The DUSTER Attack: Tor Onion Service Attribution Based on Flow Watermarking with Track Hiding

Alfonso Iacovazzi[1], Daniel Frassinelli[2], and Yuval Elovici[1,3]

[1]ST Engineering-SUTD Cyber Security Laboratory – Singapore University of Technology and Design, Singapore
[2]CISPA Helmholtz Center for Information Security, Saarland University, Germany
[3]Department of Information Systems Engineering, Ben-Gurion University, Israel

Abstract

Tor is a distributed network composed of volunteer relays which is designed to preserve the sender-receiver anonymity of communications on the Internet. Despite the use of the onion routing paradigm, Tor is vulnerable to traffic analysis attacks. In this paper we present DUSTER, an active traffic analysis attack based on flow watermarking that exploits a vulnerability in Tor's congestion control mechanism in order to link a Tor onion service with its real IP address. The proposed watermarking system embeds a watermark at the destination of a Tor circuit which is propagated throughout the Tor network and can be detected by our modified Tor relays in the proximity of the onion service. Furthermore, upon detection the watermark is cancelled so that the target onion service remains unaware of its presence. We performed a set of experiments over the real Tor network in order to evaluate the feasibility of this attack. Our results show that true positive rates above 94% and false positive rates below 0.05% can be easily obtained. Finally we discuss a solution to mitigate this and other traffic analysis attacks which exploit Tor's congestion control.

1 Introduction

Anonymous networks are utilised by Internet users to privately surf the Web without being traced by a third party. Typically, when an Internet communication is intermediated by such anonymous network, any entity observing the traffic at a single point along the communication path cannot identify the communicating peers. Tor is the most popular anonymous network currently in use, and it is widely used to circumvent any kind of censorship or network activity monitoring.

Tor is based on the onion routing paradigm in which privacy is guaranteed by encapsulating messages into packets protected by several layers of encryption and transferring these packets through a chain of relays [1]. Tor is an open-source project in which the relays constituting the infrastructure of the network are voluntarily made available by Tor users, and are distributed worldwide. In addition, Tor enables users to create "onion services" which allow them to offer content and/or services while preserving their anonymity. Onion services can only be reached by peers connected to the Tor network, and are identified only by their "onion address." Onion services ensure sender-receiver anonymity as the server and client can only communicate through the Tor network.

Related works

Tor has been investigated by researchers and attackers over the past decades for the purpose of identifying vulnerabilities and/or improving its security and privacy aspects. The most relevant attacks against Tor fall into two categories: (i) denial of service (DoS) attacks, and (ii) de-anonymization attacks. In this paper we focus on the latter category.

DoS attacks are well known attacks usually investigated in traditional Internet. When targeting Tor, the main goal of these attacks is to exhaust resources (bandwidth, memory, etc.) of Tor relays or onion services in order to make the Tor network unavailable to its users [3, 6, 11, 15]. One example is the "Sniper" attack proposed by Jansen *et al.* which exploits a vulnerability in Tor's congestion control system to anonymously and selectively disable arbitrary Tor relays. In the attack, a malicious client continuously sends `SENDME` cells to an exit relay without retrieving the returning data cells: this causes the relay to buffer those cells until exhausting its resources [11]. Another example is the "CellFlood" attack proposed by Barbera *et al.* in which the attacker overloads a target Tor relay by sending an excessive number of circuit CREATE requests, which leads the relay to start discarding the additional CREATE requests, even those originated from honest Tor clients [3].

In the second attack category we find de-anonymization attacks. In these attacks the aim of the attacker is to actively or passively analyse the network (traffic patterns, addresses, etc.) in order to breach the sender-receiver anonymity. Several attacks based on passive traffic analysis (TA) techniques have already been demonstrated to correlate sender and receiver

traffic and be able to uncover onion services [16, 18]. Fingerprinting techniques are a type of robust passive TA attack in which the adversary uses machine learning algorithms to fingerprint the traffic coming from a website [22] or an onion service [12,17,19]. As example, Kwon *et al.* demonstrate how a passive adversary can easily detect the presence of an onion service via "circuit fingerprinting." Furthermore they apply website fingerprinting on the circuit traffic to infer which onion service is being visited.

In contrast to passive TA attacks, active TA attacks have been shown to be much more powerful in breaching the sender-receiver anonymity [4, 5, 14, 20, 21]. For example, in the attack proposed by Ling *et al.*, compromised exit relays pseudo randomly delay the cells of normal circuits; the added noise is detected by compromised guard relays which de-anonymize the clients [14]. Pappas *et al.* propose the "Packet spinning" attack which allows malicious Tor relays to overload legitimate relays: in this way the attacker increases the probability that its malicious relays are selected by the clients to build their circuits [20]. An approach similar to the Packet spinning is proposed by Borisov *et al.* in which malicious relays perform selective DoS attacks on Tor circuits; this attack generates traffic patterns which can be easily identified by other malicious relays [5]. The attack investigated by Biryukov *et al.* allows to de-anonymize the entry guard of an onion service by forcing it to connect to a rendezvous and middle relay controlled by the attacker [4]. Finally, Rochet and Pereira exploits Tor's lax dropping policies to watermark a rendezvous circuit by sending numerous padding cells which are silently dropped by the targeted onion service [21].

The presented attacks, despite being successful, presents some limitations. First of all, although some DoS-based attacks can be very successful in selectively tracking/isolating a Tor peer, they are easy to detect by legitimate users and are therefore not suitable for stealthy tracking [5, 20]. Following, TA based attacks often require the attacker to control multiple relays of the same circuit (e.g, guard and exit), which is very unlikely in the current Tor implementation [4, 14]. Finally, most attacks were based on vulnerabilities which have been fixed by the Tor community over the years [5, 11].

Contribution

In this paper we show the feasibility of a new active TA attack against Tor which can be used to compromise onion services anonymity. The attack belongs to the family of network flow watermarking techniques in which the attacker modifies statistical features of the traffic to embed a watermark into the communication [8]. The proposed attack differs from previous works as it allows the attacker to cancel the tracks of the watermark once it has been detected, making it unnoticeable to the onion service – hence the name DUSTER. The DUSTER attack embeds a watermark into a Tor stream by exploiting Tor's congestion control system. When detected by one of the relays controlled by the attacker, the watermark is used to uncover the real IP address of the onion service.

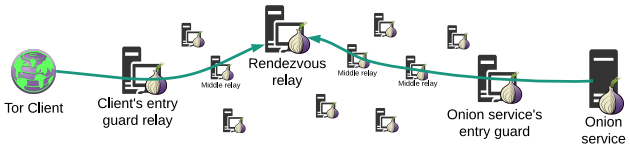


Figure 1: Tor rendezvous circuit.

The DUSTER attack has several advantages compared to traditional active traffic correlation attacks with the same privileged attacker position. Unlike the cell counter attack proposed by Ling *et al.* [14], which target only standard circuits, DUSTER works on both standard and rendezvous circuits, is hidden from the tracked endpoint, and does not affect the network performance. These last two features also make DUSTER stronger than the padding cell attack proposed by Rochet and Pereira [21]; in fact, their attack can be detected easily by the targeted onion service and requires additional traffic to be sent, which may affect the network performance. In addition, unlike the attack proposed by Biryukov *et al.* [4], the DUSTER attack does not require the attacker to control the rendezvous relay.

To summarise, the main novelties in DUSTER are:

- it exploits a vulnerability in Tor's congestion control mechanism;
- it is hidden from the target endpoint;
- it applies to both standard and rendezvous circuits;
- it is lightweight and does not affect network performance;
- it works on up-to-date Tor implementations.

Paper organisation

The rest of the paper is organised as follows. An overview of the Tor network, its congestion control mechanism, and onion services is provided in Section 2. In Section 3 we describe the attack and the rules used to embed, detect, and cancel the watermark. The experimental evaluation is provided in Section 4. A solution to mitigate the attack is described in Section 5, which is followed by the conclusion in Section 6.

2 Tor, onion services, and congestion control

In this section we provide an overview of Tor's key concepts useful for the understanding of the paper. Readers interested in more details may refer to the official project page for complete specifications [1].

In the Tor network, peers communicate via Tor *circuits*. A Tor circuit is a virtual path that traverses the Tor network and

is obtained by having the traffic routed via a series of Tor *relays*. Each Tor relay can be seen as a proxy which hides the IP addresses of the communication endpoints and applies encryption/decryption on the traversing data units according to the onion routing paradigm.

There are two types of circuits in Tor. First, standard circuits are used by Tor clients (e.g., Tor Browser) to communicate with Internet services outside of the Tor network (e.g., google.com), and they typically traverse 3 different relays. These relays are chosen by the client and are of three types: *guard*, *middle*, and *exit* relays. This categorisation is provided by the Tor consensus, a document containing all the information about the relays, compiled by the directory authorities and accessible to all clients. Each relay is given a flag on the basis of various metrics such as bandwidth, up-time, stability, configuration, etc. These flags are used by clients to select the relays for their circuits according to Tor's circuit creation rules; For example, the first relay of a circuit must always have the guard flag.

The second type of Tor circuits are the "rendezvous" circuits. These circuits are used by Tor clients to communicate with Tor onion services. Apart for the circuit creation, which has no relevance for the attack discussed in this paper, the main difference between rendezvous and standard circuits is that the number of traversed relays is 6 instead of 3. In a rendezvous circuit, both the client and the onion service open a (respectively) two and three hop circuit towards a previously agreed relay called *rendezvous* relay. This relay acts as a bridge that interconnects the two independent circuits. This means that the client communicates to the rendezvous relay via a circuit composed of a guard and middle relay; The server (onion service) uses a three hop circuit consisting of a guard and two middle relays to communicate to the same rendezvous relay. Figure 1 presents an example of a rendezvous circuit.

Packets exchanged between peers in Tor are organised in fixed-size "cells" and grouped in logically separated "streams." All streams sharing both sender and receiver are multiplexed into the same circuit and are processed according to the onion paradigm in which each traversed relay adds or removes an encryption layer. In a rendezvous circuit, the rendezvous relay has the role of routing the cells coming from one circuit to the other (with related encryption/decryption). In Tor jargon, cells travelling from one endpoint to the rendezvous relay are referred to as "OUT cells," while cells moving in the opposite direction are referred to as "IN cells."

Finally, to control the network congestion, Tor uses an end-to-end congestion avoidance mechanism based on dedicated `SENDME` cells. The congestion control rule is that an endpoint transmits one `SENDME` cell after receiving 50 data cells per stream and one `SENDME` cell after receiving 100 data cells per circuit. On the opposite, an endpoint can transmit a data cell if and only if there are less than 500 data cells per stream and 1000 data cells per circuit that have not yet been acknowledged. In a circuit with a single stream, assuming a continuous flow of data from the sender to the receiver, this results in a pattern of 1-2-1-2- `SENDME` cells from the receiver to the sender (Figure 2). In a rendezvous circuit, the `SENDME` and data cells are transferred between the client and the onion service, they are always encrypted along the path, and none of the traversed relays can distinguish between `SENDME` and data cells.

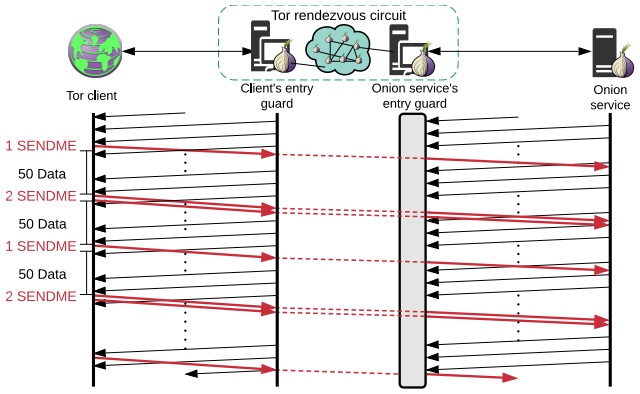


Figure 2: Traditional Tor cell exchange.

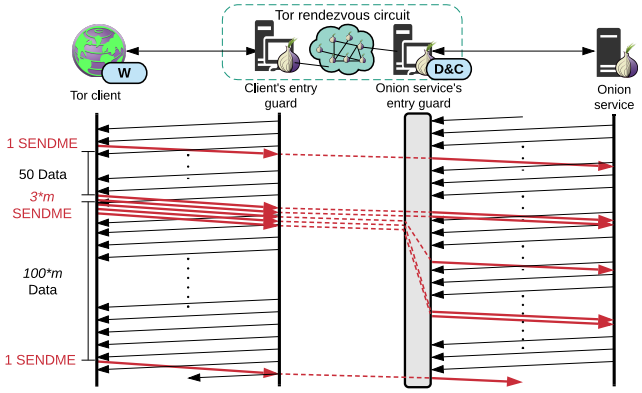


Figure 3: Watermarked Tor cell exchange.

For simplicity, in the following we often refer to a `SENDME` triplet which consists of a set of stream-circuit-stream `SENDME` cells which acknowledge 100 data cells.

3 Watermarking system

We consider a scenario in which an adversary is interested in linking as many onion services as possible to their real IP addresses. For this purpose, the attacker creates a system to crawl the dark Web. The system is composed of (i) a watermarker, which is a Tor client modified to inject a watermark into its incoming traffic, and (ii) multiple detectors, which are

Tor guard relays controlled by the adversary and modified to search for the watermark and, if detected, to cancel it.

The client builds a list of onion addresses and uses it to crawl all corresponding services. After establishing a rendezvous circuit with an onion service, the client downloads some content from it (e.g., HTML documents, files, multimedia) and in parallel injects a watermark into the data stream. At the same time, the detectors passively analyses all circuits passing through them. If one of the detectors detects the watermark, it provides the IP address of the circuit endpoint to the attacker, and cancel the watermark. The attacker can associate the onion address contacted by the client to its real IP address discovered by the detector.

After an instance of download is completed, the client moves to the next onion address on its list. The list of onion addresses can be built in various ways: most onion services' addresses are publicly listed by Tor's directory services whereas others require an invitation and/or some sort of manual setup. How to gather the list of onion addresses is outside the scope of this paper. Ghosh *et al.* offer an example of how to automatically discover and categorise onion services [7].

The core watermarking/detection functions are described in the following subsections and in Figure 3. For sake of clarity, the corresponding pseudo-code is provided in Appendix A.

3.1 Embedding process

We detail the watermarking process by considering a single instance of DUSTER activity in which the adversary's Tor client knows an onion address and establishes a circuit with the corresponding onion service. Once the rendezvous circuit has been established, the client requests some content from the onion service. This generally happens via a client-server protocol (e.g., HTTPS) that encodes the content length together with the response. Knowing the content length is required to determine whether the content is large enough to embed a watermark into the data stream. Given a content provided by the onion service, if its length L is greater than or equal to a predefined threshold L_{min}, the watermark can be embedded. The selection of the minimum content length L_{min} is based on a couple of system parameters and is discussed in Subsection 3.4.

Assuming a resource that meets the mentioned criteria is found, the client starts the watermarking process. Figure 4 depicts the finite state machine of the watermarking process. The transitions between states are described by means of connecting arrows. Each arrow is labelled with the events causing the transition (above the horizontal line) and the actions executed because of the transition (below the horizontal line). The watermarking process consist of three states: INIT, SLEEP, and QUIT states.

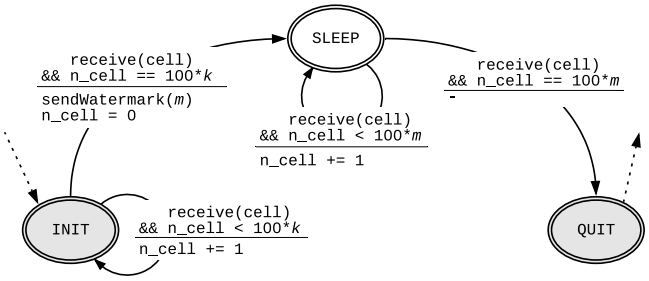


Figure 4: Finite-state machine of the watermarker.

INIT

The client starts the watermarking process at the INIT state. When in this state, the client strictly follows the Tor congestion protocol, counts the received data cells, and acts normally until it receives $100 \cdot k$ data cells. This initial waiting interval is needed to generate enough traffic that can be used by the detectors to validate the circuit and compute a set of bandwidth-dependent detection parameters. Upon receiving $100 \cdot k$ data cells the client executes the core of the attack consisting of the anticipation of m triplets of SENDME cells, which are sent in a single batch. Afterwards, the client transits to the SLEEP state.

SLEEP

The client stops sending SENDME cells until it receives $100 \cdot m$ data cells; any other cell is processed normally. Upon receiving $100 \cdot m$ data cells the client moves to the QUIT state.

QUIT

After receiving the $100 \cdot m$ data cells in the SLEEP state, the client quits the watermarking process and resumes the normal Tor SENDME process.

The watermarking process depends on two main variables: the number k of regular SENDME triplets that have to be sent by the client before embedding the watermark, and the watermark batch size m composing the watermark. The values given to these variables play an important role to the trade-off between accuracy and usability. We describe how we select the values of these variables in Subsection 3.4.

3.2 Detection and cancelling

A single detector consists of a modified Tor relay with guard capability that analyses every circuit passing through it. As any other relay in the circuit, the detector can only distinguish between IN and OUT cells (where IN cells are those going

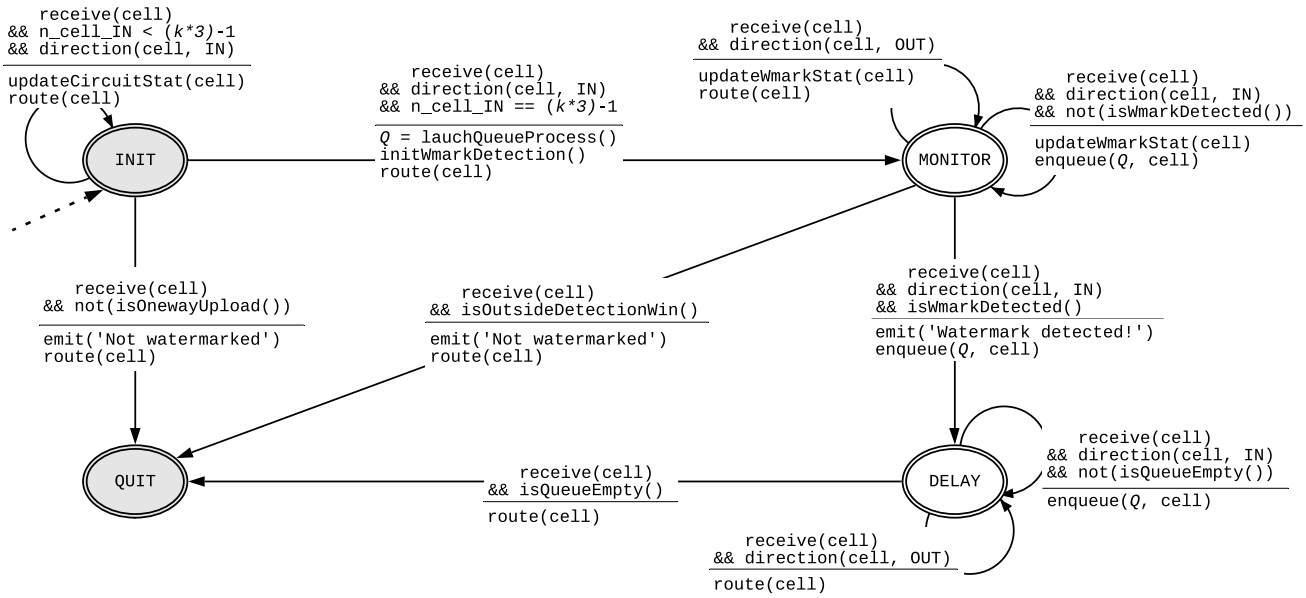


Figure 5: Finite-state machine of the detection and cancelling process.

towards the endpoint, and OUT cells those leaving the endpoint). When a new circuit is created, the detector checks that the IP address of the circuit's origin does not belong to the known list of Tor relays retrieved from the consensus document; if it does not, the detector is certain of being the guard relay for that circuit.[1] Assuming a circuit meets the initial filtering criteria, the detector starts the detection and cancelling process. Figure 5 depicts a simplified version of the finite state machine of the process, which consists of four states: INIT, MONITOR, DELAY, and QUIT states.

INIT

When the detector is in this initial state, it performs two tasks. The first consists of verifying whether the traffic in the circuit is a one-way upload data transfer.[2] This is enforced by the *isOnewayUpload* function that checks if the circuit statistics respect a 3-IN:100-OUT cell ratio. This ratio is a distinctive feature of a one-way upload data transfer and permits to filter out all of the interactive or one-way download circuits which are the vast majority of the circuits in Tor and are not relevant for our detection. The one-way upload circuits are the only ones useful to analyse since the adversary's Tor client is supposed to watermark only those circuits encapsulating onion content transfer (i.e., content flowing from the onion server to the client).

The second task is the computation of few bandwidth-dependant statistics required for the watermark detection. These statistics are updated by the *updateStatistics* function and consist of (i) the average, minimum, and maximum number of IN and OUT cells per second, and (ii) the average, minimum, and maximum number of IN and OUT cell batch sizes.

The detector remains in this state until $3 \cdot k$ IN cells are detected. Upon receiving the $(3 \cdot k)$-th IN cell, the detector starts the "core" detection process and calls the *initWmarkDetection* function which uses the statistics collected during the INIT state to compute the following detection parameters: (i) the observation time window d_{win} corresponding to the duration of the interval within which to look for the watermark, (ii) the minimum number $w_{th} \leq (3 \cdot m)$ of IN cells that are expected to be received for a successful detection, and (iii) the descriptors of the Markov model $\mathcal{M}$ characterising the IN/OUT cell alternating process.

Following, the detector invokes the *launchQueueProcess* function. The function launches a process which (i) creates a queue where the IN cells are temporarily buffered and (ii) emits the IN cells according to the model $\mathcal{M}$ in order to emulate the correct Tor IN/OUT cell timing process. This ensures that the SENDME cells belonging to the watermark batch are properly relayed (refer to Figure 3 which depicts the watermarking and cancelling process). In this way the watermark is cancelled, leaving the onion service unaware of the tracking process. Afterwards the detector moves to the MONITOR state.

[1] The detection process works even if the detector is a middle relay. However, the resulting IP address would point to the next relay in the circuit. This information can be used to pinpoint specific guard relays. If the onion service uses a bridge to connect to Tor, DUSTER will obtain the bridge's IP address, and this occurs in 2-3% of all circuits.

[2] We use the term "one-way upload/download" to indicate those circuits in which the traffic is highly unbalanced in favour of one of the two directions (upload or download, respectively).

MONITOR

In this phase the detector actively searches the watermark. It does this by analysing the circuit for an observation time window of duration d_{win}: if at least $w_{th} \leq 3 \cdot m$ IN cells are received within d_{win}, the circuit is considered watermarked. During this analysis, any received IN cell is not directly relayed but is instead forwarded to the queue process; the process takes care of the IN cells and emits them according to the model $\mathcal{M}$. To do that, the queue process needs to also receive information about the OUT cells routed by the relay. Upon receiving $3 \cdot m$ IN cells or at the end of the observation time window, if the watermark has been detected the detector moves to the DELAY state, otherwise it moves to the QUIT state.

DELAY

This state is coupled to the SLEEP state of the watermarker, and it can be reached only if the circuit has been labelled as watermarked. During this phase the onion service keeps sending OUT cells regularly. The OUT cells are observed by the detector and are used by the queue process to establish the timing of relaying the queued IN cells. Any additional IN cell that may arrive is still appended to the queue, since the IN cells must be forwarded in the order they come. All of the OUT cells are directly forwarded to the client. When all of the queued IN cells have been transmitted, the detector moves to the QUIT state.

QUIT

This is the final state and can be reached from both the MONITOR and DELAY states. If coming from the MONITOR state, the detector flushes eventually queued cells and destroys the queue process. If coming from the DELAY state, the detector destroys the already empty queue process. Afterwards it resumes routing cells normally.

As for the watermarking process, the detection depends on three variables: the duration of the observation time window d_{win}, the minimum number of IN cells required to detect the watermark w_{th}, and the set of descriptors of the Markov model $\mathcal{M}$ characterising the IN/OUT cell process. We describe how we select the values of these variables in Subsection 3.4.

3.3 Attack reasoning

There are some vulnerabilities in the Tor protocol which make this attack possible. On the client side, the core of the watermarking algorithm is the anticipated transmission of a batch of `SENDME` cells. This is possible as Tor does not perform any consistency or ordering control on the `SENDME` cells. In addition, the payload of a `SENDME` cell is composed by all zeros. This allows a malicious Tor client downloading a resource to craft and send all of the required `SENDME` cells in advance. The DUSTER attack exploits this vulnerability to anticipate a small batch of `SENDME` cells in a predefined manner: this generates a pattern which is propagated throughout the network with little variance, due to Tor's relay mechanism. For this reason the watermark can be identified by the detectors.

On the server side the `SENDME` cells are only used to acknowledge the correct reception of the data cells, and they are not authenticated.

Due to past vulnerabilities, the Tor community limited the maximum number of non-acknowledged data cells to 500 per stream and 1000 per circuit. If an onion service receives a `SENDME` cell acknowledging more than 500 data cells per stream or 1000 per circuit, it notices a protocol violation and drops the circuit. This means that if our *queueProcess* makes sure to never exceed this limit, the onion service won't drop the circuit or notice the attack. At the same time, if none of the detectors is the onion service's entry guard, the watermark won't be detected and cancelled. This implies the onion service will detect the `SENDME` anomaly and drop the circuit. The early interruption can be detected by the watermarker which can then move to the next onion address in the list.

3.4 System parameters

In this subsection we describe how the watermarking and detection parameters have been selected.

- Watermark batch size m. Number of `SENDME` triplets composing the watermark. A triplet is made of 3 `SENDME` cells (stream, circuit, and stream cells) which acknowledge 100 data cells received in the stream.[3] The value of m can be tuned in order to achieve the desired trade-off between accuracy and detection time. In addition, the lower the value of m, the lower is the detection rate but the higher is the probability of finding a resource exceeding the minimum size required to embed the watermark. At the same time, a low value of m corresponds to a shorter interval during which the watermarker interferes with the circuit. We use m as a free parameter and we examine it in Section 4.

- Number k of monitored triplets. k represents the number of `SENDME` triplets that must be sent by the client and seen by the detector before running the "core" watermark embedding and detection, respectively. This waiting interval is necessary as it gives time to the detector to (i) recognise and filter out all of the circuits that do not convey a one-way upload data transfer, and (ii) collect the

[3] For simplicity we opted to always work in triplets of `SENDME` cells, which can be easily mapped to 100 data cells in the opposite direction. However one can choose any number of `SENDME` cells for both k and m.

circuit statistics needed by the detection and cancelling phase. The value of k can be tuned to achieve the desired trade-off between the accuracy of the collected statistics and the probability of finding a resource exceeding the minimum size required to embed the watermark. The greater the value of k the more accurate are the collected statistics, but the lower is the probability of finding a resource exceeding the minimum size to embed the watermark.

- Minimum content length L_{min}. Minimum content length required to embed the watermark onto the corresponding circuit. The value is computed as $L_{min} = 100 \cdot (m+k) \cdot L_c$ where L_c is the cell's payload length, that is $L_c = 498$ bytes.

- Detection size w_{th}. Minimum number of IN cells that must be received within the interval d_{win} to label the circuit as watermarked. w_{th} must be lower than or equal to $3 \cdot m$, and its value can be tuned in order to achieve the desired trade-off between true positive and false positive rates. The lower w_{th}, the higher the true positive and false positive rates.

- Duration of the observation time window d_{win}. Duration of the time window to search for the watermark. As for w_{th}, d_{win} can be tuned to achieve the desired trade-off between true positive and false positive rates. The higher d_{win}, the higher the true positive and false positive rates.

- Markov model $\mathcal{M}$. The model consists of an empirically computed Markov process that describes the probability of emitting the first IN cell from the queue, given that (i) the queue is not empty, (ii) b OUT cells have been routed (with $0 \leq b \leq 500$), and (iii) a time interval δ has elapsed since the last OUT cell. In our tests, a base model was first pre-computed from the IN/OUT cell statistics collected by a Tor guard relay over two days of logging at 1+Mbps. Then, during the INIT state of the detection process, each detector updates its model by tweaking the probabilities based on the updated traffic statistics of the analysed circuit.

4 System evaluation

We tested the attack on the real Tor network by setting up a guard relay on Amazon Web Services (AWS), an HTTPS onion service on a dedicated server, and a Tor client on a personal computer. We used an up-to-date Tor implementation (Tor version 0.3.2.10). The implementation was modified to perform the watermark detection on the entry guard and the watermark embedding on the client. For testing purposes, we configured our onion service to always select our testing relay as its entry guard.

A single test consisted of the client performing an HTTPS file download from the onion service with the detector trying to infer the presence of the watermark. Depending on the type of test, the client did or did not embed the watermark during the transfer. To ensure that different relays for circuit building were always chosen, we rebooted both the onion service and client after each test. Thanks to the worldwide distribution of AWS, we also moved our relay over three different locations during our experiments.

4.1 Numerical results

The plots in Figures 6–11 present the true positive (TPR – watermark present and correctly detected) and false positive (FPR – watermark detected although not present) rates obtained from our experiments. The results are shown by varying the parameters m, d_{win}, and w_{th}. Each value in the TPR plots is computed over 200 instances of experiments with the watermarker activated, while each value in the FPR plots is computed over 200 instances with the watermarker deactivated.

Experiments were carried out by setting $k = 15$, which entails that at least 1500 data cells (about 750 KB) are to be transferred from the onion service to the client before starting the watermarking process. Any variation of this parameter does not have an impact on the accuracy of the detection process.

In greater detail, the results in Figures 6 and 7 show the trend of TPR and FPR as a function of the ratio $u = \frac{d_{win}}{\phi(m)}$ between the observation window and the estimated time $\phi(m)$ to receive m IN cell triplets, providing different curves for different values of m. The estimation of $\phi(m)$ is based on the statistics collected from the detector during the INIT phase. As expected, both TPR and FPR increase when increasing the observation window; results show that for values of u between 0.125 and 0.5 we can get a good trade-off between TPR and FPR.

Similarly, the results in Figures 8–11 show the trend of TPR and FPR as a function of the ratio $t = \frac{w_{th}}{3 \cdot m}$ between detection size and watermark size, for two values of the observation window ($d_{win} = 0.25 \cdot \phi(m)$ and $0.5 \cdot \phi(m)$, respectively) and providing different curves for different values of m.

We observe that the TPR decreases when increasing the watermark detection size; it falls below 0.75 for t greater than 0.4, with $d_{win} = 0.25 \cdot \phi(m)$ (Figure 8). At the same time, we obtain FPR values close to zero when t is equal to or greater than 0.4. For a wider observation window ($d_{win} = 0.5 \cdot \phi(m)$) the TPR decreasing trends are slightly slower (Figure 10).

Observing the results as a whole, they demonstrate the strength of the DUSTER attack. This is also confirmed by the ROC function in Figure 12 in which the plotted ROC curve is close to the upper left corner of the graph. The ROC curves are obtained by varying u in the range $[0.125; 2]$ and w_{th} in $[0.2; 1]$, for different values of m. Considering the best

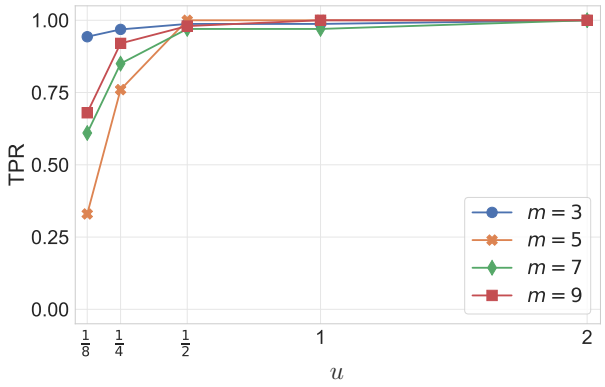


Figure 6: TPR as a function of d_{win} for different values of m and with $t = 0.4$.

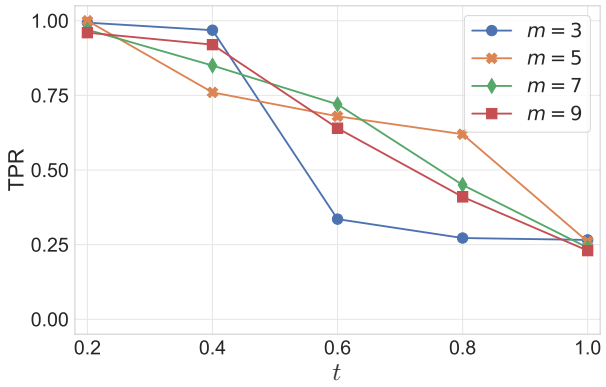


Figure 8: TPR as a function of w_{th} for different values of m and with $u = 0.25$.

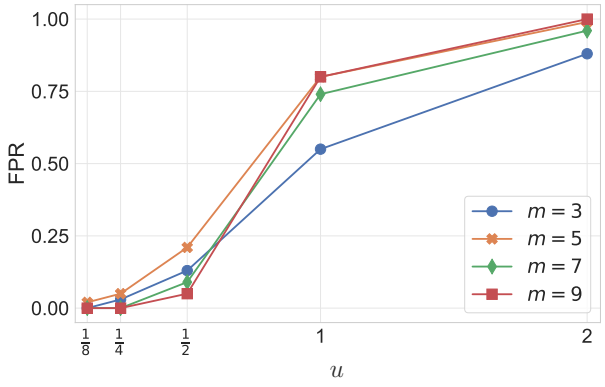


Figure 7: FPR as a function of d_{win} for different values of m and with $t = 0.4$.

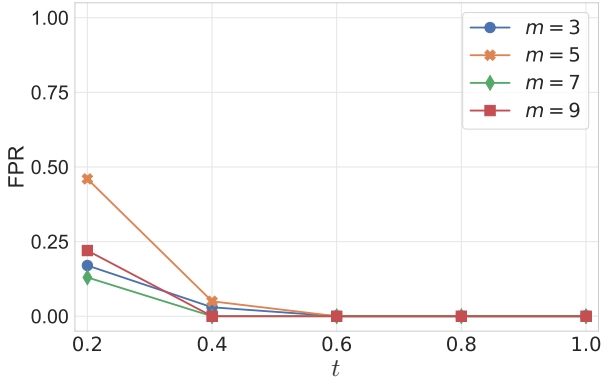


Figure 9: FPR as a function of w_{th}, for different values of m and with $u = 0.25$.

combination of variables, that is $m = 5$, $w_{th} = 9$, and $u = 0.5$, we are able to obtain TPR equal to 0.98 with FPR equal to 0.03. We can reduce the FPR to 0.01 by selecting $m = 7$, $w_{th} = 13$, and $u = 0.5$ while maintaining the TPR above 0.94. We do not report results for $m < 3$ as a minimum of $m = 3$ is required to distinguish the watermark from natural cell batches.

For completeness, we have also verified the FPR assuming a real deployment scenario in which the detector analyses each relayed circuit. For the purpose we allowed the detector to analyse each circuit passing through it for few hours; of the approximately $2.5 \cdot 10^5$ relayed circuits, none were incorrectly labelled as watermarked. In the experiment, the detector was instantiated with $m = 5$, $w_{th} = 9$, and $u = 0.5$.

Finally, to analyse the effect of the watermark cancelling process, Figure 13 presents a statistical comparison between watermarked and un-watermarked circuits as observed by the onion service based on $2 \cdot 10^5$ statistical samples. The plots show the joint histogram of (i) the SENDME batch size along the ordinate and (ii) the distance between consecutive SENDME batches along the abscissa. The figure shows that the histograms of the watermarked circuits nearly match those of the un-watermarked circuits. This is confirmed by the Wasserstein distance[4] computed between the two joint histograms, which is equal to $0.98 \cdot 10^{-5}$.

4.2 Attack applicability

The DUSTER attack, like the majority of known TA attacks against Tor, requires the attacker to control the guard relay of a circuit. The Tor community has put significant effort into mitigating these types of targeted attacks, mostly by applying strategies in which only few random entry guards are selected by each Tor endpoint, and they are kept for long periods of time. Nonetheless, the probability of being selected as a

[4]The Wasserstein distance returns a lower bound of zero in the case of two identical distributions and an upper bound of one in the case of disjoint distributions.

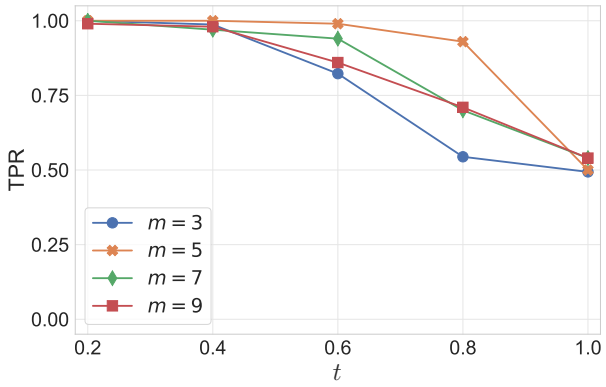


Figure 10: TPR as a function of w_{th} for different values of m and with $u = 0.5$.

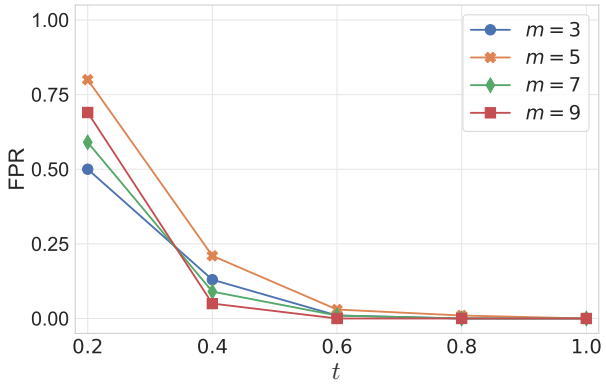


Figure 11: FPR as a function of w_{th} for different values of m and with $u = 0.5$.

guard by an endpoint is not negligible and this entails that the attack is successfully able to de-anonymize a number of onion services [13]. This number can be roughly estimated as the ratio $(\alpha \cdot D \cdot S)/G$ in which, given a generic instance of crawling, S is the number of onion services, D is the total guard bandwidth offered by the detectors controlled by the attacker, G is the total guard bandwidth offered by all of the entry guards, and α is the probability of the onion service being reached by the attacker's crawler and providing at least a file to download of size greater than L_{min}. The values of S and G can be estimated from the metrics provided by Tor. For example, for values of $S = 10^5$, $G = 100$ *Gbps*, $D = 1$ *Gbps*, and $\alpha = 0.01$, the attacker would be able to de-anonymize about 10 onion services during an instance of crawling.[5]

We presented and analysed the DUSTER attack against Tor's rendezvous circuits because it is, in our opinion, the most compelling case. Nevertheless, the attack exploits a vulnerability of the congestion control mechanism; this means that it can also target standard circuits. In such cases, the purpose of the attack would be to correlate the flows of a Tor client with its visited websites. This scenario requires the watermarker implemented on the exit relay and the detector placed onto the client's entry guard.

[5]The approximation assumes that the average bandwidth offered by a guard relay controlled by the attacker is the same as the bandwidth averagely offered by a regular guard relay.

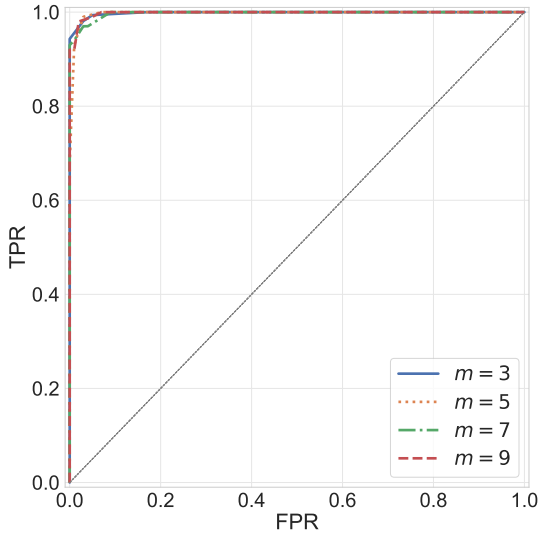


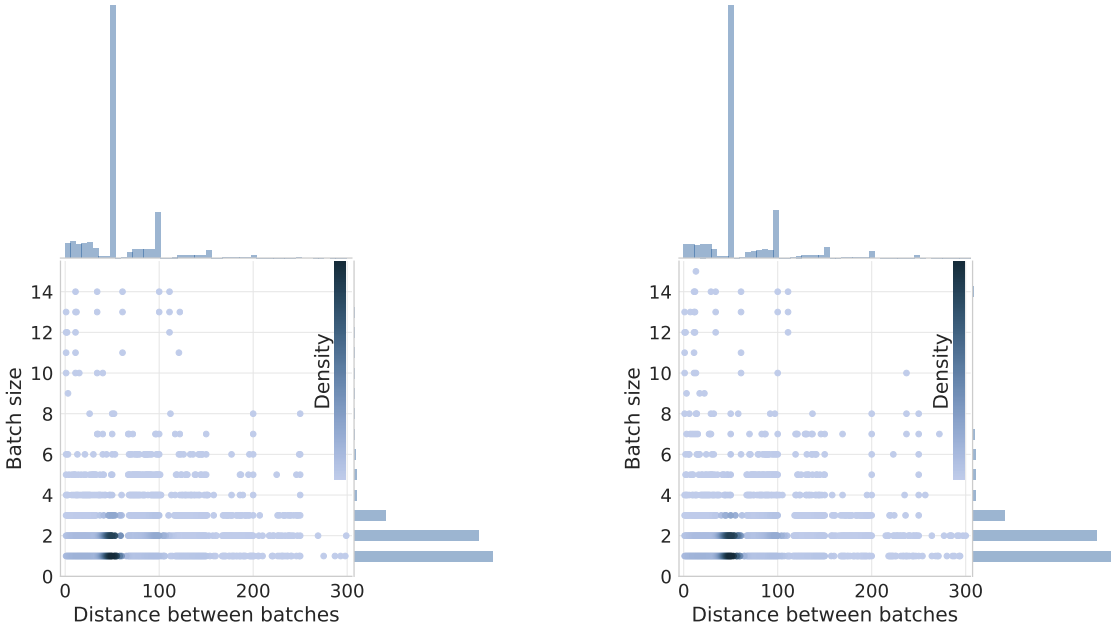
Figure 12: ROC curves for different values of m.

4.3 User safety and ethics

The Tor research board published a set of safety guidelines that researchers should respect to preserve legitimate users' anonymity [2]. We relied on the guidelines to mount our experiments in order to (i) always work with only our modified onion service, client, and entry guard, (ii) test and prototype the implementation on the shadow simulator [10], (iii) actively modify only the traffic going to/from our onion service during active experiments, (iv) log and store only the minimal amount of data required for our analysis (in our case cells' timing and direction). Finally, we disclosed the attack to Tor community before the publishing of this paper.

5 Attack mitigation

In the current Tor implementation, a stream is end-to-end encrypted using AES in counter block mode, which mitigates forgery and/or reordering attacks. However the congestion control system is predictable and can be exploited by a malicious client to anticipate `SENDME` cells and use them, as in DUSTER, to embed a detectable pattern in the traffic flow. This pattern can be analysed by detectors strategically placed in the network to de-anonymize the communicating parties.



(a) Un-watermarked

(b) Watermarked

Figure 13: Tor traffic statistics.

Tor has had a history of attacks against its congestion control system. After each vulnerability disclosure the community implemented specific countermeasures. However, these countermeasures have never really solved the underlying problems and the congestion control system remains vulnerable. In fact, although we have implemented DUSTER proof of concept in Tor version 0.3.2.10 (March 2018), there was no countermeasure implemented until May 2019.

A possible solution to mitigate DUSTER and other attacks targeting Tor's congestion control consists of a protocol ensuring that (i) the receiver computes a checksum of the acknowledged data cells and appends it to the payload of each SENDME cell, (ii) the sender verifies the checksum and drops the circuit if the checksum is incorrect, and (iii) both parties make use of some randomness to avoid reply attacks.

There has been a related proposal in late 2016 in which the authors suggested to integrate a SENDME authentication scheme in Tor [9]. As the referred document describes the protocol, requirement and guarantees, we won't discuss it here. The Tor community released Tor version 0.4.1.1-alpha on May 22, 2019; this version incorporates the SENDME authentication scheme proposed in [9]. We inspected the implementation and verified that it is able to mitigate the DUSTER attack. However, the unauthenticated SENDME cells are expected yet to be supported until 2022 for backward compatibility, according to the documentation of the release. This entails that the vulnerability will be fully exploitable as long as unauthenticated SENDME cells will be accepted by Tor nodes. In fact, a malicious client will always be able to force an onion service to downgrade to the unauthenticated congestion control mechanism.

6 Conclusions

In this paper we present DUSTER, an attack against Tor's server-receiver anonymity. The attack is based on network flow watermarking and exploits a vulnerability in Tor's congestion control mechanism. We implemented and evaluated the DUSTER attack on the real Tor network, and demonstrated that whenever a detector intercepts a watermarked circuit it can detect the watermark with a true positive rate up to 98%, while the false positive is close to zero; This enables the attacker to link the downloaded content with the IP address of the onion service. Further, as the watermark is cancelled by the detector, the onion service remains unaware of the tracking.

Finally we mention that, despite the partial countermeasures implemented over time, the underlying congestion control system remains vulnerable to the DUSTER attack and some possible other attacks. We stress that the community should deploy the solution proposed in [9] to once and for all mitigate attacks targeting Tor's congestion control system.

References

[1] Tor protocol specs. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt.

[2] Tor research safety board. https://research.torproject.org/safetyboard/.

[3] Marco Valerio Barbera, Vasileios P Kemerlis, Vasilis Pappas, and Angelos D Keromytis. Cellflood: Attacking tor onion routers on the cheap. In *European Symposium on Research in Computer Security*, pages 664–681. Springer, 2013.

[4] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *IEEE Symp. on Security and Privacy*, pages 80–94, 2013.

[5] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 92–102. ACM, 2007.

[6] Norman Danner, Sam Defabbia-Kane, Danny Krizanc, and Marc Liberatore. Effectiveness and detection of denial-of-service attacks in tor. *ACM Transactions on Information and System Security (TISSEC)*, 15(3):11, 2012.

[7] Shalini Ghosh, Ariyam Das, Phil Porras, Vinod Yegneswaran, and Ashish Gehani. Automated categorization of onion sites for analyzing the darkweb ecosystem. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1793–1802. ACM, 2017.

[8] Alfonso Iacovazzi and Yuval Elovici. Network flow watermarking: A survey. *IEEE Commun. Surveys & Tutorials*, 19(1):512–530, 2017.

[9] Rob Jansen and Roger Dingledine. Authenticating sendme cells to mitigate bandwidth attack. https://github.com/torproject/torspec/blob/master/proposals/289-authenticated-sendmes.txt.

[10] Rob Jansen and Nicholas Hooper. Shadow: Running tor in a box for accurate and efficient experimentation. Technical report, 2011.

[11] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. In *NDSS*, 2014.

[12] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security*, volume 20, 2015.

[13] Z. Ling, J. Luo, K. Wu, and X. Fu. Protocol-level hidden server discovery. In *Proc. IEEE INFOCOM*, pages 1043–1051, 2013.

[14] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. A new cell counter based attack against tor. In *Proc. ACM Conf. on Comput. and Commun. Security*, pages 578–589, 2009.

[15] Nick Mathewson. Denial-of-service attacks in Tor: Taxonomy and defenses. Technical report, The Tor Project, 2015.

[16] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proc. ACM SIGSAC Conf. on Comput. and Commun. Security*, pages 2053–2069, 2017.

[17] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. How unique is your .onion?: An analysis of the fingerprintability of tor onion services. In *Proc. ACM SIGSAC Conf. on Comput. and Commun. Security*, pages 2021–2036, 2017.

[18] Lasse Overlier and Paul Syverson. Locating hidden servers. In *IEEE Symp. on Security and Privacy*, pages 15–pp, 2006.

[19] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. Analysis of fingerprinting techniques for tor hidden services. In *Proc. on Workshop on Privacy in the Electron. Soc.*, pages 165–175, 2017.

[20] Vasilis Pappas, Elias Athanasopoulos, Sotiris Ioannidis, and Evangelos P Markatos. Compromising anonymity using packet spinning. In *International Conference on Information Security*, pages 161–174. Springer, 2008.

[21] Florentin Rochet and Olivier Pereira. Dropping on the edge: Flexibility and traffic confirmation in onion routing protocols. In *Proc. on Privacy Enhancing Technologies*, pages 27–46, 2018.

[22] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proc. on Workshop on Privacy in the Electron. Soc.*, pages 201–212, 2013.

A Appendix

We preset some simplified pseudo-code useful for the understanding of the paper. The code assumes each circuit has only one stream: in the real Tor network a circuit might have one or more data streams. Listing 1 and 2 present the use of Tor's SENDME cells and the client download process, respectively. Listing 3 describes how a Tor onion service sends data and validates SENDME cells. Listing 4 describes the DUSTER watermarker which substitutes the default *tor_considerSendingSendme* showed in Listing 1.

Following, Listing 5 shows how a Tor guard relay routes cells. Listing 6 shows the DUSTER watermark detector which replaces the original *tor_routeIncomingCell* in Listing 5.

```
"""
  - circ: Tor circuit
  - cell: Tor cell
"""
def tor_considerSendingSendme(circ, cell):
   if (circ.n_cell % 100) == 0:
      tor_sendCircuitSendme(circ)
   if (circ.stream.n_cell % 50) == 0:
      tor_sendStreamSendme(circ.stream)
```

Listing 1: Tor SENDME handing function.

```
"""
  - addr: the onion service address.
  - res:  the resource to be fetched.
"""
circ = tor_openRendezvousCircuit(addr)
circ.stream = tor_getResource(circ, res)
circ.n_cell = 0
circ.stream.n_cell = 0
while True:
   cell = tor_getNextCell(circ.stream)
   if tor_isEndCell(cell):
      break
   if tor_isDataCell(cell):
      tor_commitPayload(cell)
      circ.n_cell += 1
      circ.stream.n_cell += 1
      tor_considerSendingSendme(circ, cell)
tor_closeCircuit(circ)
```

Listing 2: Tor client resource download process.

```
circ = tor_acceptConnection()
circ.stream, req_res = tor_handleRequest()
bin_data = tor_fetchResource(req_res)
circ.cwin = 1000
circ.stream.cwin = 500
while True:
   # Send data if congestion windows are not empty.
   if circ.stream.cwin > 0 and circ.cwin > 0:
      wr = tor_sendDataCell(circ.stream, bin_data)
      bin_data = bin_data[wr:]
      circ.cwin -= 1
      circ.stream.cwin -= 1
   # Send end-cell if transmission is completed.
   if len(bin_data) == 0:
      tor_sendEndCell(circ.stream)
      break
   # If a cell from the client has been received
   if tor_isCellQueued():
      cell = tor_getNextCell(circ.stream)
      if tor_isCircuitSendmeCell(cell):
         circ.cwin += 100
      if tor_isStreamSendmeCell(cell):
         circ.stream.cwin += 50
   # Sanity checks to mitigate previous attacks.
   if circ.cwin > 1000 or circ.stream.cwin > 500:
      tor_print("Unexpected SENDME. Dropping circ.")
      break
tor_closeCircuit(circ)
```

Listing 3: Onion service client handling.

```
"""
  - circ: Tor circuit
  - cell: Tor cell
Duster parameters:
  - {k, m}: attack variables.
  - {INIT, SLEEP, QUIT}: Duster states.
"""
def duster_considerSendingSendme(circ, cell):
   duster = duster_getOrCreateNew(circ)
   duster.n_cell += 1
   # Default behaviour until K*100 received.
   # Then send m*3 SENDME cells.
   if duster.state == INIT:
      tor_considerSendingSendme(circ, cell)
      if duster.n_cell == k*100:
         for i in range(M):
            tor_sendStreamSendme(circ.stream)
            tor_sendCircuitSendme(circ)
            tor_sendStreamSendme(circ.stream)
         duster.n_cell = 0
         duster.state = SLEEP
   # Wait m*100 data cells before moving to QUIT.
   elif duster.state == SLEEP:
      if duster.n_cell == m*100:
         duster.state = QUIT
   # Attack finished. Default Tor.
   else:
      assert(duster.state == QUIT)
      tor_considerSendingSendme(circ, cell)
```

Listing 4: DUSTER watermarker.

```
"""
  - circ: Source circuit.
  - cell: Tor cell.
"""
def tor_routeIncomingCell(circ, cell):
   dest_circ = tor_getDestinationCircuit(circ)
   tor_route(dest_circ, cell)
```

Listing 5: Onion service's entry guard routing.

```
"""
  - circ: Source circuit.
  - cell: Tor cell.
Duster parameters:
  - {k, m}: attack variables.
  - {INIT, MONITOR, DELAY, QUIT}: Duster states.
"""
def duster_routeIncomingCell(circ, cell):
   dest_circ = tor_getDestinationCircuit(circ)
   duster = duster_getOrCreateNew(circ)
   duster.updateCircuitStat(cell)
   # Wait until 3*k IN-cells are seen.
   # Meanwhile validate the circuit.
   if duster.state == INIT:
      if not duster.isOnewayUpload()
         duster.state = QUIT
         break
      if duster.n_cell_IN == (3*k):
         duster.state = MONITOR
         duster.launchQueueProcess()
         duster.initWmarkDetection()
      tor_route(dest_circ, cell)
   # Search for the watermark. Buffer IN cells
   # and route OUT cells normally.
   elif duster.state == MONITOR:
      duster.updateWmarkStat(cell)
      if tor_direction(cell, IN):
         duster.enqueue(cell)
      else:
         tor_route(dest_circ, cell)
      if duster.isWmarkDetected():
         duster.print("WATERMARK DETECTED!!!")
         duster.state = DELAY
      elif duster.isOutsideDetectionWin(cell):
         duster.state = QUIT
         break
   # Buffer IN cells until queue empties.
   elif duster.state == DELAY:
      if duster.isQueueEmpty():
         duster.state = QUIT
         break
      elif tor_direction(cell, IN):
         duster.enqueue(cell)
      else:
         tor_route(dest_circ, cell)
   # Default Tor routing.
   else:
      assert(duster.state == QUIT)
      tor_route(dest_circ, cell)
```

Listing 6: DUSTER watermark detector.