DROPWAT: An Invisible Network Flow Watermark for Data Exfiltration Traceback

Alfonso Iacovazzi, Sanat Sarda, Daniel Frassinelli, and Yuval Elovici, *Member, IEEE*

***Abstract*— Network flow watermarking techniques have been proposed during the last ten years as an approach to trace network flows for intrusion detection purposes. These techniques aim to impress a hidden signature on a traffic flow. A central property of network flow watermarking is invisibility, i.e., the ability to go unidentified by an unauthorized third party. Although widely sought after, the development of an invisible watermark is a challenging task that has not yet been accomplished. In this paper, we take a step forward in addressing the invisibility problem with DROPWAT, an active network flow watermarking technique developed for tracing Internet flows directed to the staging server that is the final destination in a data exfiltration attack, even in the presence of several intermediate stepping stones or with an anonymous network. DROPWAT is a timing-based technique that indirectly modifies interpacket delays by exploiting the network's reaction to packet loss. We empirically demonstrate that the watermark embedded by means of DROPWAT is invisible to a third party observing the watermarked traffic. We also validate DROPWAT and analyze its performance in a controlled experimental framework with a series of experiments on the Internet, using Web proxy servers as stepping stones executed on several instances in Amazon Web Services; the experiments are also conducted using the TOR anonymous network in place of the stepping stones. Our results show that the detection algorithm is able to identify an embedded watermark, achieving over 95% accuracy while being invisible.**

***Index Terms*— Watermarking, traffic analysis, data exfiltration, advanced persistent threat, traceback, network monitoring.**

I. INTRODUCTION

ADVANCED persistent threats (APTs) have received an increasing amount of attention from authorities and companies in recent years. APTs refer primarily to the high-risk threats associated with unauthorized access to a network, with the primary aim of stealing highly sensitive and valuable information. Behind every APT there usually is an adversary with specific objectives that fall into the following categories: political [1], economic [2], technical [3], and military [4] purposes. Although APTs are difficult to generalize, because each attack is focused on a specific target and designed accordingly, the process of implementing an APT can be broken down into six main stages which have been well described by Giura and Wang [5]: reconnaissance, delivery, exploitation, operation, data collection, and data exfiltration. Each step in this process merits specific attention; however, in this paper we focus on the data exfiltration stage.

Data exfiltration is the last stage of an APT, and its achievement represents a successful conclusion to the entire attack process. The term data exfiltration refers to the physical process aimed at transferring previously collected sensitive data from a private device/network to an external staging server under the control of an adversary. Data exfiltration has been widely investigated [6], [7], and much attention has been focused on developing solutions that may prevent data exfiltration, detect a data exfiltration attack, and even nip it in the bud, before data has been stolen [8]. In contrast, the research community has put less effort into developing technical solutions for attack attribution, i.e., solutions aimed at real-time identification of the adversary (individual or machine) that is attempting to obtain valuable data.

Increasingly, the process of data exfiltration is taking place via the Internet by means of digital communication between a device containing the sensitive data and the remote staging server. The adversary managing this data transfer often forwards the communication over a chain of proxy servers or an anonymous network; this is done in order to prevent others from tracing the devices under the control of the adversary (by reading the destination addresses) back to the adversary, particularly when traffic flow interception has occurred.

Identifying the final destination of a data flow is a difficult problem, which is often referred to in the literature as the "network traceback problem" [9]. Network flow watermarking is a promising solution that has provided interesting insights during the last few years. Typically, watermarking solutions aim to actively modify traffic features so that they can be easily identified by a detection system, even when several noisy network nodes are crossed. Although much progress has been made in this area, two important issues remain unresolved: robustness and invisibility. Robustness refers to the property of the watermark's resistance to active noise added by an attacker to alter the watermark carrier. Invisibility is the property of the watermark to go undetected by the adversary. Invisibility is critical, because any kind of traffic feature manipulation has the potential to be easily identified by a third party (using traffic analysis instruments).

Manuscript received May 26, 2017; revised September 27, 2017; accepted November 21, 2017. Date of publication December 1, 2017; date of current version January 29, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Tomas Pevny. *(Corresponding author: Alfonso Iacovazzi.)*

A. Iacovazzi, S. Sarda, and D. Frassinelli are with Temasek Laboratories, Singapore University of Technology and Design, Singapore 487372 (e-mail: alfonso_iacovazzi@sutd.edu.sg).

Y. Elovici is with the Department of Software and Information Systems Engineering, Cyber Security Research Center, Ben-Gurion University of the Negev, Beer-Sheva 8410501, Israel, and also with the iTrust–Centre for Research in Cyber Security, Singapore University of Technology and Design, Singapore 487372 (e-mail: yuval_elovici@sutd.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2779113

In this paper we propose DROPWAT, an invisible network flow watermarking technique for data exfiltration attacks, enabling the identification of the staging server that receives the exfiltrated data. DROPWAT is based on a completely new paradigm of injecting a watermark into the flow. The basic idea of our algorithm is to drop a few selected packets of a flow in order to alter the interpacket delay. We show that: (i) packet drop events can be identified, even in the presence of several stepping stones, and that they can be used as a way to convoy a watermark into traffic flows, (ii) natural packet loss and intentional packet drop events in the network cannot be distinguished from each other, and (iii) the watermark embedded with our technique is invisible under some assumptions. We evaluate DROPWAT under different network scenarios with different conditions of packet loss and throughput on real traffic on the Internet.

The rest of the paper is organized as follows. Section II provides an overview of previous work on network flow watermarking and its application in overcoming the traceback problem. The attack scenario and reference architecture are described in Section III. The DROPWAT embedding and detection algorithms are described in Section IV. Section V contains an in depth discussion and analysis of the invisibility property. In Section VI we briefly discuss the issues related to the watermark's robustness. Section VII provides a description of our experimental results and validation of the effectiveness of DROPWAT. In Section VIII we discuss some critical aspects of our watermarking algorithm, and our conclusions are in Section IX.

II. RELATED WORK

The traceback problem, aimed at identifying the real destination of a traffic flow, has been extensively investigated [10]–[16]. In 2001, Wang *et al.* introduced network flow watermarking as a possible means of overcoming the traceback problem [17]. Since then, many network flow watermarking algorithms have been developed and proposed. Recently, Mazurczyk *et al.* [18] and Iacovazzi *et al.* [19] presented surveys providing a comprehensive analysis and comparison of the main network flow watermarking solutions known in the literature.

The vast majority of the proposed techniques modify the packet timings in order to impress a specific timing pattern onto the network flows [20]–[27]. RAINBOW is an example of a timing-based watermarking algorithm [23], where each packet is delayed by a computed value; the delay values equal the output of a cumulative function which randomly evolves with a step of plus/minus a specified watermark amplitude per packet. RAINBOW's detection algorithm is based on the comparison between the interpacket delays (IPDs) of the flow before being watermarked and those of the flows intercepted by the detector.

The technique proposed by Peng *et al.* [21] is also based on IPDs. The authors consider two groups of randomly selected pairs of consecutive packets; the IPDs are computed for every pair in each group. The two average values of IPDs in the two groups are considered statistically equal to each other. Their proposed watermarking algorithm aims to slightly modify the IPDs, so that the difference between the two average values is not zero. The numerosity of the two groups represents a kind of redundancy and determines the reliability of detection.

A technique called interval centroid-based watermarking was introduced by Wang [24] in 2007. In this technique, the time axis is divided into intervals of fixed duration T. A centroid is computed for each interval as the average value of the remainders remaining after dividing the timestamps of packets observed in that interval by T. In the embedding algorithm, some packets of the flow are delayed so that the statistical balances among groups of intervals are altered. Watermark detection is based on the statistical analysis of interval centroids. A variety of similar methods have also been suggested by other researchers [26], [28], [29].

In interval packet counting-based techniques, the time axis is divided within intervals [22], [27], [30]; the number of packets in each interval is the carrier of the watermark, and some packets of the flow are delayed in order to alter the statistical balance of the packet counting per interval.

Timing-based algorithms are very attractive, because packet timing can easily be modified by the watermarker without having to access the data at any protocol level. Nevertheless, timing can also be altered by natural network perturbation or be artificially modified by an attacker, resulting in the failure of watermark detection. For this reason, other watermarking algorithms have been created that are robust against timing perturbation [20], [21], repacketization [22], and chaff packet injection attacks [21], [23].

One major drawback of timing-based schemes is that they primarily target flows with less than 50 packets per second (PPS). If, for example, we consider a scenario such as an illegal data transfer in which the transfer rate can easily be 200 – 500 PPS or more (assuming 1500 bytes/packet, and a speed of 300 – 750 KB/s), these algorithms would not be effective. The reason for this is that most of the parameters have to be re-adapted in order to cope with the higher network speed and lower IPD. However, at higher network speed, proxy servers tend to obfuscate any kind of slight timing perturbation, making small changes impossible to detect. One could argue for the use of more significant perturbations, but this would make the watermark more visible and significantly impact the performance of the network (and not necessarily improve the detection rate, since generally the parameters need to be chosen proportionally to the IPD). The only technique that would be effective with bulk traffic is the centroid-based solution developed by Wang [24], however this technique also requires a lot of buffering and TCP level multi-flow analysis which makes its implementation impractical in scenarios in which network speed, memory, and computational power are strictly constraining (e.g., in a border router).

Timing is not the only feature that can be used as a watermark carrier; packet size [31]–[33] and bit rate [34], [35] are two traffic features that have attracted attention as well. However, size-based watermarks need to be embedded directly at the source of the traffic flow, while rate-based watermarks are highly visible to third parties.

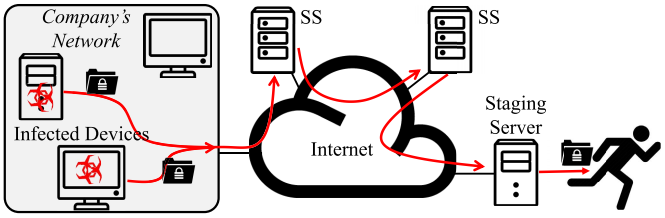

Fig. 1. Attack scenario.

Invisibility (the capability of passing unnoticed by an attacker) is one of the most important properties of a watermark algorithm. Although some researchers have designed watermarking algorithms that were claimed to be invisible [23], [35], [36], later studies have empirically shown that a completely invisible watermark does not exist yet [37]–[41].

III. Attack Scenario

A. Data Exfiltration Attack

We consider the scenario shown in Figure 1 in which an adversary wants to take possession of confidential data, files, or documents that belong to a person or company and are stored in digital format on a device connected to the Internet in some way. These documents can be sensitive, private, copyrighted, or accessible only with required permission. In our scenario, the attacker has managed to install a malware on the targeted device. This malware allows the attacker to control the device and exfiltrate data from the private network to an external server (staging server) under her control, via an Internet connection. Two or more stepping stones are used in order to disallow possible identification of the staging server (its IP address, IP address geolocation, etc.). Once the targeted data is saved on the staging server, the attacker is able to access the data at any time. If the staging server is identified, the attacker may be identified as well, when it connects to the server.

B. Stepping Stones

A stepping stone (SS), also referred to as a proxy server, is an intermediary device or application interposed in the communication between two hosts in a network. The main purpose of an SS is to prevent the identification of the real sender and/or recipient of the exchanged messages in the event that a third party intercepts the communication. The property of a flow to not be associated with the communication's real endpoints is known as the "unlinkability" of the sender and receiver. In this case, whenever a client wishes to contact a server for Web content, it does not send messages directly to the server, but instead it connects and sends the messages to a proxy server which is responsible for forwarding the traffic to the real recipient. Conversely, reply messages from the server to the client will first be delivered to the SS and then be forwarded to the client. In most cases, communications to and from an SS are based on encrypted and authenticated connections. Thus, the integrity of the unlinkability property is preserved when a third party observes the traffic in the middle of one of the two connections involved; nevertheless, the communication is vulnerable to passive attacks performed on the proxy server. A single point of vulnerability can be avoided by using two or more SSs in a chain.

1) Implementation and Packet Loss Propagation: There are many types of SSs and ways of implementing them: Web proxy servers, TOR software, etc. [42]. An explanation of different SS implementations and a description of their operations are not within the scope of this paper; we prefer to focus on how the implementation of an SS may influence traffic patterns in cases in which a packet loss occurs before reaching the SS. In these cases, the SS can behave as the propagator or retriever of lost packets. The SS behavior depends on the combination of two factors: (i) the protocols used for transferring the traffic, and (ii) the protocol layer at which the SS operates. For example, let us consider communication over TCP: when the SS handles data units at the transport layer, two independent TCP connections are established, one from the client to the SS, and the other from the SS to the server; when a packet directed to the SS is lost, the SS notices that a packet is missing and requests retransmission, so the loss is not propagated. Thus, here the SS acts as a retriever. Alternatively, an SS can also be implemented to work at the network layer (such as an NAT service). In this case, the source and destination of transport layer segments retain the real communication's source and destination. Here the SS is only responsible for being an intermediary at the network layer. The two endpoints send their IP packets to the SS which decapsulates transport segments from packets, makes port translation, and encapsulates each segment in a new IP packet containing the SS's IP address in the source address field and the real destination's IP address (or the next hop's IP address in case of a chain of SSs) in the destination address field. Here the SS changes the transport layer ports, but it does not interfere with the operations performed by the transport protocol which means that packet loss is propagated to the next hop in the path. Thus, in this scenario, the SS acts merely as a propagator.

In this paper we refer to an attack scenario in which SSs do not propagate packet losses, as this scenario is used by most attackers by implementing their own proxy networks or using TOR because it does not leave a trace of the real IP address of their staging server. Nevertheless, a slightly modified version of our algorithm would work in cases of SSs that propagate loss, since the recognition of the losses would become a trivial operation. Without loss of generality, hereafter we base our analysis on a scenario in which communications travel over TCP, and the SS operates at the transport layer.

IV. DropWat

In this section we describe DropWat, a network flow watermarking technique based on packet dropping, which indirectly modifies IPDs of selected packets. The basic idea of our technique is to mimic a natural network behavior, namely packet loss events caused by a single bottleneck node, and exploit it as a watermark that is identifiable despite the traffic flows crossing one or more SSs.

An attacker is not able to distinguish between naturally lost packets and those intentionally dropped, because both events

cause the same behavior in a network.[1] If the attacker is unable to distinguish between a sequence of lost packet events due to a real bottleneck node and a sequence of dropped packet events caused by an emulated bottleneck node, then the watermark will be invisible.

In the following subsections we explain what happens in our scenario when a packet loss event occurs; we then provide a detailed description of DROPWAT, our proposed watermarking method for tracing data exfiltration attacks.

A. Terminology

Throughout the paper we refer to the term "network flow watermarking" (often abbreviated as "flow watermarking" or just "watermarking") to indicate those hiding information techniques used in the traffic analysis field that actively manipulate some features of a targeted network flow in order to uniquely identify it among a set of flows. The main characteristics that differentiate network flow watermarking from other methods of information hiding (e.g., "digital watermarking," "steganography," etc.) are: (i) the information carrier is a network flow, and (ii) the hidden information's validity and detectability are strictly correlated to the flow duration.

B. Packet Loss Occurrence

Packet losses occur naturally in computer networks and are caused by several reasons, such as faulty hardware or cabling, buffer overflow due to link or node congestion, data corruption due to components with high bit error rates, packet filtering, etc. Internet protocol (IP) provides a service of best effort delivery and it does not deal with detecting and recovering lost packets. The management of packet recovery for reliable delivery is left to higher layer protocols. Recovery of lost packets can be guaranteed at the transport layer with the TCP protocol.

The behavior of an SS handling data units at the transport layer in a case of packet loss is depicted in Figure 2. The left half of the figure shows the typical TCP behavior when a packet is lost. It can be seen that the SS sends duplicated acknowledgements until it receives the expected packet. The server keeps sending subsequent packets until it realizes that a loss has occurred, and then it resends the lost packet. The amount of time that elapses before resending a lost packet depends on the TCP implementation used by the sender. When fast retransmission is adopted, a packet is sent a second time after receiving a specified number of repeated acknowledgements (usually set to three in the most commonly used TCP stack implementations). Since the TCP connection endpoint is at the SS, the TCP protocol reorganizes out of order data in the SS application layer, so that $data[11] - data[16]$ cannot be delivered until $data[10]$ is correctly received. For this reason, when a packet is lost, the SS cannot keep sending data to the next hop even though out of order packets are received by the TCP protocol. This entails that the IPD between $data[10]$ and $data[9]$ at the destination is altered and equal to a value greater than the round trip time from the server to the SS.

[1]We use the term *intentionally dropped packets* to indicate only those packets that are dropped in order to embed a watermark in the traffic flow. Packets dropped due to other causes (e.g., buffer overflow, framing error, etc.) are considered *naturally lost packets*.

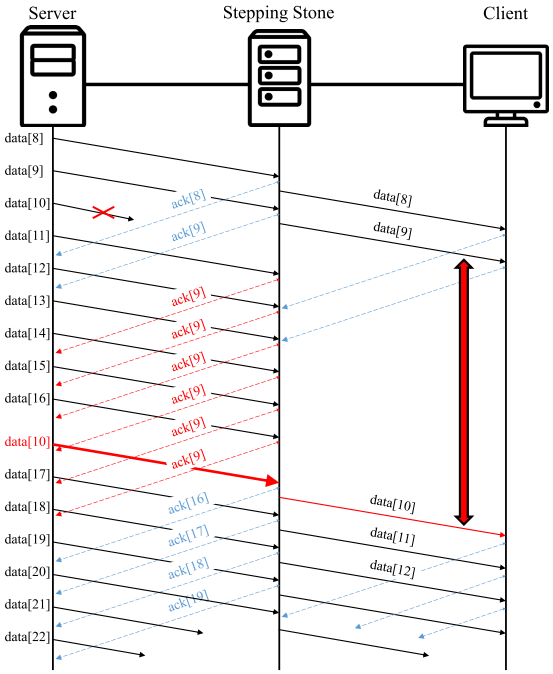

Fig. 2. Packet loss event in a scenario with one SS.

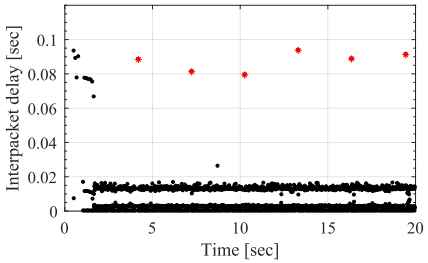

Fig. 3. The impact of packet loss events on IPDs measured on the client side.

In Figure 3 we show the trend of the IPDs measured at the client endpoint, when a 50 MB file is downloaded from the server. The communication is intermediated by two SSs. A packet was periodically dropped in the connection between the server and the first SS encountered. The round trip time (RTT) between the two was 80 ms. During the first few seconds of the communication, IPDs are affected by the TCP's slow start. After the slow start phase, the system reaches a stable state in which the IPDs maintain regular values. The regularity is broken when a packet loss event occurs, as highlighted in the figure. The trend is maintained even in the presence of multiple SSs. Thus, we can claim that although the packets are sent sequentially from the SS, any packet lost (and later retrieved) in the first connection can be identified in the second connection by analyzing IPDs on the client side.

The server packet transfer rate and the RTT between the server and the first SS may change the effect of packet loss events on IPDs. To give an idea of this effect, we averaged the values of the IPDs that correspond to the packet loss events measured on the client side, and we plotted them in Figure 4

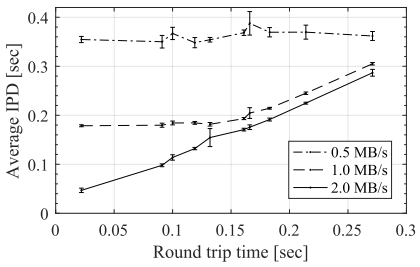

Fig. 4. Average value of the IPDs that correspond to the packet loss events plotted as a function of the RTT from the server to the SS.

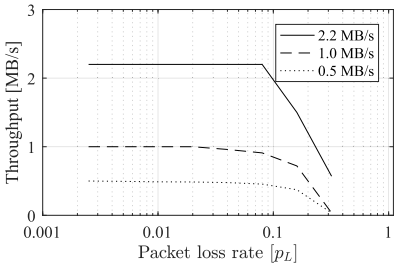

Fig. 5. Throughput as a function of packet loss rate.

Fig. 6. Architecture of the DROPWAT watermarking system.

as a function of the RTT from the server to the SS. The graph shows a linear trend when the transfer rate R is 2.2 MB/s, a constant and later linear ramp for $R = 1$ MB/s, and a constant trend for $R = 0.5$ MB/s. This is due to the fact that IPDs, altered by packet loss, are a function of both transfer rate and RTT.

When the number of lost packets in the network becomes high, the TCP protocol interprets this behavior as network congestion and reacts by reducing the rate at which packets are sent. The reduction of the throughput caused by varying the packet loss rate is shown in Figure 5. Thus, in order to ensure that the embedded watermark does not have a significant impact on the network performance, the packet loss rate should be less than 1%.

C. Watermarking Architecture

The architecture of DROPWAT is similar to other existing active network flow watermarking techniques. As shown in Figure 6, the system is composed of a watermarker and a detector. The watermarker intercepts targeted flows and embeds the watermark. In our case, this action corresponds to selectively dropping some packets. The detector observes and analyzes traffic flows, and looks for the presence of a watermark. In the following two subsections the embedding and detection algorithms are described in greater detail.

D. Watermark Embedding

DROPWAT's embedding algorithm aims at dropping pseudo-randomly selected packets so that the sequence of dropped packets looks like a loss sequence caused by a single bottleneck node. A single bottleneck node can be described as a buffer which can hold a specific number of packets. An input process fills the buffer with packets coming from several sources; an output process extracts packets from the buffer at a fixed rate limited by the output link rate. When an incoming packet finds the buffer full, it will be discarded and a loss event will occur.

In order to emulate the behavior of a single bottleneck node, we model packet loss behavior according to a modified version of the extended Gilbert model. The extended Gilbert model was used to reflect packet loss behaviors in noisy networks by Sanneck and Carle [43], and Yu *et al.* [44] demonstrated that this model approximates the packet loss behavior of a single multiplexer very well. Let X_i be the binary event for the i-th packet of a flow, which can assume the value 1 for a dropped packet and 0 for a non-dropped packet. In our modified version of the extended Gilbert model, an event state is assumed to be dependent on the last run composed of up to n consecutive identical events. In this model (hereafter referred to as $\mathcal{M}_{\mathcal{W}}$) we need only $2n$ different states, and it can be completely described by the set of probabilities $\{p_{W,k}\}_{0<|k|\leq n}$. The state model diagram of packet drops is depicted in Figure 7.

The watermarking process, as depicted in Figure 8, can be divided into two parts: offline initialization and online packet dropping. The algorithm evolves as a periodic process with time period T. Let $T_0 = 0$ be the zero time reference; we indicate the starting time of the i-th time period as $T_i = iT$.

The offline initialization takes as input: (i) the model probabilities $\{p_{W,k}\}_{0<|k|\leq n}$, (ii) a secret key shared with the watermark detector, (iii) a watermarker identifier ID_j, and (iv) the reference throughput R. The concatenation of the secret key and ID_j will be used as the seed of the dropping sequence generator (DSG), a cryptographically secure function generating a pseudo-random binary sequence (sequence of events) which follows the model $\mathcal{M}_{\mathcal{W}}$. Let $\mathbf{B}^{(i,j)} = [b_1^{(i,j)}, b_2^{(i,j)}, \ldots, b_N^{(i,j)}]$ indicate the i-th binary sequence generated by the DSG of the j-th watermarker identified by ID_j, where $N = \lceil RT/L_{ref} \rceil$ is the expected number of packets in a period, L_{ref} is the reference packet size computed as the maximum transmission unit (MTU), and $\Delta t_{pkt} = L_{ref}/R$ is the time required to send a packet. The throughput R can be set at the maximum transfer rate of the watermarker.

The DSG can be efficiently implemented by using two secure pseudo-random number generators (PRNG). The first, $prng_{syn}$, is used to synchronize the watermarker and the detector, as shown in Algorithm 1, and is initialized using the shared key $shared\_key_j = secret\_key_j | ID_j$ and the initial time T_0.

After every time period T, a new $seed = prng_{syn}$ is generated by Algorithm 1 and used to initialize a second $prng_{dsg}$. This newly created $prng_{dsg}$ is used to generate a valid binary sequence $\mathbf{B}^{(i,j)}$ of length N by executing Algorithm 2. The binary sequence is then converted to a dropping sequence.

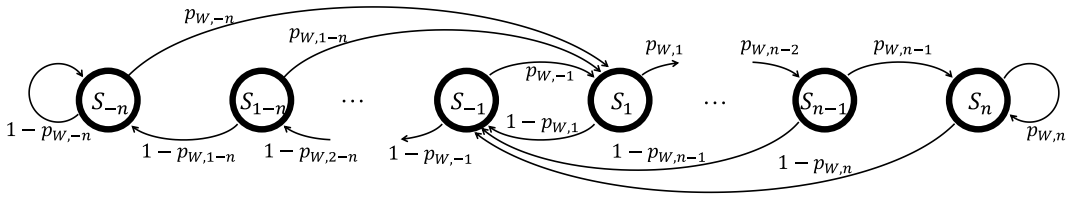

Fig. 7. Variation on the extended Gilbert model.

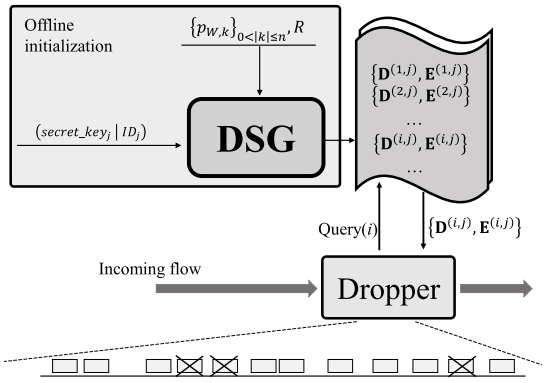

Fig. 8. DROPWAT's embedding scheme.

Algorithm 1 Synchronization

```
procedure SYNCDSG(shared_key, T_0, T)
    prng_syn ← new PRNG(shared_key)
    T_curr ← T_0
    while T_curr < system.timeNow() do
        prng_syn.genRand()   ▷ Generate a pseudo-random number in [0, 1)
        T_curr ← T_curr + T
    end while
    return prng_syn
end procedure
```

A dropping sequence corresponds to a sequence of packet dropping time intervals, and it is described by two vectors $\mathbf{D}^{(i,j)} = [d_1^{(i,j)}, d_2^{(i,j)}, \ldots, d_{K_{i,j}}^{(i,j)}]$ and $\mathbf{E}^{(i,j)} = [e_1^{(i,j)}, e_2^{(i,j)}, \ldots, e_{K_{i,j}}^{(i,j)}]$ of length $K_{i,j}$, where $d_k^{(i,j)}$ and $e_k^{(i,j)}$ indicate the starting time and the duration, respectively, for the k-th dropping time interval, expressed in nanoseconds, and $K_{i,j}$ is the number of dropping intervals in the i-th time period. The dropping sequence conversion is performed by means of Algorithm 3.

The dropper works by discarding all of the packets traversing the watermarker during any dropping time interval. All of the other packets will be correctly forwarded to the proper interface.

E. Watermark Detection

The detector is placed at one or more points in the network where we might expect to observe watermarked flows. The detector analyzes all traffic and tries to understand whether a watermark is embedded in any of the observed flows.

Algorithm 2 Binary Sequence Generation

```
procedure GENDSG(prng_syn, {p_{W,k}}_{0<|k|≤n}, N)
    seed ← prng_syn.genRand()
    prng_dsg ← new PRNG(seed)
    B ← new vector()
    k ← −n
    while B.size < N do
        if prng_dsg.genRand() < p_{W,k} then
            B.append(1)
            k = max{1, min{k + 1, n}}
        else
            B.append(0)
            k = min{−1, max{k − 1, −n}}
        end if
    end while
    return B
end procedure
```

Algorithm 3 Dropping Sequence Conversion

```
procedure DSC(B, Δt_pkt)
    D, E ← new vector()
    while k <= B.size do
        n ← 1
        if B[k] == 1 then
            n ← countOnes(k, B)   ▷ Count consecutive ones from position k in B
            D.append(k · Δt_pkt)
            E.append(n · Δt_pkt)
        end if
        k ← k + n
    end while
    return D, E
end procedure
```

The detector is aware of the input data to the DSG and the cryptographical function used by the watermarker, so it can compute all of the dropping time intervals. The detector analyzes the IPDs for packets observed during the dropping time intervals, and for each flow it builds the sequence of identified lost packets. If a significant percentage of lost packets of a flow are detected during the dropping time intervals, the flow is suspected of being watermarked.[2]

[2]Since burst losses are managed by the TCP protocol through burst retransmissions, the detector can only identify the first dropped packet of a burst. For this reason, the burstness of packet loss is not relevant to detection.

The detector and the watermarker must be accurately synchronized in order to agree on the valid dropping sequence for a time period; to maintain synchronization over a long period of time, an external synchronization server (such as NTP) may be used to reset the internal clocks of the two devices.[3]

The watermark detection algorithm can be summarized in three main steps: (i) IPD computation, (ii) outlier detection, (iii) watermark detection and identification.

- *IPD computation.* An IP flow is sniffed, and packet timestamps are measured. A nominal task, IPD computation is based on the difference between consecutive packet timestamps.
- *Outlier detection.* IPDs are analyzed to detect packet loss events. The detection is based on a simple outlier detection algorithm. Let $\hat{t}_k$ be the timestamp of the k-th packet observed by the detector, $\Delta\hat{t}_k = \hat{t}_k - \hat{t}_{k-1}$ be the k-th IPD, and υ be a comparison window size. $\Delta\hat{t}_k$ is considered an outlier if $(\alpha \cdot \Delta\hat{t}_k) > \Delta\hat{t}_h$ for all $h \in \{k-\upsilon, \ldots, k-1, k+1, \ldots, k+\upsilon\}$, with $0 < \alpha < 1$. The observation times of the outlier packets are used to compile an outlier time vector $\hat{\mathbf{D}}^{(i)} = [\hat{d}_1^{(i)}, \hat{d}_2^{(i)}, \ldots, \hat{d}_{\hat{K}_i}^{(i)}]$ where $\hat{d}_k^{(i)}$ is the observation time measured considering the start time of the current period T_i as the reference time. The outlier time vector can be compiled almost in real-time, with a delay of υ packets.
- *Watermark detection and identification.* The detector compares $\hat{\mathbf{D}}^{(i)}$ with $\mathbf{D}^{(i,j)}$ for every j, in order to find the $\kappa_{i,j}$ number of matching outliers, corresponding to the number of disjoint couples $(\hat{d}^{(i)}, d^{(i,j)})$, such that $|\hat{d}^{(i)} - d^{(i,j)}|$ is less than the matching distance δ. For a given j, if $\kappa_{i,j}/K_{i,j}$ is greater than a predefined threshold β, the flow is labelled with ID_j. Thus, a flow can be labelled with zero, one, or more than one identifiers. In a case in which no label is assigned, the flow is considered unwatermarked; otherwise the flow is considered watermarked. The watermark identification is based on the label with the greatest value $\kappa_{i,G} = \max_j\{\kappa_{i,j}\}$ among the labels assigned to the flow. If there is more than one label with the same greatest value $\kappa_{i,G}$, the watermark is considered detected but not identified.

Given a reference period, for each flow to analyze the detector requires: (i) computing $O(N)$ IPDs, (ii) making $O(\upsilon \cdot N)$ comparisons for the outlier detection, and (iii) making $O(J \cdot K^2)$ comparisons for the watermark identification.

It may happen that the detector intercepts genuine flows that have been watermarked but have not traversed any SSs. In this case, a packet loss is retrieved by the legitimate receiver, and no IPD outlier is generated due to that loss. Thus, even if a legitimate flow has been watermarked, the watermark will not be erroneously detected.

F. Analysis of False Identification

Let us consider a generic time period of duration T, and suppose that there are J different watermarkers, identified by $ID_1, ID_2, \ldots, ID_J$, which are simultaneously active in the given time period. For the sake of simplicity, let us suppose that each watermarker has K dropping intervals for the time period. Let p be the probability that a generic watermarker decides to start a dropping interval within an interval of duration δ, and p_L be the probability that there will be a natural loss within an interval of duration δ. When the detector observes a watermarked flow, the outlier vector $\hat{D} = [\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_{\hat{K}}]$ includes κ_{j^*} $(\leq \hat{K})$ outliers induced by the watermarker ID_{j^*} and $\hat{K} - \kappa_{j^*}$ outliers due to natural loss. Thus, we can have the following cases: (i) $\kappa_j < (\beta K)$ for all j – the watermark is not detected; (ii) $\kappa_{j^*} \geq (\beta K)$ and $\kappa_j < \kappa_{j^*}$ for all j with $j \neq j^*$ – the watermark is correctly detected and identified; (iii) $\kappa_{j^*} \geq (\beta K)$ and there exists j' with $j' \neq j^*$ such that $\kappa_{j'} = \kappa_{j^*}$ and $\kappa_j \leq \kappa_{j^*}$ for all j with $j \neq j^*$ – the watermark is correctly detected but not identified; and (iv) there exists j' with $j' \neq j^*$ such that $\kappa_{j'} > \kappa_{j^*}$ – the watermark is detected but incorrectly identified. The probability of true identification (TI), given a detection, (second case) is given by:

$$P_{TI|D} = \left(\sum_{\kappa=0}^{\kappa_{j^*}-1} F(\kappa)\right)^{J-1} \tag{1}$$

where

$$F(\kappa) = \binom{\hat{K}}{\kappa}\left(1-(1-p)^2\right)^{\kappa}(1-p)^{2\left(\hat{K}-\kappa\right)} \tag{2}$$

The probability of not identification (NI), given a detection, (third case) is given by:

$$P_{NI|D} = \left(\sum_{\kappa=0}^{\kappa_{j^*}} F(\kappa)\right)^{J-1} - P_{TI|D} \tag{3}$$

The probability of false identification (FI), given a detection, (fourth case) is given by:

$$P_{FI|D} = 1 - P_{TI|D} - P_{NI|D} \tag{4}$$

In Figure 9 we show the trend of $P_{FI|D}$ and $P_{NI|D}$ according to Formulas 3 and 4 where we consider $T = 300$ sec; $\delta = 0.04$ sec; $K = 10$; $p = K\delta/T$; $\hat{K} = \kappa_{j^*} + p_L T/\delta$; by varying J; for three values of loss probability ($p_L = 10^{-4}$, 10^{-3}, and 10^{-2}) and three values of κ_{j^*} (3, 4, and 5). The plots show that we can have a reasonably low probability of false identification by keeping J less than 40 when $\kappa_{j^*} = 3$, while for $\kappa_{j^*} > 3$ we can manage over 100 IDs.

G. Placing the Watermarker and the Detector Into the Internet

According to the watermarking architecture shown in Figure 6, the watermarker may be placed at any point in the communication path between the traffic source (server side) and the first SS, while the detector may be placed at any point in the communication path between the last SS and the final destination (client side). In an actual operational scenario, placing the two components is not a trivial task, especially considering that: (i) the client location is not known

[3] DROPWAT requires a resolution of the order of few milliseconds to be effective which is much less fine-grained than the resolution of an NTP server which offers a theoretical precision of up to 200 picoseconds.

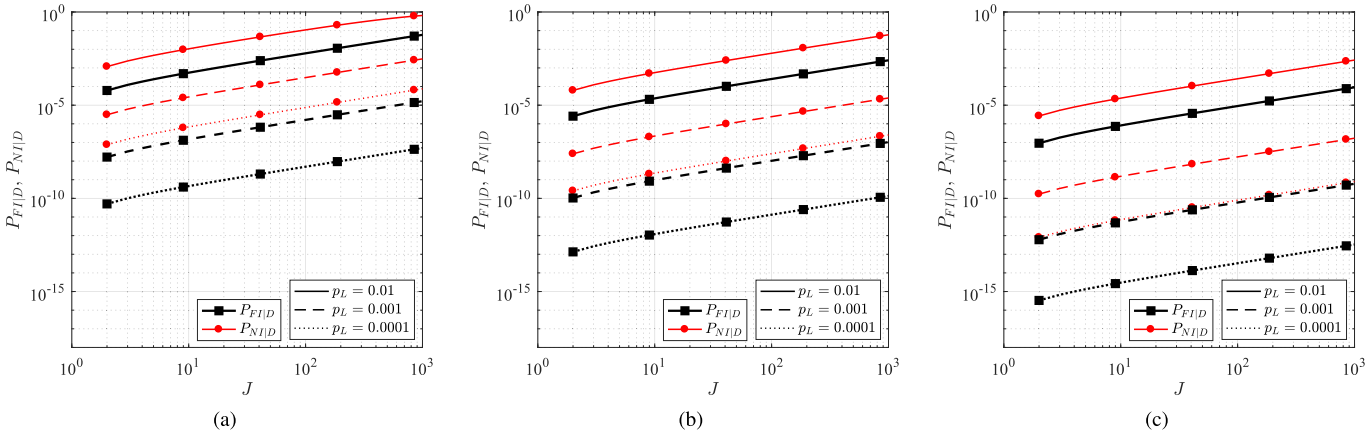

Fig. 9. Probability of false identification and not identification, given a detection. (a) $\kappa_{j^*} = 3$. (b) $\kappa_{j^*} = 4$. (c) $\kappa_{j^*} = 5$

(it is what we want to detect); (ii) the two communication paths are usually decoupled, and it is not very likely that a single entity is able to have access to both communications; and (iii) the two components should be controlled by the same entity, or at least they should be controlled by two different entities that trust each other and collaborate by sharing the watermarking information (e.g., keys, IDs, algorithms, etc.). This argument is broad, and the related discussion may involve many aspects regarding to the resources available to the monitoring entity, the objectives, etc. Although it is not within the scope of this paper to investigate all of the potential use case scenarios along with the related issues and solutions, we provide a number of possible real-life situations in which the watermarking system may be successfully employed.

First we consider a scenario in which a law enforcement agency suspects that there are known subjects operating on the Internet that are interested in exfiltrating sensitive data stored in a secured database. The agency may create a honeypot that looks like the secured database and make it vulnerable to a set of attacks. In this case, the watermaker can be integrated in the honeypot or placed in the communication path from the honeypot to the Internet. Because the agency knows the identity of the suspected subjects, it may establish a partnership with one or a few ISPs and identify the best locations to place the watermark detectors in their network(s) and observe some of the traffic that reaches the suspected subjects. If the agency suspects that an attacker might use a cloud service to store the exfiltrated data, the agency may decide to establish a partnership with the cloud service provider and define watermark detection policies for the traffic reaching the cloud.

In today's cloud era, an enormous number of services are virtualized and offered by cloud service providers. Considering their widespread usage, the probability that the source and destination of a traffic flow are within the same platform is not negligible. Therefore, a single cloud service provider may decide to apply strategies of selective watermarking with monitoring and detection on its own network.

Another use case is a scenario in which an attacker uses the TOR network (or any other anonymous network) to hide its communication with the victim; in this case the attacker does not usually have control of traversed SSs. If the entity embedding the watermark (e.g., a potential fake victim) also controls a number of TOR relays, it can place a detector on each of them; then it is sufficient that one of these relays is selected as exit node of the TOR circuit, and the detector would be able to detect the watermark and link it with the real destination of the watermarked flow. The probability that a user can select a relay controlled by a monitoring entity has been studied by Ling *et al.* [45].

V. Invisibility

A watermark should go unidentified by the adversary, because otherwise the adversary could take some action to prevent the staging server from being detected, for example, by interrupting the communication or in some way preventing the adversary from connecting to the staging server so as not to be identified. In the next subsections, we provide our assumptions about the adversary against the watermark's invisibility and define the statistical loss-invisibility. We also provide empirical evidence regarding DropWat's loss-invisibility. At the end, we show that the assumptions can be relaxed in order to make the watermark implementation easier.

A. Assumptions About the Adversary and Her Capabilities

We base our analysis of the invisibility on a set of assumptions about the adversary and her capabilities: (i) the adversary cannot directly access the hardware/software of the watermarker device, (ii) the adversary cannot distinguish between a naturally lost packet and one intentionally dropped when single events are observed independently; (iii) the adversary can passively observe the traffic passing through the watermarker (the traffic can also be actively injected by the adversary) and extract the sequence of loss events; (iv) the adversary cannot perform side-channel attacks.

The assumptions above are motivated by two main considerations. First, all of the successful attempts to detect malicious packet losses were based on methods that exploit expected (based on statistics) behaviors and patterns of the network traffic (e.g., packets dropped by an intrusion detection system or port blocked by a firewall) [46], [47]. Second, the watermarker embeds information into flows by modulating

the sequence of packet loss events, and that sequence is the only information carrier analyzed by the detector. If, for some reason, the embedding process produces a side-channel that may reveal the presence of the watermark, this side-channel should be taken into account and eventually removed.

Some examples of side-channel attacks that could be conducted against DROPWAT are: (i) packet timing channels that can be used by the adversary to infer the internal state of the watermarker's buffers and recognize inconsistencies between the loss and forwarding delay; this side-channel can be removed by means of an implementation reproducing the buffer timing effect on all of the packets passing through the watermarker or by using real buffers inside the watermarker, etc.; (ii) processing timing channels used to distinguish among several operations involved during the normal handling of the traversing packets; this side-channel can be removed by executing sensitive operations on a timely planned manner. These and similar types of side-channel attacks are largely implementation dependent and strictly correlated to the device used and its forwarding policies. Investigating their impact and possible countermeasures is not within the scope of this paper.

B. Loss-Invisibility

We start from the definition of statistical invisibility provided by Iacovazzi [19] that says that "a watermark is statistically invisible if the difference between the statistical distribution of a watermarked flow and a non-watermarked flow is negligible." Considering that the only modification made by the watermarker is dropping packets, the definition of invisibility can be adapted to our specific case; thus for this reason we define and refer to the *statistical loss-invisibility*.

Hereafter, we refer to a *network component* as any entity in a network that can be traversed by Internet packets along their path. A component can be a router, firewall, network segment delimited by two nodes, a demilitarized zone, etc. For the sake of simplicity, we only consider dual-homed components, i.e., network components having two interfaces, in which packets entering one interface will be released out of the second interface, unless they have been lost inside the component.

We say that two different network components are *statistically loss-equivalent* if the statistical distance between the two statistical units associated with the packet loss sequence induced by the two components is negligible.[4]

Let B be a component in a private network described by the packet loss statistical model $\mathcal{M}_{\mathcal{B}}$. Suppose that B can be substituted by a new component S which consists of the concatenation of two sub-components: a sub-component $\hat{B}$ performing the same functionalities of B but with a lower level of packet loss, which is statistically described by the loss model $\mathcal{M}_{\hat{\mathcal{B}}}$; and a loss-based watermarker W which drops packets according to a statistical loss model $\mathcal{M}_{\mathcal{W}}$. The watermark impressed by W is *statistically loss-invisible* if the new component S is statistically loss-equivalent to the component B.

[4]The statistical distance is a metric that is defined in statistics and measures the distance between two statistical units. Typically, there is not a single choice for the distance.

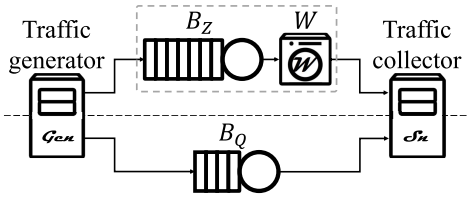

Fig. 10. Experimental setup with artificial traffic.

Accordingly, in order for DROPWAT to be invisible, we should identify a bottleneck component than can be substituted with a new component incorporating the watermarker in such a way that an attacker observing the traffic before and after the substitution does not detect any difference in the loss model from a statistical point of view. If the bottleneck component is substituted with a zero-loss component, i.e., it does not lose any packets, the statistical loss-equivalence will be imposed between $\mathcal{M}_{\mathcal{B}}$ and $\mathcal{M}_{\mathcal{W}}$.

This problem can be translated into a hypothesis test problem about an observed model $\mathcal{M}$, with two simple hypotheses $\mathcal{M} = \mathcal{M}_{\mathcal{B}}$ and $\mathcal{M} = \mathcal{M}_{\mathcal{S}}$. It is clear that loss-invisibility should be evaluated on a case by case basis, according to the model $\mathcal{M}_{\mathcal{B}}$. In addition, evaluating the statistical distance between two complex models is a difficult problem due to the presence of correlation between events.

C. Evaluation of DROPWAT's Loss-Invisibility

We evaluate the loss-invisibility according to a statistical comparison between: (i) a bottleneck component B_Q with buffer size Q that loses packets because of natural buffer overflow, and (ii) the concatenation of DROPWAT's watermarker W and a bottleneck component B_Z with buffer size Z, with $Q < Z < \infty$, that also loses packets because of natural buffer overflow. In the absence of a standardized metric to evaluate the statistical distance between two statistical processes, we perform a statistical comparison based on an empirical study of the loss density and the autocorrelation function, according to the analysis adopted by Yu *et al.* [44].

Let $\mathbf{B} = [b_1, b_2, \ldots, b_N]$ be the binary vector of packet loss events observed for traffic going out of a generic component, which can be either B_Q or $S = B_Z + W$, composed of N packets. The loss density $\psi_{\mathbf{B}}(k, q)$ is the frequency of k loss events in a block of q events, and the autocorrelation function $\rho_{\mathbf{B}}(h)$ for lag h is defined as

$$\rho_{\mathbf{B}}(h) = \frac{c_h}{c_0} \tag{5}$$

where

$$c_h = \frac{1}{N-1} \cdot \sum_{i=1}^{N-i} (b_i - \bar{b})(b_{i+h} - \bar{b}). \tag{6}$$

We measured $\psi_{\mathbf{B}}(k, q)$ and $\rho_{\mathbf{B}}(h)$ for two types of traffic: (i) heterogeneous traffic artificially generated in a controlled experimental setup, and (ii) real Internet traffic captured on a TOR relay.

To collect heterogeneous traffic, we created a simple network composed of three components (Figure 10): (i) a traffic

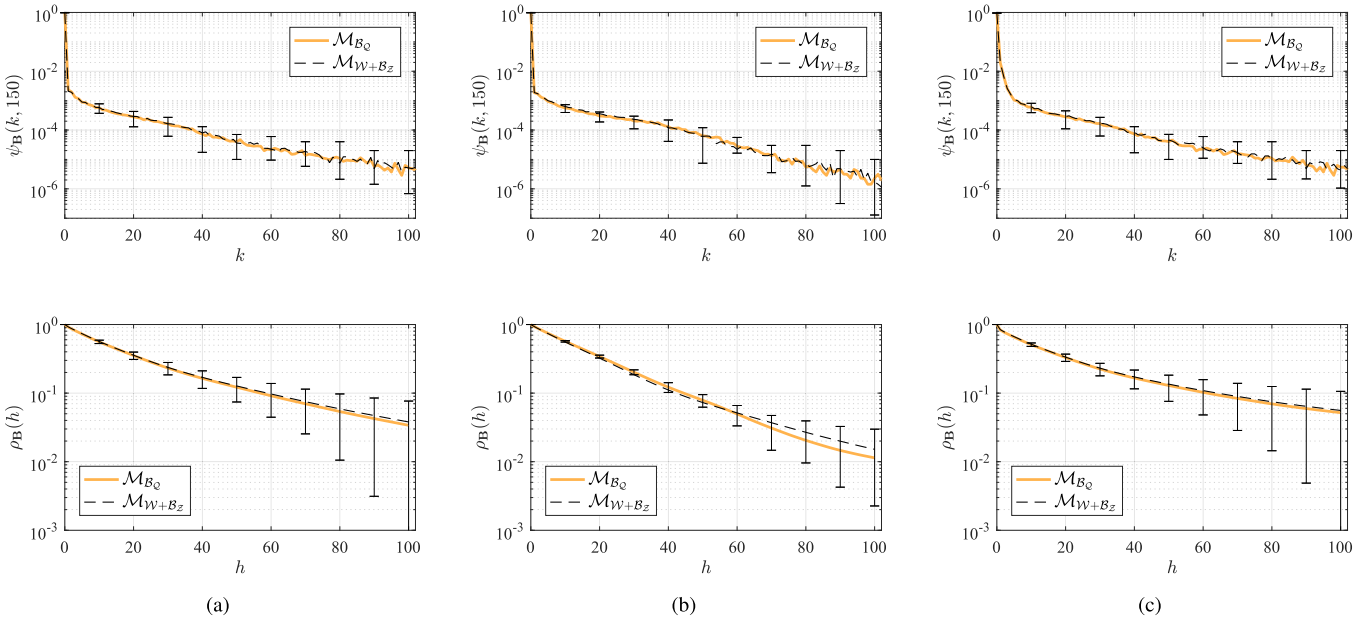

Fig. 11. Loss density and autocorrelation functions for bandwidth-consuming, enterprise, and TOR traffic ($Q = 8$, $Z = 50$). (a) Enterprise. (b) Bandwidth-consuming. (c) TOR.

source equipped with IXIA BreakingPoint VE, a commercial traffic generator software capable of generating network traffic at a rate up to 1 Gbit/s; (ii) an intermediate component alternating the role of either a single bottleneck B_Q or the concatenation of the bottleneck B_Z and the watermarker W; and (iii) a traffic destination node where traffic was collected in order to extract the corresponding binary vector of packet loss events. We configured the source node in order to generate two types of traffic: (i) "enterprise traffic" composed of a mix of 15 classes,[5] and (ii) "bandwidth-consuming traffic" consisting of a mix of HTTP streaming and P2P traffic at a constant rate.

In addition, we also evaluated the loss-equivalence on real Internet traffic. We first implemented a real TOR relay and then redefined the two settings shown in Figure 10 by replacing the traffic generator with the Internet, and the destination node with the TOR relay. According to the TOR protocol, the TOR relay was selected to forward traffic by several TOR clients. This allowed us to collect real TOR traffic. In this setup there is no traffic generator, and the traffic comes from the Internet.

We executed two sets of experiments for each type of traffic: one in which the intermediate component was a bottleneck node implemented on a Linux device with a limited egress queue of predetermined size $Q = 8$ pkts (B_Q), and one in which the intermediate component was the concatenation of a node with a limited egress queue of predetermined size $Z = 50$ pkts and the watermarker ($S = B_Z + W$).[6] 100 GB of traffic was generated and transferred through the Linux device for each experiment with artificial traffic, and 80 GB of traffic was collected at the relay coming from the Internet for each experiment with TOR. Using the scenario with the bottleneck node, we conducted 11 experiments; the binary vectors extracted from 10 experiments were used to compute the selected metrics, while the last binary vector was used as a training dataset to estimate the probabilities $\{p_{W,k}\}_{0<|k|\leq n}$ to use in the model $\mathcal{M}_{\mathcal{W}}$. 10 experiments were also conducted using the scenario with the watermarker.

Figure 11 provides a comparison of the loss density (for $q = 150$) and the autocorrelation function for the two models $\mathcal{M}_{\mathcal{S}}$ and $\mathcal{M}_{\mathcal{B}_Q}$ with three types of traffic. The figures show that the statistics for model $\mathcal{M}_{\mathcal{S}}$ nearly match those measured for $\mathcal{M}_{\mathcal{B}_Q}$, and they always stay within the uncertainty level of $\mathcal{M}_{\mathcal{B}_Q}$.

In order to obtain a numerical measure of the invisibility of DROPWAT, we used the Kolmogorov-Smirnov (KS) test to determine whether an observed sample generated by the model $\mathcal{M}_{\mathcal{S}}$ induces to accept or reject the hypothesis $\mathcal{M} = \mathcal{M}_{\mathcal{B}_Q}$. The test is based on the cumulative distribution function $\Psi_B(k, q)$ defined as

$$\Psi_{\mathbf{B}}(k, q) = \sum_{i=0}^{k} \psi_{\mathbf{B}}(i, q) \tag{7}$$

Let $\Psi_{\mathbf{B}}^{\mathcal{J}}(\cdot)$ be the empirical distribution function of the model $\mathcal{J}$, with $\mathcal{J} \in \{\mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{B}_Q}\}$. In the KS test the hypothesis $\mathcal{M} = \mathcal{M}_{\mathcal{B}_Q}$ is accepted if

$$\sup_k |\Psi_{\mathbf{B}}^{\mathcal{M}_{\mathcal{S}}}(k, q) - \Psi_{\mathbf{B}}^{\mathcal{M}_{\mathcal{B}_Q}}(k, q)| < \epsilon \tag{8}$$

We conducted the hypothesis test against $\mathbf{B}_{\mathcal{B}_Q}$ and $\mathbf{B}_{\mathcal{S}}$ (two sequences of events observed in the two experiments with a bottleneck and a watermarker, respectively). The KS distances obtained for the three types of traffic are listed in Table I.

[5]HTTP Video, HTTP Audio, HTTP Text, SIP/RTP Direct Voice Call over TCP, SIP/RTP Direct Voice Call over UDP, SMTP Email, AOL Instant Messenger, DCE RPC, SMB Null Session, SMB Client File Download, NFSv3, PostgreSQL, RTSP, SSH, and FTP

[6]The value of Z was selected so that B_Q and $B_Z + W$ had the same value of $\bar{b}$.

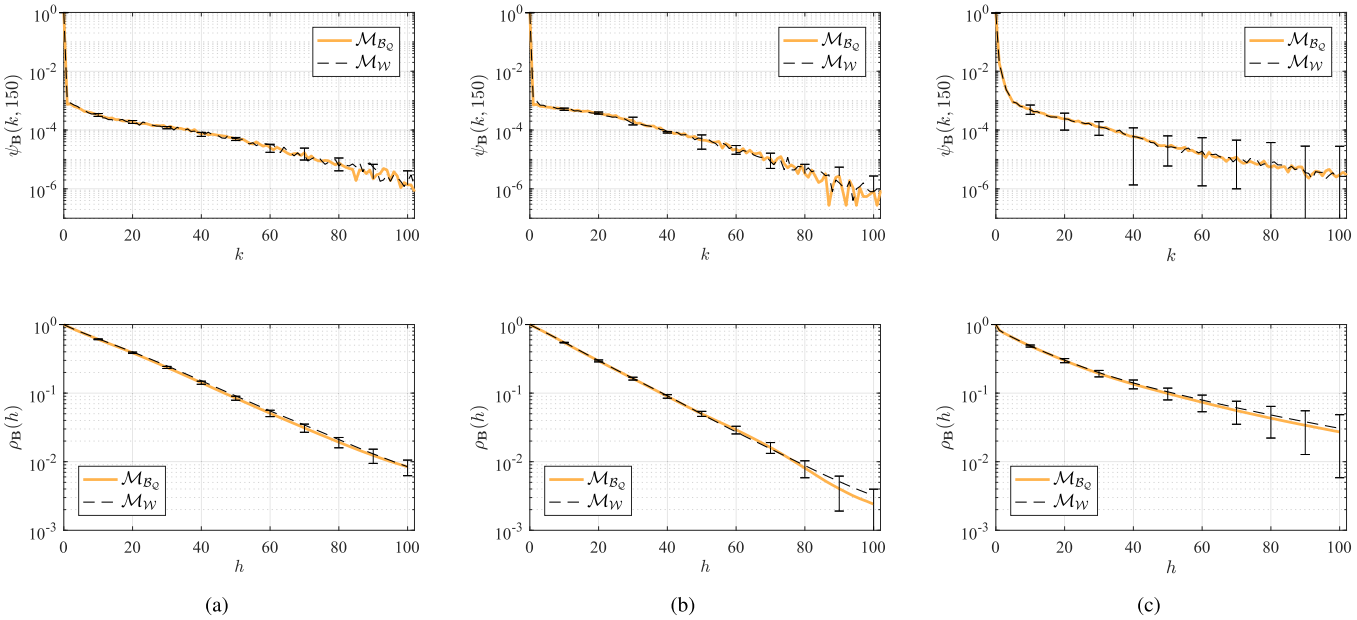

Fig. 12. Loss density and autocorrelation functions for bandwidth-consuming, enterprise, and TOR traffic ($Q = 10$, $Z = \infty$). (a) Enterprise. (b) Bandwidth-consuming. (c) TOR.

TABLE I
KS DISTANCES ($Q = 8$, $Z = 50$)

Type of traffic	b	KS distance
Enterprise	0.001978	0.000566
Bandwidth-consuming	0.001588	0.000886
TOR	0.002006	0.000374

TABLE II
KS DISTANCES ($Q = 10$, $Z = \infty$)

Type of traffic	b	KS distance
Enterprise	0.001834	0.000873
Bandwidth-consuming	0.001413	0.000733
TOR	0.001945	0.000339

In each case the KS distance is below 0.0009 which corresponds to high confidence (99%) that the two sequences, $\mathbf{B}_{\mathcal{B}_Q}$ and $\mathbf{B}_{\mathcal{W}}$, come from the same distribution. Thus, the watermark injected through DROPWAT will be invisible to any third party.

D. Relaxed Assumptions

Finding a component to substitute for B_Q and perform the same functionalities as B_Q but with a lower level of packet loss might be difficult. However, we can simplify the assumptions by noting that real enterprise networks are not static entities; instead, they evolve and change continuously: old routers may be replaced, new switches, routers, or servers may be added, the topology can change, a firewall can be updated, the access control lists of some routers can be modified, a new intrusion prevention system can be added, some services may be moved to the cloud, and so on. Thus, if a new component (which can be a single device, a network segment, or an entire subnet) is suddenly found, it should not come as a surprise to anybody. Given this, we can relax our assumptions and tolerate an adversary that is able to recognize the presence of a new component in the network but is not able to distinguish between a watermarker and another network component. According to the new assumption along with those defined in Subsection V-A, we can verify the loss-invisibility by just showing the loss-equivalence between the watermarker and a bottleneck node B.

We evaluate the loss-invisibility according to the relaxed assumptions, by following the analysis presented in the previous subsection, in which we remove the buffer B_Z (i.e., $Z = \infty$). Figure 12 show the comparison of $\psi_{\mathbf{B}}(k,q)$ and $\rho_{\mathbf{B}}(h)$, with $Q = 10$, and the KS distances are listed in Table II. In this case too, we can see that the statistics for model $\mathcal{M}_{\mathcal{B}}$ match those of $\mathcal{M}_{\mathcal{W}}$.

VI. ROBUSTNESS

Watermark robustness concerns the capacity of the watermark to survive along the path from the source to the destination despite (i) the natural noise introduced by the traversed networks, and (ii) possible attacks aiming at voluntarily distorting the watermark. In Section VII we present the results obtained by testing the DROPWAT's robustness against natural noise in two different network scenarios: in a controlled environment and in the real TOR network.

However, natural noise is not the only threat against network flow watermarking; several attacks were reported by the research community to be effective against timing-based network flow watermarking techniques: (i) timing perturbations, (ii) packet losses, (iii) dummy packet insertion, (iv) packet padding, (v) flow splitting and mixing, (vi) flow repacketization, and (vii) store and forward attacks [19].

As far as we know, there is currently no traceback method robust to all of these attacks. This is a challenging problem which requires further investigation. Like all timing-based

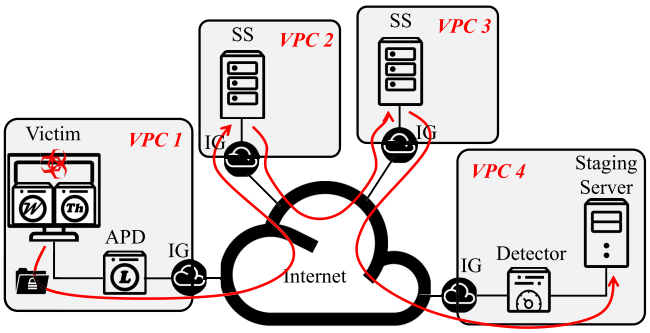

Fig. 13. Scenario with Web proxy servers in AWS.

techniques, DROPWAT is also vulnerable to the aforementioned attacks. However, the feasibility of these attacks requires that the SSs be under the control of the attacker, or at the very least, the controller of the SSs has to be cooperating with the attacker; this is not the case, including those involving anonymous networks (e.g., TOR, Anonymizer, I2P, JonDonym, etc.). The feasibility and cost of these attacks against timing-based network flow watermarking has been studied in several works [21], [37], [40] and is not further investigated in this paper.

VII. PERFORMANCE EVALUATION

We analyze the efficacy of DROPWAT based on experiments performed in the wild, with real traffic passing through the Internet. Performance was evaluated for two different network scenarios that are usually used for data exfiltration: (i) "Scenario A" where SSs were implemented as Web proxy servers on Amazon Web Services (AWS), and (ii) "Scenario B" where the traffic is forwarded over TOR, the well-known onion routing network.

A. Scenario With Web Proxy Servers

We developed a testing framework in which each node of the network topology is executed in an Amazon Elastic Compute Cloud (Amazon EC2) instance on AWS. Figure 13 shows the scheme of the framework. The main components in this architecture are:

- *Virtual private cloud (VPC)*: a logically isolated network unit in AWS where one or more EC2 instances can be launched.
- *Internet gateway (IG)*: a gateway that interconnects the instances in a VPC with the Internet.
- *Victim*: an EC2 instance representing the infected device of a company or person where sensitive data is stored. For testing purposes the module implementing the watermarker has been installed on this instance. A module which throttles the traffic in order to limit and control the bandwidth used by the malware is also installed on this instance.
- *Staging server*: an EC2 instance representing the remote server where the attacker forwards the exfiltrated data.
- *Stepping stones (SSs)*: two EC2 instances used in two different VPCs, interposed in the communication from the victim to the staging server.
- *Additional packet dropper (APD)*: an EC2 instance that randomly drops packets independently of the watermarker. This is used to test the robustness of DROPWAT.
- *Detector*: an EC2 instance which sniffs and collects all of the traffic going to the staging server and is located in the same VPC as the staging server.

The four VPCs were distributed in different geographic regions. A VPC can be launched from any one of AWS' 14 regions, distributed around the world; this implies that all of the traffic going from one node to another node passes through the Internet. Our experiment relied upon all of these regions: 10 regions were used to run the SSs, and the remaining four regions were employed to run the VPCs of the victim and the staging server. We used an "*m4.xlarge*" instance for the victim and an "*m4.large*" for the staging server, both equipped with a Microsoft Windows Server 2012 R2 Base Operating System. All other instances were "*t2.micro*" equipped with an Ubuntu Server 16.04 LTS.

B. Scenario With Onion Routing Servers

In this scenario we used the testing framework described in the previous subsection with a couple of differences: (i) the two SSs running on the EC2 instances were substituted with three onion routers belonging to the real TOR network, and (ii) the module throttling the traffic installed on the victim's instance and the APD were removed.

C. Implementation

1) Remote Administration Tool: Typically, an intruder performs an exfiltration attack by exploiting a remote access Trojan (RAT) which is usually downloaded invisibly on a victim's device within the targeted company's network. Once the RAT malware program has been installed, a backdoor is created allowing the attacker to obtain administrative control of the targeted computer. We used a commonly used backdoor malware for Windows systems, generated by Cerberus RAT (a RAT software publicly available on the Internet) and installed on the victim instance; the Cerberus remote controller was installed on the staging server.

2) Stepping Stones: In Scenario A, SS implementation is based on the SSH protocol. A PuTTY SSH client was used to create two SSH tunnels from the victim to each SS. Proxifier [48], a Windows-based proxy software, was used to set up two SOCKS-based proxies on the victim: one to channel Trojan-based TCP connections to the SSH tunnel that connects with the second SS, and the other one to divert the SSH tunnel of the second SS via the SSH tunnel that connects with the first SS. This creates an end-to-end encrypted channel, with one SSH tunnel encapsulated into the other. We set up 20 SSs distributed over 10 different AWS regions. At the beginning of each experiment two SSs were randomly selected by the victim, and the two corresponding SSH tunnels were established.

In Scenario B, the application traffic from the victim instance was tunnelled through the onion network using Torifier, a Windows-based torification tool [49]. We used the default TOR configuration which uses three relays to build the circuit. At the beginning of each experiment three

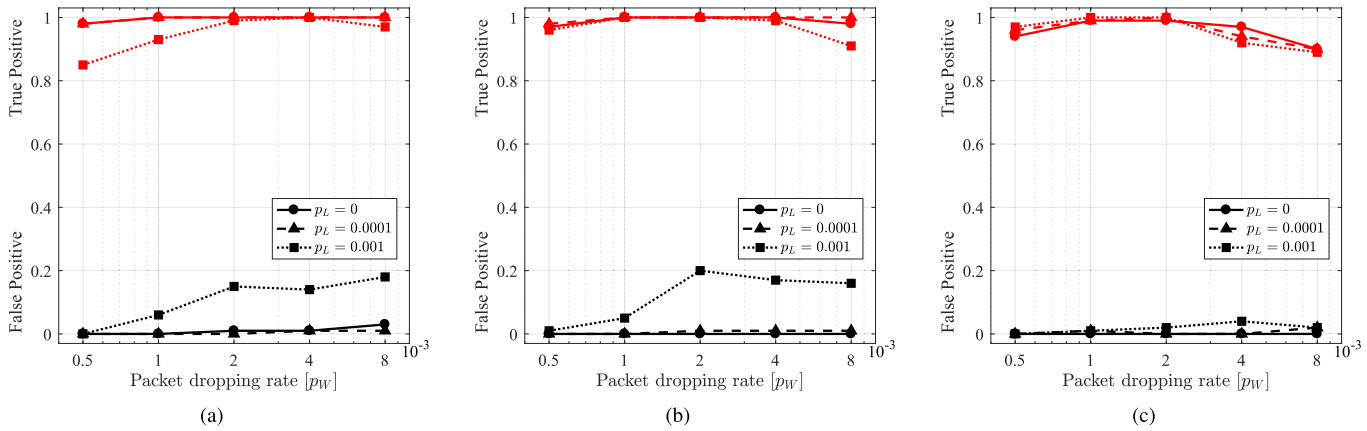

Fig. 14. True and false positive rates in the scenario with Web proxy servers ($\beta = 0.25$). (a) $R = 0.5$ MB/s. (b) $R = 1.0$ MB/s. (c) $R = 2.2$ MB/s.

TOR relays were selected by the victim, and a new circuit was established. In cases in which the selected circuit was exactly the same as the previous experiment, one of the three relays was substituted with a new randomly selected relay.

3) Watermarker: An application implemented in C++ that conducts the offline and online functions was executed on the victim instance. In the online application, Windows Packet Divert (WinDivert) [50], a packet filter library available for Windows distribution, was used to filter and queue network flow packets from the Windows network stack to the watermarker. Based on the precomputed dropping sequence, all of the packets observed during dropping time intervals were dropped.

4) Network Throughput and Additional Packet Loss: Network throughput can affect the performance of DROPWAT. We used the NetLimiter program [51] (installed on the victim instance) to throttle the Cerberus traffic and test different values of bandwidth use. Increasing the packet loss in the network was suggested by Sadeghi *et al.* [52] to mitigate covert channels based on packet drops. We tested the robustness of DROPWAT using several rates of additional packet loss; NetEm, a Linux facility for traffic control, was used in the Linux instance acting as an APD, in order to emulate different network packet loss rates and test the robustness of DROPWAT.

5) Detector: For testing purposes, detection was performed offline. Thus, no specific implementation was required in our framework - only an instance to intercept and sniff all of the traffic directed to the staging server was deployed.

D. Numerical Results

DROPWAT's accuracy was evaluated by conducting an extensive series of experiments; 7200 experiments on AWS and 500 experiments on TOR were executed, varying the values of several parameters: the transfer rate R, the threshold β, the packet dropping rate p_W, and the additional packet loss rate p_L. Each experiment consisted of transferring a 150 MB file from the victim to the staging server. We measured the true positive (TP) rate as the percentage of watermarked flows correctly classified as watermarked, and the false positive (FP) rate as the percentage of non-watermarked flows erroneously classified as watermarked.

A training phase was performed on a training dataset composed of 50 traces in order to test several values of the outlier threshold α and comparison window υ, and to select the values to use in the evaluation of the system. After the training phase, we selected $\alpha = 0.8$ and $\upsilon = 300$ pkts. The selection of the $\{p_{W,k}\}_{0<|k|\leq n}$ to use in the model $\mathcal{M}_W$ was made based on a training trace made up of 100 GB of traffic captured in the bottleneck setup described in Section V.

Figures 14 and 15 show the TP and FP rates obtained in our experiments in Scenario A involving Web proxy servers on AWS for three transfer rate values ($R = 0.5$, 1.0, and 2.2 MB/s) and two β values (0.25 and 0.35). On each graph, TP (red) and FP (black) rates are plotted for different values of p_L and with various packet dropping rates p_W. Each point on the curves corresponds to an average value computed over 100 experiments. FP rates were evaluated by testing the detector with both non-watermarked traces and traces watermarked with an incorrect seed.

As can be seen in the graphs, the detection algorithm is able to correctly detect watermarks, achieving very high TP rates (over 95% in most cases) and low FP rates (below 5%). Although variations of the transfer rates did not significantly affect performance, we observed a minor deterioration in the TP rate for cases in which the transfer rate is 2.2 MB/s; nevertheless it is still effective at detecting watermarks with few errors. When packet loss is greater than a specific threshold, a significant amount of noise is added to the sequence of IPDs which very slightly hinders the outlier detection function. For the same reason, TP rates also worsened as the combination of packet loss and packet drop frequencies increased. In addition, we observed a slight deterioration in the TP rate for $\beta = 0.35$. This is due to the fact that increasing the level of the detection threshold β reduces the implicit redundancy inside the watermark, which affects DROPWAT's TP rate, but at the same time it drastically reduces the FP rate. Thus, the detection system is highly effective with less error even at a higher threshold. A slight decrease in the TP rate can also be observed for $p_W = 0.5 \cdot 10^{-3}$; this is explained by the fact that

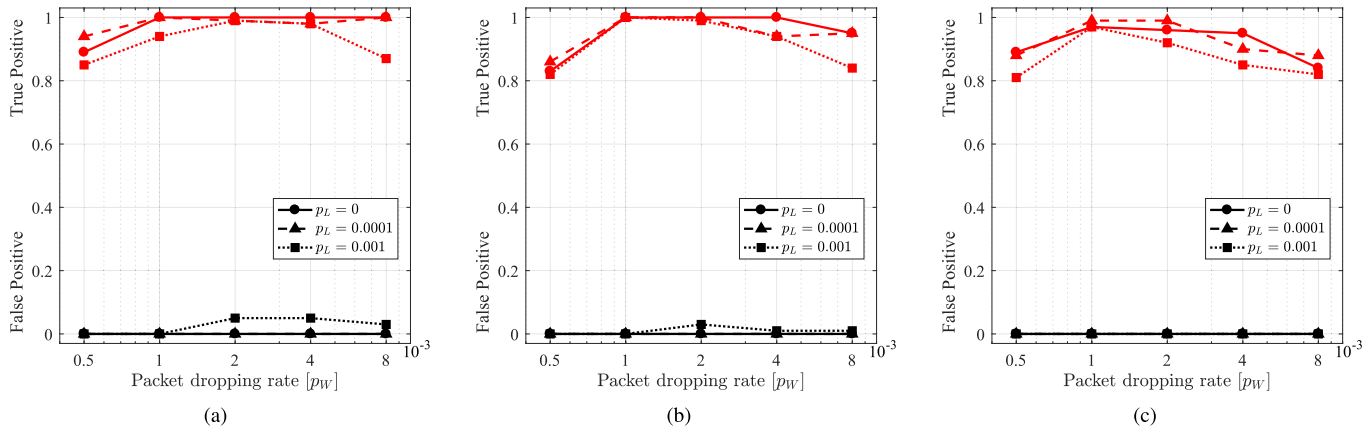

Fig. 15. True and false positive rates in the scenario with Web proxy servers ($\beta = 0.35$). (a) $R = 0.5$ MB/s. (b) $R = 1.0$ MB/s. (c) $R = 2.2$ MB/s.

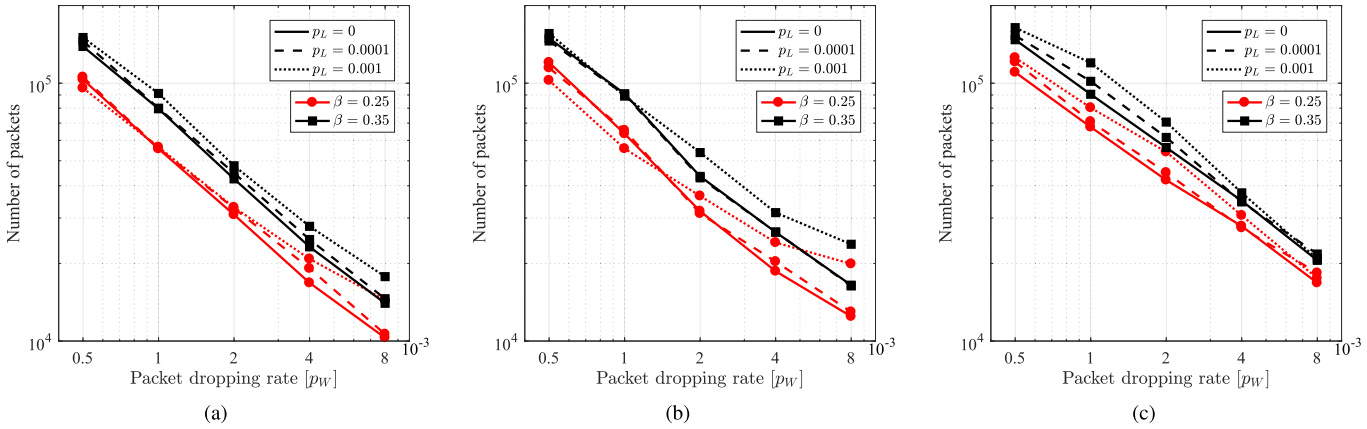

Fig. 16. Number of packets required to detect a watermark in the scenario with Web proxy servers ($\beta = 0.25$ in red and $\beta = 0.35$ in black). (a) $R = 0.5$ MB/s. (b) $R = 1.0$ MB/s. (c) $R = 2.2$ MB/s.

in this scenario not enough packets are dropped before the file transfer ends. FP rates increase in a scenario with high packet loss rate and low throughput when we consider a threshold $\beta = 0.25$; this is a sign that the watermark is negatively affected in the case of a high level of natural packet losses. Nevertheless FP rates are below 5% in all of the other cases. Therefore, we can state that for practical implementation, watermarks can be detected with almost 100% TP and 0% FP rates by fine-tuning the system parameters based on the knowledge of network loss behavior, even with the presence of a mitigation technique.

Figure 16 shows the number of packets required to detect the watermark for three transfer rate values ($R = 0.5$, 1.0, and 2.2 MB/s). On each graph, curves are plotted for three values of p_L, and for $\beta = 0.25$ (red) and $\beta = 0.35$ (black), by varying the packet dropping probability p_W. Each point on the curves corresponds to an average value computed over all of the experiments that resulted in the correct identification of a watermark. The number of packets needed to identify the watermark ranges from 10^4 to $1.5 \cdot 10^5$. It is no surprise that in all of the cases the number of packets required for detection decreases linearly as the packet dropping rate increases.

We also tested DROPWAT in a scenario with the TOR network (Scenario B). Even though TOR is not optimal for performing the transfer of a massive amount of data, testing the watermarking system in a scenario with onion routing servers allows us to stress robustness in the presence of a significant amount of noise that is primarily due to relay instability, significant end-to-end delay, and large jitter. Figure 17 shows the TP and FP rates for two values of p_W by varying the threshold β. Despite the slight decrease in performance, the proposed method can detect the watermark in 95% of cases (best instance), with an FP rate of less than 10%.

In this scenario we also measured the number of packets observed by the detector before identifying the watermark. Figure 18 shows the number of packets for two dropping rate values ($p_W = 10^{-3}$ and $2 \cdot 10^{-3}$) by varying the threshold β. The curves confirm the results obtained for the scenario with Web proxy servers and highlight the linear trend of the number of packets required for detection as a function of parameter β.

VIII. DISCUSSION AND CHALLENGES

Unlike other watermarking algorithms presented in the literature, DROPWAT has the following properties: (i) it is invisible to the adversary; (ii) it is effective, even with a high transfer rate; and (iii) it is effective against traffic passing through the TOR network.

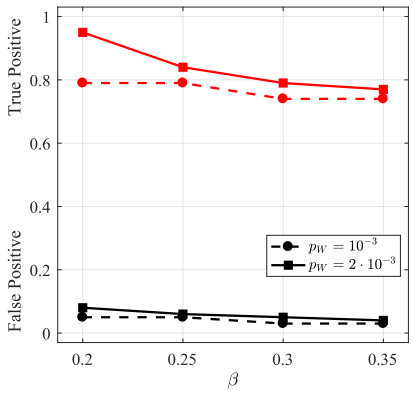

Fig. 17. True and false positive rates in the scenario with onion routing servers.

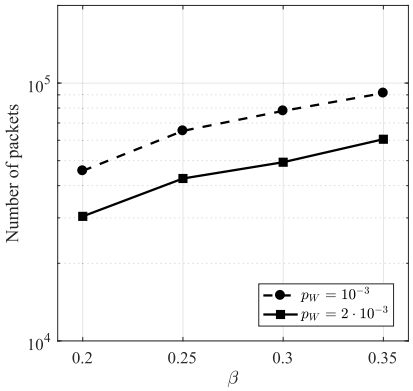

Fig. 18. Number of packets required to detect a watermark in the scenario with onion routing servers.

Dropping packets on a pseudo-random basis implies that deterministic analysis would not provide any evidence of the watermark. Additionally, statistical analysis would be incapable of this as well, because the packet loss behavior induced by DROPWAT reflects a natural behavior of loss in the network.

The extensive number of experiments performed showed that DROPWAT is effective in a variety of scenarios, with different network conditions, even in the presence of different TCP stack implementations. The robustness of DROPWAT was also verified in the case of an attacker that intentionally drops packets with the aim of obfuscating the watermark.

The DROPWAT's peculiarity is the ability to take advantage of the network features, the operation of the SS, and the interaction with network protocols. Nevertheless, as previously stated, the way the SS handles the traffic may affect the incisiveness and detectability of any watermark. For instance, a timing-based algorithm needs the SS to work seamlessly, in order to safeguard the temporal patterns. If an SS uses a store and forward method, in which received data is buffered for a period of time before being forwarded to the next hop, watermarking algorithms (including DROPWAT) cannot work properly. In addition, although typical SS implementations work at the application level and do not propagate losses up to the destination, an implementation in which the recovery of the losses is left to the final destination is certainly feasible. In this case, although outliers are not generated and our current implementation of the detector would not be able to detect a watermark, the detection algorithm can be implemented according to a much simpler logic.

Another limitation is that DROPWAT is ineffective for short-lived or interactive flows, because the p_W must be low enough to ensure that (i) packet dropping does not affect the throughput, and (ii) TP rates are sufficiently high.

IX. CONCLUSION

In this paper we proposed a new watermarking technique for tracing data exfiltration attacks. DROPWAT has two main characteristics that differentiate it from other existing solutions for the network traceback problem. First, DROPWAT's embedding algorithm is based on a new paradigm to impress a watermark within a network flow that takes advantage of a network's reaction to packet loss. We have shown that dropping a few selected packets of a flow allows a timing-based watermark to be embedded into the flow. Second, the watermark embedded by DROPWAT is invisible to the adversary under some assumptions. The invisibility is due to the fact that the time modification generated by an artificially dropped packet is the same as that of a packet that is naturally lost. In addition, because the statistical behavior of the loss pattern induced by DROPWAT matches the loss behavior of a real bottleneck node, an adversary cannot distinguish between a watermark embedded by DROPWAT and a natural loss pattern in the network. Our experimental results showed that DROPWAT achieves very high TP rates and very low FP rates, even in realistic scenarios where traffic passes through Web proxy servers on AWS or an anonymous network like TOR.

REFERENCES

[1] F. Li, A. Lai, and D. Ddl, "Evidence of advanced persistent threat: A case study of malware for political espionage," in *Proc. IEEE 6th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2011, pp. 102–109.

[2] B. Binde, R. McRee, and T. J. O'Connor, "Assessing outbound traffic to uncover advanced persistent threat," SANS Inst., Bethesda, MD, USA, White Paper, 2011.

[3] M. Lee and D. Lewis, "Clustering disparate attacks: Mapping the activities of the advanced persistent threat," in *Proc. 21st Virus Bull. Int. Conf.*, vol. 26. 2011, pp. 1–22.

[4] S. DeWeese, *Capability of the People's Republic of China to Conduct Cyber Warfare and Computer Network Exploitation*. Collingdale, PA, USA: Diane Publishing, 2009.

[5] P. Giura and W. Wang, "A context-based detection framework for advanced persistent threats," in *Proc. IEEE Int. Conf. Cyber Secur. (CyberSecurity)*, Dec. 2012, pp. 69–74.

[6] A. Giani, V. H. Berk, and G. V. Cybenko, "Data exfiltration and covert channels," *Proc. SPIE*, vol. 6201, p. 620103, May 2006.

[7] E. Bertino and G. Ghinita, "Towards mechanisms for detection and prevention of data exfiltration by insiders: Keynote talk paper," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Secur.*, 2011, pp. 10–19.

[8] Y. Liu, C. Corbett, K. Chiang, R. Archibald, B. Mukherjee, and D. Ghosal, "SIDD: A framework for detecting sensitive data exfiltration by an insider attack," in *Proc. IEEE 42nd Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2009, pp. 1–10.

[9] F. P. Buchholz and C. Shields, "Providing process origin information to aid in network traceback," in *Proc. USENIX Annu. Tech. Conf., Gen. Track*, 2002, pp. 261–274.

[10] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 295–306, 2000.

[11] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proc. IEEE 20th Annu. Joint Conf. Comput. Commun. Soc. (INFOCOM)*, vol. 2. Apr. 2001, pp. 878–886.

[12] M. Sung and J. Xu, "IP traceback-based intelligent packet filtering: A novel technique for defending against Internet DDoS attacks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 9, pp. 861–872, Sep. 2003.

[13] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale IP traceback in high-speed Internet: Practical techniques and theoretical foundation," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 115–129.

[14] Y.-S. Choi, D.-I. Seo, S.-W. Sohn, and S.-H. Lee, "Network-based real-time connection traceback system (NRCTS) with packet marking technology," in *Proc. Int. Conf. Comput. Sci. Appl. (ICCSA)*, Montreal, QC, Canada, 2003, pp. 31–40.

[15] S. Mitropoulos, D. Patsos, and C. Douligeris, "Network forensics: Towards a classification of traceback mechanisms," in *Proc. IEEE Workshop 1st Int. Conf. Secur. Privacy Emerg. Areas Commun. Netw.*, Sep. 2005, pp. 9–16.

[16] I. Hamadeh and G. Kesidis, "A taxonomy of Internet traceback," *Int. J. Secur. Netw.*, vol. 1, nos. 1–2, pp. 54–61, 2006.

[17] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, "Sleepy watermark tracing: An active network-based intrusion response framework," in *Proc. 16th Annu. Working Conf. Inf. Secur. (IFIP/SEC)*, Paris, France, 2001, pp. 369–384.

[18] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, and K. Szczypiorski, *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. Hoboken, NJ, USA: Wiley, 2016.

[19] A. Iacovazzi and Y. Elovici, "Network flow watermarking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 512–530, 1st Quart., 2017.

[20] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proc. 10th ACM Conf. Comput. Commun. Secur.*, 2003, pp. 20–29.

[21] P. Peng, P. Ning, D. S. Reeves, and X. Wang, "Active timing-based correlation of perturbed traffic flows with chaff packets," in *Proc. 25th IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2005, pp. 107–113.

[22] Y. J. Pyun, Y. H. Park, X. Wang, D. S. Reeves, and P. Ning, "Tracing traffic through intermediate hosts that repacketize flows," in *Proc. 26th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, May 2007, pp. 634–642.

[23] A. Houmansadr, N. Kiyavash, and N. Borisov, "Rainbow: A robust and invisible non-blind watermark for network flows," in *Proc. NDSS*, 2009, pp. 1–13.

[24] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 116–130.

[25] X. Gong, M. Rodrigues, and N. Kiyavash, "Invisible flow watermarks for channels with dependent substitution and deletion errors," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 1773–1776.

[26] A. Houmansadr and N. Borisov, "SWIRL: A scalable watermark to detect correlated network flows," in *Proc. NDSS*, 2011, pp. 1–15.

[27] A. Houmansadr and N. Borisov, "BotMosaic: Collaborative network watermark for the detection of IRC-based botnets," *J. Syst. Softw.*, vol. 86, no. 3, pp. 707–715, 2013.

[28] J. Luo, X. Wang, and M. Yang, "An interval centroid based spread spectrum watermarking scheme for multi-flow traceback," *J. Netw. Comput. Appl.*, vol. 35, no. 1, pp. 60–71, 2012.

[29] X. Wang, J. Luo, and M. Yang, "A double interval centroid-based watermark for network flow traceback," in *Proc. IEEE 14th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, Apr. 2010, pp. 146–151.

[30] A. Zand, G. Vigna, R. Kemmerer, and C. Kruegel, "Rippler: Delay injection for service dependency detection," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 2157–2165.

[31] D. Ramsbrock, X. Wang, and X. Jiang, "A first step towards live botmaster traceback," in *Proc. 11th Int. Symp. Recent Adv. Intrusion Detection (RAID)*, Cambridge, MA, USA, 2008, pp. 59–77.

[32] Z. Ling, X. Fu, W. Jia, W. Yu, D. Xuan, and J. Luo, "Novel packet size-based covert channel attacks against anonymizer," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2411–2426, Dec. 2013.

[33] D. Arp, F. Yamaguchi, and K. Rieck, "Torben: A practical side-channel attack for deanonymizing Tor communication," in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur.*, 2015, pp. 597–602.

[34] E. Chan-Tin, J. Shin, and J. Yu, "Revisiting circuit clogging attacks on Tor," in *Proc. IEEE 8th Int. Conf. Availability, Rel. Secur. (ARES)*, Sep. 2013, pp. 131–140.

[35] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 18–32.

[36] X. Gong, M. Rodrigues, and N. Kiyavash, "Invisible flow watermarks for channels with dependent substitution, deletion, and bursty insertion errors," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 11, pp. 1850–1859, Nov. 2013.

[37] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *Proc. USENIX Secur. Symp.*, 2008, pp. 307–320.

[38] X. Luo, J. Zhang, R. Perdisci, and W. Lee, "On the secrecy of spread spectrum flow watermarks," in *Proc. 15th Eur. Symp. Res. Comput. Secur. (ESORICS)*, Athens, Greece, 2010, pp. 232–248

[39] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. Chang, "Exposing invisible timing-based traffic watermarks with BACKLIT," in *Proc. ACM 27th Annu. Comput. Secur. Appl. Conf.*, 2011, pp. 197–206.

[40] Z. Lin and N. Hopper, "New attacks on timing-based network flow watermarks," in *Proc. USENIX Secur. Symp.*, 2012, pp. 381–396.

[41] W. Jia, F. P. Tso, Z. Ling, X. Fu, D. Xuan, and W. Yu, "Blind detection of spread spectrum flow watermarks," *Secur. Commun. Netw.*, vol. 6, no. 3, pp. 257–274, 2013.

[42] M. Edman and B. Yener, "On anonymity in an electronic society: A survey of anonymous communication systems," *ACM Comput. Surv.*, vol. 42, no. 1, p. 5, 2009.

[43] H. A. Sanneck and G. Carle, "Framework model for packet loss metrics based on loss runlengths," *Proc. SPIE*, vol. 3969, pp. 177–187, Dec. 1999.

[44] X. Yu, J. W. Modestino, and X. Tian, "The accuracy of Gilbert models in predicting packet-loss statistics for a single-multiplexer network model," in *Proc. IEEE 24th Annu. Joint Conf. Comput. Commun. Soc. (INFOCOM)* , vol. 4. Mar. 2005, pp. 2602–2612.

[45] Z. Ling, J. Luo, K. Wu, and X. Fu, "Protocol-level hidden server discovery," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1043–1051.

[46] X. Zhang, S. F. Wu, Z. Fu, and T.-L. Wu, "Malicious packet dropping: How it might impact the TCP performance and how we can detect it," in *Proc. Int. Conf. IEEE Netw. Protocols*, Nov. 2000, pp. 263–272.

[47] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting intentional packet drops on the Internet via TCP/IP side channels," in *Proc. 15th Int. Conf. Passive Active Meas. (PAM)*, Los Angeles, CA, USA, 2014, pp. 109–118.

[48] *Windows Proxifier*. Accessed: Sep. 1, 2016. [Online]. Available: https://www.proxifier.com/

[49] *Torifier*. Accessed: Sep. 1, 2016. [Online]. Available: http://www.torifier.com/

[50] *Windows Packet Divert (WinDivert)*. Accessed: Sep. 1, 2016. [Online]. Available: https://reqrypt.org/windivert.html

[51] *Netlimiter*. Accessed: Sep. 1, 2016. [Online]. Available: https://www.netlimiter.com/

[52] S. Schulz, V. Varadharajan, and A.-R. Sadeghi, "The silence of the LANs: Efficient leakage resilience for IPsec VPNs," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 2, pp. 221–232, Feb. 2014.

Alfonso Iacovazzi, photograph and biography not available at the time of publication.

Sanat Sarda, photograph and biography not available at the time of publication.

Daniel Frassinelli, photograph and biography not available at the time of publication.

Yuval Elovici, photograph and biography not available at the time of publication.