average value and the standard deviation of Π respectively. According to the one-side Chebyshev inequality [24], we have $Pr(\pi_i \le \mu_\Pi - K\sigma_\Pi) \le \frac{1}{1+K^2}$. When $K = 1$, $Pr(\pi_i \le \mu_\Pi - \sigma_\Pi) = Pr(\hat{\pi}_i = -1) \le \frac{1}{2}$. Since PN code $Z = \{z_1, \ldots, z_V\}$ has similar number of -1 and 1, $Pr(z_i = -1) \approx \frac{1}{2}$. For a V-bit m-sequence, $Pr(z_i = -1) \equiv \frac{V+1}{2V}$. Since SSFW degrades a flow's throughput when the bit to be embedded is -1, the probability of observing a low-throughput sample approximates $\frac{1}{2}$. Therefore, by letting $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$, we have high probability to observe all low throughput values.

Throughput degradations caused by network congestion are noise to both SSFW's decoder and our detection algorithm. We exploit TCP's congestion control mechanism to filter out low-throughput periods caused by network congestions that occur on the path where the detection system is located. Detailed information can be found in [17].

3.4 Detection Algorithm

After locating a sequence of low-throughput periods, we employ the sequential probability ratio testing (SPRT) to carry out the detection. More precisely, as many recent Internet measurement studies have shown that the packet loss events in the Internet could be modeled as a Poisson process [2,18], we use SPRT to detect the abnormal increment in the rate of such events [9,13].

Let $x(t)$ be a poisson process modeling low-throughput periods. Its probability function is $Pr(x, \lambda) = \exp(-\lambda t)\frac{(\lambda t)^x}{x!}$, where λ is the rate of low-throughput periods. We define two hypotheses H_0 and H_1 as follows:

- H_0, the rate of low-throughput periods is within normal range.
- H_1, the rate of low-throughput periods is abnormal.

The log-likelihood ratio is defined as: $\Theta(t) = \ln \frac{Pr(x(t)|H_1)}{Pr(x(t)|H_0)} = x(t)\ln(\gamma) + \lambda_0(1-\gamma)t$, $\gamma = \frac{\lambda_1}{\lambda_0}$, where λ_0 and λ_1 indicate the normal rate of low-throughput periods and the abnormal one individually.

We choose H_0 if $\Theta(n) \leq B$ or select H_1 if $\Theta(n) \geq A$. Otherwise, the detection system continues monitoring. A and B are determined according to two user-defined parameters: α is the probability of false positive (i.e. select H_1 but H_0 is correct.) β is the probability of false negative (i.e. select H_0 but H_1 is correct.). Dvoretzky et. al. proved that $B = \ln\frac{\beta}{1-\alpha}$ and $A \leq \ln\frac{1-\beta}{\alpha} \leq A + \ln(\gamma)$ [9]. Since computing the exact value of A is time-consuming, Haggstrom suggested that $A \approx \ln(\frac{1-\beta}{\alpha}) - \frac{\ln(\gamma)}{3}$ [13].

Note that waiting periods, denoted as y, between consecutive events in a Poisson process follow the exponential distribution, whose probability function is $\lambda e^{-y\lambda}$. Haggstrom constructed an alternative SPRT of H_0 versus H_1 based on the waiting times after N observations. He proved that these two SPRTs will lead to the same decision and showed that the expected number of events when the SPRT stops is $E(N|\lambda_1) = E[x(t)|\lambda_1] + L(\lambda_1)$, where $L(\lambda_1)$ is the operating characteristic function of the test H_1 [13]. As Haggstrom has detailed every step of computing $E(N|\lambda_1)$, interested readers please refer to that report [13].

4 Elimination System

An effective approach to remove SSFW is to shape a flow's throughput. Although it is easy to regulate the *outgoing* traffic using mechanisms like Linux traffic control or Tor's bandwidth limit on relayed traffic, they can not regulate the *incoming* traffic. Therefore, if an upstream middlebox does not shape outgoing traffic, a downstream host will receive watermarked traffic.

Public proxies and one-hop anonymity network like Anonymizer usually do not apply traffic shaping to avoid performance degradation. Although Tor uses a windowing scheme for each circuit to prevent congestion [6, 22], it could not eliminate SSFW because it only limits the high throughput instead of removing the low-through periods. Though long delays introduced by the Tor network may affect SSFW's decoding rate, new mechanisms for increasing the performance of Tor network [23] will mitigate the noise to SSFW's encoding/decoding procedure. Therefore, end users need solutions to remove SSFW by themselves.

Being a complement to existing traffic control mechanism, our elimination system allows a middlebox or an end user to shape the throughput of *incoming* traffic. More precisely, our system modifies the advertising window in outgoing packets and adds additional delays if necessary. We employ the *leaky bucket* algorithm to determine the throughput of incoming traffic [25].

Let S_{pkt} and N_{pkt} denote the packet size and the number of packets received in a T_{rtt}. N_{pkt} is controlled by the available data in the TCP sender, its congestion window (i.e. `cwnd`) and the advertising window (i.e. `rwnd`) announced by the receiver. The instantaneous throughput is equal to $\frac{S_{pkt} \times N_{pkt}}{T_{rtt}}$. To scrub SSFW, we manipulate N_{pkt} and introduce additional delay, named T_{dly}. The throughput becomes $\frac{S_{pkt} \times \widetilde{N}_{pkt}}{T_{rtt}+T_{dly}}$, where $\widetilde{N}_{pkt}$ is the number of packets received during the period of $T_{rtt} + T_{dly}$. More precisely, whenever a packet is going to be sent, the elimination system delays it for T_{dly} and checks whether there is enough quota, denoted as `BukCap`, to *receive* a packet of S_{pkt} bytes from the upstream host. If so, our system changes the advertising window in that packet to S_{pkt}. Otherwise, the packet's advertising window will be set to 0 (or 1 for unpatched windows Vista/2003/2008 that do not handle zero window correctly [5]). If a packet of size S_{pkt} is received, `BukCap` is decreased by S_{pkt}. The quota will be recovered to a pre-defined value every second. Since sometimes the TCP receiver does not has outgoing packets, our system will generate an ACK packet every 200 ms to trigger packets from the TCP sender. These ACK packets' advertised windows are set according to the above leaky bucket algorithm.

5 Evaluation

We implemented SSFW's encoder, decoder and our detection and elimination system on Linux with the help of iptables 1.4.0, the libnetfilter_queue 0.0.16 library and Linux raw socket. We realized the advanced encoding approach suggested in [28, 15]. Given a dropping probability, this approach discards packets probabilistically during a T_c period if the bit to be embedded is -1. Detailed information can be found in [17] due to limited space.

We evaluated our detection system using both the traces in [28] and the traces collected by ourselves. The traces in [28] include watermarked flows going through the Tor network. Our traces include watermarked flows going through

12 PlanetLab nodes that are located in different countries [17]. It represents the scenario of using public proxies or one-hop anonymity network like Anonymizer. Since Anonymizer does not provide free trail service now, we run a light-weight HTTP proxy, `Tiny HTTP Proxy` [1], on those PlanetLab nodes. In this case, the sender becomes a web server and the receiver downloads files from it.

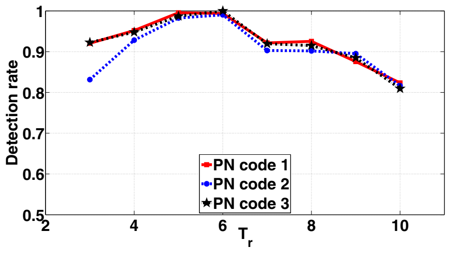
Recall that there are six parameters used in our detection system: α and β are user-defined false positive rate and false negative rate; T_r determines the minimal duration of a low-throughput period; ρ is related to the deepness of a low-throughput period; λ_0 and λ_1 are the expected normal rate and abnormal rate of low-throughput periods. In our evaluation, we fix two parameters' value (i.e. α and β), formulate the computation of another two parameters (i.e. λ_0 and λ_1) and examine the effect of the remaining two (i.e. T_r and ρ). The major reason is that only T_r and ρ affect the number of low-throughput periods, whose abnormal behavior is the basis of our detection system. We set $\alpha = 0.001$ and $\beta = 0.01$. Let Λ be the set of rates of low-throughput periods calculated from the training data. In all of our experiments, we let λ_0 be the mean value of Λ and λ_1 be the maximal value in Λ.

5.1 Evaluation of the Detection System Using Planetlab Traces

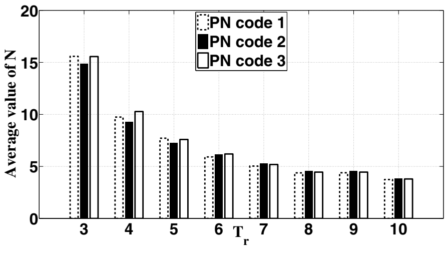
In the PlanetLab experiments, we used the same watermark (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$) in [28] and three chip durations T_c (i.e. 1s, 2s and 3s). We evaluated the detection system using three different PN codes:

1. *PN code 1*: the PN code mentioned in [28] (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$).
2. *PN code 2*: a m-sequence $\{1, -1, -1, 1, 1, 1, -1\}$ generated by [4].
3. *PN code 3*: a pair of Walsh-Hadamard code generated by [4] that spreads 1 and -1 using $\{1, -1, 1, -1, 1, -1, 1\}$ and $\{1, 1, -1, -1, 1, 1, -1\}$ respectively. They are orthogonal codes [10].

We downloaded a large file from the web server through each proxy 100 times. Half of the traces were used to compute the RTT between the client and the proxy, λ_0 and λ_1 used in the SPRT. The remaining traces were used to evaluate



(a) The detection rate when different PN codes were used



(b) The average value of N when a spread-spectrum flow watermark was detected

Fig. 5. The detection rate and the average value of N v.s. T_r

our system's false positive. Then we downloaded the same file 150 times through each proxy and the encoder started embedding the watermark to the flows.

We first examined the effect of different PN codes and that of T_r on the detection rate and false positive. In these experiments, we let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$. Figure 5(a) illustrates the impact of T_r on the detection rate. It is worth noting that when T_r changes the number of low-throughput periods in all flows may also vary. Consequently, Λ and λ_1 may also change. We found that when T_r increases from a small value 3 to a large value 10 the false positive remains *zero* while the detection rate first raises and then decreases.

The low false positive rate may result from the fact that we let $\lambda_1 = \max(\Lambda)$. The reason for the trend of detection rate is two-fold. On the one hand, using a small T_r may include many short low-throughput periods in both normal flows and watermarked flows. While the difference between a normal flow and a watermarked flow is that the latter has more *long* low-throughput periods, a large number of short low-throughput periods may obscure this feature and cause more false alarms. On the other hand, when T_r is too large (e.g. much larger than T_c) the low-throughput periods caused by SSFW with small chip duration may be ignored and therefore the detection rate decreases. It is rational to let T_r be 4 or 5 because from a decoder's point of view T_c should be larger than several RTTs to mitigate the noise from ordinary network congestion, where a TCP sender needs a few RTTs to recover its throughput from a mild network congestion. Such recovery process may lead to low-throughput periods similar to the effect of embedding flow watermark. Therefore if T_c is shorter than such recovery process, ordinary network congestion may give rise to decoding errors.

Figure 5(b) illustrates the average number of low-throughput periods needed by our system to raise an alarm. As T_r increases, less number of low-throughput periods is needed because the number of low-throughput periods in both normal flows and watermarked flows decreases. Since each flow carried only one watermark in our experiments, the decoder needs to observe the whole flow before recovering the watermark. Therefore, if a watermarked flow is detected before the end of the flow, the watermark must be identified before the decoder has the chance to recover the watermark. All detected flows in our experiments were identified before the end of each flow. A conservative approach is to let the middlebox shape the outgoing traffic when it uses our detection system to determine the existence of SSFW in incoming traffic.

When examining the effect of different PN codes, we observed from Figure 5(a) that the result of the *PN code 1* and that of the *PN code 3* are similar. It may be due to the similarity in their run properties. To encode the watermark (i.e. $\{1, -1, 1, 1, -1, 1, -1\}$) , *PN code 1* has 18 runs of -1 including twelve 1-bit runs and six 2-bit runs. *PN code 3* also has 18 runs of -1 including fifteen 1-bit runs and three 2-bit runs. All these runs of -1 will cause low-throughput periods. In comparison with them, *PN code 2* has only 11 runs of -1 including one 1-bit run, seven 2-bit runs and three 3-bit runs. However, Figure 5(b) illustrates that to detect these watermarks the number of required low-throughput period for different PN codes is similar.

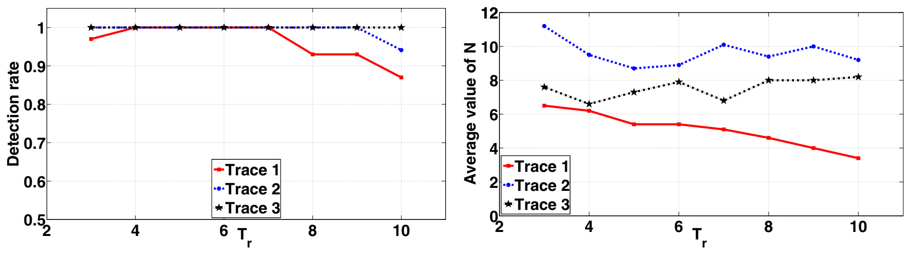
Table 1. Average detection rate (R_d) and false positive rate v.s. K ($\rho = K\frac{\sigma_{\Pi}}{\mu_{\Pi}}$)

	PN Code 1		PN Code 2		PN Code 3		
K	R_d	N	R_d	N	R_d	N	False Positive
1	0.996	7.71	0.983	7.26	0.987	7.59	0
2	0.939	5.53	0.908	5.53	0.935	5.76	0
3	0.94	4.08	0.91	4.26	0.934	4.3	0.003
4	0.781	4.1	0.751	4.36	0.795	4.28	0.006
5	0.74	3.82	0.655	3.99	0.711	4.01	0.006

To evaluate the effect of ρ, we let $T_r = 5$ and $\rho = K\frac{\sigma_{\Pi}}{\mu_{\Pi}}$ and then increase K from 1 to 5. The experiment result is listed in Table 1, which includes the detection rate (R_d) and the average number of N for different PN codes and the false positive rate. We can see that the best detection rate is achieved when $K = 1$. This observation is in accordance with the analysis in section 3.3. When K increases, the detection rate decreases. The reason is that a large ρ may filter out many low-throughput periods caused by the flow watermarks, for example, those generated by small dropping probability (e.g. 0.1). However, when K is not larger than 3, the detection rate is still larger than 90%. Moreover, a very large ρ may also increase the false positive rate because it may lead to a very small λ_0 (i.e. most normal flows do not have such kind of low-throughput periods) and flows with a few throughput outliers might cause false positive.

5.2 Evaluation of the Detection System Using Tor Traces from [28]

In [28]'s Tor experiments, the encoder used 98 seconds to embed a watermark because the chip duration is 2 seconds and both the m-sequence and the watermark have 7 bits (i.e. 2*7*7=98). After that, their encoder will wait for 98 seconds before embeding another watermark to the same flow [28].



(a) The detection rate in different traces (b) The average value of N when a spread-spectrum flow watermark was detected

Fig. 6. The detection rate and the average value of N v.s. T_r

The author of [28] provided us three traces that contain throughput aggregated in 0.1s. We let the basic unit time be 0.1s because we could not know the exact RTT from the traces. For each trace, we first identified normal periods and watermark periods and then divided the normal periods into two groups. Based on the data in the first group of normal periods, we computed λ_0, λ_1, σ_Π and μ_Π. Then, we applied the detection algorithms to the data in the second group of normal periods to calculate the false positive rate and applied the detection algorithms to data in watermark periods to compute the detection rate.

Figure 6(a) shows the detection rate in different traces when T_r varies. In most cases our detection scheme can identify all watermarks. We observed that when T_r increases from a small value 3 to a large value 10 the false positive remains *zero*. Figure 6(b) illustrates the average number of low-throughput periods needed by our detection system to raise an alarm. Compared to the PlanetLab experiment results shown in Figure 5(b), the number of steps needed to detect watermarked flows going through Tor is relatively stable. It is because λ_0 and λ_1 computed from normal periods in those Tor traces are less sensitive to T_r. In these experiments, we let $\rho = \frac{\sigma_\Pi}{\mu_\Pi}$. When fixing T_r to 5, we still get *zero* false positive when $\rho = 2\frac{\sigma_\Pi}{\mu_\Pi}$.

5.3 Evaluation of the Elimination System

In [28], a flow is deemed as being watermarked if and only if the decoder can recover the 7-bit watermark. The experiment results showed that our elimination system can successfully remove SSFW from flows going through both one-hop proxies and Tor network. To explain the result clearly, we define the bit decoding rate as the ratio of the number of bits decoded correctly to the length of a watermark. A watermark is removed if the bit decoding rate is less than 1.

Table 2. Bit decoding rate and throughput ratio after a watermarked flow is processed by our elimination system

Dropping probability	0.1	0.2	0.4	0.6
Average bit decoding rate	0.651	0.51	0.5	0.571
Throughput ratio	0.771	0.672	0.536	0.424

Since a SSFW encoder degrades the throughput of the target TCP flow, the major goal of our elimination system is to remove SSFW and at the same time minimize the negative effect on its throughput. Table 2 lists the average bit decoding rate and throughput ratio obtained from the flows between a PlanetLab node in UK (194.36.10.154) and the client when the encoder adopted different dropping probability. The throughput ratio indicates the ratio of the average throughput of watermarked flows that were scrubbed by our elimination system to the average throughput of watermarked flows. In these experiments, we set the regulated throughput to the median of the throughput of watermarked flows. The experiment results showed that *all* watermarks were removed. When the dropping probability is small, the throughput degradation is around 23%. When the dropping probability increases, the throughput degradation becomes severe. The reason is that under high dropping probability the modulated TCP's throughput is already very low and consequently its median value (i.e. the regulated throughput) is very small.

We also applied our elimination system to flows going through the Tor network. Since Tor changed the paths during the experiments, we set the regulated throughput to fix values. Figure 7 shows a box plot of the bit decoding rates after our elimination system regulated the incoming traffic from Tor's entry node. The dot within each box indicates the median value of bit decoding rate. We can see that *all* watermarks were removed.

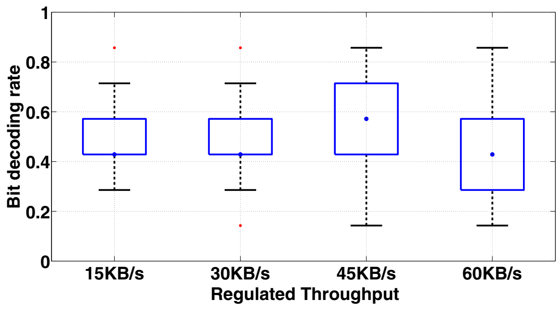


Fig. 7. Bit decoding rate v.s. regulated throughput

6 Conclusion

In this paper we proposed a novel method to detect spread-spectrum flow watermarks (SSFW). Our method leverages the intrinsic features of SSFW, and is mainly based on detecting anomalous sequences of low-throughput periods in network flows. Furthermore, we introduced a novel receiver-based approach to remove SSFW by leveraging TCP's flow control mechanism. This approach is complementary to router-based traffic shaping methods, and allows end users to autonomously remove SSFW even in case of non-cooperative middleboxes. We conducted an extensive evaluation of our watermark detection and elimination systems. The experimental results confirm that our detection approach is able to identify SSFW quickly and with high accuracy, and that our elimination system can effectively remove SSFW from network flows. In the further work, we will investigate how to mitigate the throughput degradation caused by the elimination system. Another possible direction is to select more suitable carriers and watermarks for the design of flow watermarks [3].

Acknowledgments

We thank Xinwen Fu and Ling Zhen for giving us Tor traces and suggestions on Tor's experiments. This material is based upon work supported in part by the National Science Foundation under grants no. 0716570 and 0831300, the Department of Homeland Security under contract no. FA8750-08-2-0141, the Office of Naval Research under grant no. N000140911042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Homeland Security, or the Office of Naval Research.

References

[1] hisao, S. (2009), `http://www.okisoft.co.jp/esc/python/proxy`

[2] Altman, E., Avrachenkov, K., Barakat, C.: A stochastic model for tcp with stationary random losses. In: ACM SIGCOMM (2000)

[3] Cayre, F., Fontaine, C., Furon, T.: Watermarking security: Theory and practice. IEEE Transactions on Signal Processing 53(10), 3976–3987 (2005)

[4] Choi, B.: PN code generator (2000), `http://www-mobile.ecs.soton.ac.uk/bjc97r/pnseq-1.1/pnseq-1.1.tar.gz`

[5] Microsoft Corporation. Microsoft security bulletin ms09-048 (2009), `http://www.microsoft.com/technet/security/Bulletin/ms09-048.mspx`

[6] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX SEC (2004)

[7] Dixon, R.: Spread Spectrum Systems, 2nd edn. John Wiley & Sons, Chichester (1984)

[8] Donoho, D., Flesia, A., Shankar, U., Paxson, V., Coit, J., Staniford, S.: Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516, p. 17. Springer, Heidelberg (2002)

[9] Dvoretzky, A., Kiefer, J., Wolfowitz, J.: Sequential decision problems for processes with continuous time parameter testing hypotheses. Annals of Mathematical Statistics 24 (1953)

[10] Fazel, K., Kaiser, S.: Multi-Carrier and Spread Spectrum Systems. Wiley, Chichester (2003)

[11] Gibbons, J., Chakraborti, S.: Nonparametric Statistical Inference, 4th edn. CRC, Boca Raton (2003)

[12] Golomb, S.: Shift Register Sequences (revised edition). Aegean Park Press, Laguna Hills (1982)

[13] Haggstrom, G.: Sequential tests for exponential populations and poisson processes. Technical report, RAND Corporation (1979)

[14] Houmansadr, A., Kiyavash, N., Borisov, N.: Multi-flow attack resistant watermarks for network flows. In: IEEE ICASSP (2009)

[15] Jia, W., Tso, F., Ling, Z., Fu, X., Xuan, D., Yu, W.: Blind detection of spread spectrum flow watermarks. In: IEEE INFOCOM (2009)

[16] Kiyavash, N., HoumanSadr, A., Borisov, N.: Multi-flow attacks against network flow watermarking schemes. In: USENIX Security (2008)

[17] Luo, X., Zhang, J., Perdisci, R., Lee, W.: On the secrecy of spread-spectrum flow watermarks (2010), `http://roberto.perdisci.com/publications/publication-files/DSSSWM_Extended_TechReport.pdf`

[18] Markopoulou, A., Tobagi, F., Karam, M.: Loss and delay measurements of internet backbones. Computer communications (June 2006)

[19] Peng, P., Ning, P., Reeves, D.: On the secrecy of timing-based active watermarking trace-back techniques. In: IEEE Symp. on Security and Privacy (2006)

[20] Pyun, Y., Park, Y., Wang, X., Reeves, D., Ning, P.: Tracing traffic through intermediate hosts that repacketize flows. In: IEEE INFOCOM (2007)

[21] Ramsbrock, D., Wang, X., Jiang, X.: A first step towards live botmaster traceback. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 59–77. Springer, Heidelberg (2008)

[22] Reardon, J., Goldberg, I.: Improving tor using a TCP-over-DTLS tunnel. In: USENIX Security (2009)
[23] Tang, C., Goldberg, I.: An improved algorithm for Tor circuit scheduling. Technical report, University of Waterloo (2010)
[24] Therrien, C., Tummala, M.: Probability for Electrical and Computer Engineers. CRC, Boca Raton (2004)
[25] Turner, J.: New directions in communications (or which way to the information age?). In: IEEE Commun. Magazine (1986)
[26] Wang, X., Chen, S., Jajodia, S.: Tracking anonymous peer-to-peer voip calls on the internet. In: ACM CCS (2005)
[27] Wang, X., Chen, S., Jajodia, S.: Network flow watermarking attack on low-latency anonymous communication systems. In: IEEE Symp. on Security and Privacy (2007)
[28] Yu, W., Fu, X., Graham, S., Xuan, D., Zhao, W.: DSSS-based flow marking technique for invisible traceback. In: IEEE Symp. on Security and Privacy (2007)
[29] Zhang, Y., Paxson, V.: Detecting stepping stones. In: USENIX Security (2000)