Florentin Rochet* and Olivier Pereira

Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols

Abstract: The design of Tor includes a feature that is common to most distributed systems: the protocol is flexible. In particular, the Tor protocol requires nodes to ignore messages that are not understood, in order to guarantee the compatibility with future protocol versions. This paper shows how to exploit this flexibility by proposing two new active attacks: one against onion services and the other against Tor clients.

Our attack against onion services is a new low-cost side-channel guard discovery attack that makes it possible to retrieve the entry node used by an onion service in one day, without injecting any relay in the network. This attack uses the possibility to send dummy cells that are silently dropped by onion services, in accordance with the flexible protocol design, and the possibility to observe those cells by inspecting public bandwidth measurements, which act as a side channel.

Our attack against Tor clients, called the dropmark attack, is an efficient 1-bit conveying active attack that correlates flows. Simulations performed in Shadow show that the attack succeeds with an overwhelming probability and with no noticeable impact on user performance.

Finally, we open the discussion regarding a trade-off between flexibility and security in anonymous communication systems, based on what we learned within the scope of our attacks.

Keywords: Onion Routing, Tor, Traffic Confirmation, Guard Discovery Attack, Protocol Flexibility

DOI 10.1515/popets-2018-0011
Received 2017-08-31; revised 2017-12-15; accepted 2017-12-16.

***Corresponding Author: Florentin Rochet:** UCLouvain, ICTEAM, Crypto Group, B-1348 Louvain-la-Neuve, Belgium. E-mail: florentin.rochet@uclouvain.be

Olivier Pereira: UCLouvain, ICTEAM, Crypto Group, B-1348 Louvain-la-Neuve, Belgium. E-mail: olivier.pereira@uclouvain.be

1 Introduction

Onion Routing offers an application-independent communication infrastructure on top of a public network, which offers some degree of anonymity and traffic analysis resistance [28, 45, 50, 51]. These security features are obtained by routing communications through a set of relays called Onion Routers. As of today, Tor [22] is the most popular Onion Routing implementation: several millions of users communicate through Tor every day, and it is investigated by a dynamic research community, which produced hundreds of related scientific publications.

Tor is a low-latency network aiming to separate identification from routing. Among other anonymizer proposals [2, 19, 20, 38, 42, 48, 55], Tor got popular mainly due to its ease of deployment, ease of use, high performance, censorship circumvention and active development. Tor is distributed, and any volunteer can contribute by running relays. As a consequence of this openness, it is accepted that routers running different versions of the protocol co-exist in the network: the Tor protocol is forward compatible, in the sense that additions to the protocol do not break old implementations: the packets, or cells, that are not understood are just ignored or, in practice, dropped. This forward compatibility offers a form of flexibility in the Tor protocol that is present since Tor's earliest version.

In this paper, we focus on the Tor implementation and show how choices that have been made in order to ensure forward compatibility enable a cheap targeted guard discovery attack and an efficient active end-to-end correlation attack. We show that the decision to drop unintelligible packets may harm anonymity by opening the way to two active attacks: a novel guard discovery attack and yet another end-to-end correlation attack that shows interesting properties, motivating our investigation. This result is demonstrated for the Tor network, but the same weaknesses are likely to be found in any anonymous communication system that routes packets through circuits and tries to defend against end-to-end traffic correlation attacks.

Our contributions

In this paper, we show experimentally how forward compatibility in anonymous communication protocols can threaten anonymity. More precisely:

- We shed light on conflicting interactions between two desired properties of low-latency anonymous network protocols: flexibility and resilience against cheap traffic confirmation.
- We design a new guard discovery method that makes it possible to retrieve the guard relay of any onion service in one day without running any relay in the network. This guard discovery method is a two-phase process: in the first phase, a modified Tor client creates many circuits towards the onion and sends cells that are dropped at the circuit layer by the onion service. In the second phase, a side-channel confirms the identity of the onion service's guard based on the public measurements of the relayed bandwidth. We use OnionShape [9], a tool that we developed to analyze relay bandwidth measurements. Based on our results, we discuss the appropriate timing between two reported bandwidth measurements.[1]
- We discover and report a design flaw in the Tor cell counter implementation, which amplifies the effectiveness of our guard discovery attack. The Tor project acknowledged this design flaw.
- We design and implement a new efficient active attack called the *dropmark* attack, that supports flow correlation in the Tor network. This attack is invisible to anyone running the current Tor code but could be detected if the code were adapted. We argue that this attack is part of a new family of active end-to-end correlation attacks that take advantage of the flexibility of the Tor protocol. Because of the flexibility that we would like to keep, this attack is difficult to mitigate.
- We evaluate the effectiveness of the dropmark attack using the Shadow simulator [33], and obtain results under an ethical test-bed that would be similar to the real Tor network. The dropmark attack has true positive success rate of $\approx 99.86\%$ probability and a false positive rate of $\approx 0.03\%$. Compared to passive attacks, it has some interesting properties: 1) It does not need the victim to transfer any packet to succeed. 2) The activity of the victim at the application level does not influence the success rate of the attack (i.e., we do not miss users whatever is happening in their applications).
- Based on our results, we open a discussion for a further research direction regarding a flexibility/security trade-off in anonymous communication systems.

Roadmap

We start by reviewing the related works on congestion attacks and end-to-end traffic correlation in Section 2. We provide the necessary background on the Tor protocol in Section 3 and we detail how the packet dropping behavior of the Tor protocol can jeopardize anonymity in Section 3.4. We exploit this feature to create a new guard discovery attack in Section 4 and a new active traffic confirmation attack in Section 5. Based on those new attack channels, we discuss the risks of flexibility in Section 6. Section 7 points to the code and data that we make available to support the replication of our experiments, and Section 8 concludes.

2 Related work

2.1 Congestion attacks

Congestion attacks are a well-known technique to perform guard discovery of Tor users and onion services. The general idea is to play a two-step attack. The first step consists in applying a clogging attack [16] variant: sending lots of data to saturate a relay. The second one resides in observing whether the effect of this clogging is visible on the channel between the adversary and the target (a Tor circuit in our case). If the effect can be observed, then the adversary knows that this relay is part of the target's Tor circuit. Murdoch and Danezis [40] provided experimental proofs on the 2005 Tor network that it is possible to reconstruct a Tor user's circuit without directly observing network links.

Later, Evans *et al.* [24] showed that this deanonymization method is not achievable anymore given that the network has grown with more relays and users. They proposed an improved variant of Murdoch and Danezis's clogging attack based on the observation that the Tor protocol does not prevent the creation of circuits of infinite length. Pappas *et al.* [43] presented a similar congestion attack creating looping circuits where the malicious relay is not involved in cryptographic oper-

1 Each relay reports its average relayed bandwidth each day, in 4-hour intervals

ation and spins continuously the same packets. In the meantime, Hopper *et al.* [29] used an estimation of link-latency and node-latency of Tor circuits, in combination with the Murdoch and Danezis clogging approach. Their approach allows them to determine the Tor client's network location with increased precision as the victim interacts with the adversary's web server. Instead of latency estimation, Chakravarty *et al.* [21] used a single-end controlled available bandwidth estimation tool to observe the effect of a perturbation performed by a colluding relay.

Our congestion attack, described in Section 4.1, can also be considered as a variant of the original clogging attack of Murdoch and Danezis and aims at overloading one relay. We use our congestion strategy along with the observation of its effect over a side-channel in order to identify the guard relay of an onion service.

2.2 End-to-end correlation

End-to-end timing [37, 52] and traffic analysis attacks such as packet counting [47] are part of the few unresolved issues listed in AlSabah and Goldberg's survey [15], that impact low-latency anonymizers such as the Tor network. Different models of realistic attackers have been extensively studied in the literature. Local passive adversaries that control a part of the network, like Autonomous Systems (ASes) or Internet Exchange Points (IXPs), can circumvent Tor's anonymity [17, 23, 25, 41].

Apart from these passive attacks, adversaries can also increase the confidence in their correlation attack by becoming active. Active adversaries induce perturbations of a traffic flow: they make a particular flow different from the inherent pattern of other network flows, which highly decreases the risk of false positives when correlating flows. These techniques are usually referred to as watermarking [30, 32, 53, 54]. A watermark encodes only one bit of information, which prevents the distinction between several watermarks that would be observed at the same time. Biryukov *et al.* [18] use a watermark technique during the circuit construction to reveal the onion service's IP. Houmansadr and Borisov [31] push the concept of watermarking further and define the notion of flow fingerprint as a set of techniques that can be used to encode and decode a sequence of watermarks (carrying multiple bits of information).

To reduce these threats, two types of approaches are often considered. The first approach consists in designing counter-measures that hide the traffic characteristic used to match streams [39, 49], while the second approach aims at increasing the diversity in the Tor network [17, 46].

Regarding Tor, watermarking and flow fingerprint require to relay a sufficient amount of data to embed the watermark(s). In practice, the adversary must carefully choose the timing interval in the watermark encoding, in order to avoid the Tor user to induce a circuit rotation when he detects latency issues. Naturally, an adversary would be interested in attacks that are not subject to those limiting requirements. A previous (unpublished) study, documented as the relay-early traffic confirmation attack [3] achieved that by injecting (instead of delaying) a crafted sequence of special cells. Then, the sequence can be decoded on the path by a colluding relay as it flows through the circuit. This attack was stronger than previous active attacks because it would be performed without information loss, and because it eliminated the latency issue inherent to the delaying approach. When the relay-early confirmation attack has been used in practice to deanonymize Tor users, the Tor project reacted by detecting and removing the Sybil relays from the network and produced a software update preventing *relay-early* cells to be used in this way.

In Section 5, we present a new watermarking scheme that bears some similarities with the relay-early confirmation attack, and takes advantage of the flexibility in the Tor protocol.

3 More Background on Tor

In this section, we depict a high-level explanation of the innermost Tor security design: the Tor protocol [1]. We focus on the interaction between Tor nodes, on the way communication is structured, and on the way circuits are built and destroyed.

3.1 Components and interaction overview

Tor is a distributed overlay network with the same code base for all of its components, namely: clients, relays, directories and hidden services. All of these components can be seen as event-driven network daemons that are tied together by circuits. Those circuits are established with public-key cryptography and handle cells, the unit of communication of the Tor network. Those fixed-size cells are forwarded along the circuits carrying layers of

encryption. Each component of the circuit is responsible for peeling one layer (unwrapping by a symmetric cipher) or adding one, depending on the direction of the cells within the circuit. An outgoing direction causes a component of the circuit to peel a layer and forward the cell. Conversely, an incoming direction causes a component to add a layer before it forwards the cell. This behavior is called Onion Routing. In the Tor protocol, each cell is addressed to an edge relay, which is by definition, the relay able to recognize the cell. Every component of a circuit is an edge relay for some cells, at some points (e.g., during the construction of the circuit). Most of the time, cells are recognized at the endpoints of Tor circuits because it is the intended behavior when carrying application data. We call those cells, the *circuit-level* cells. The remaining cells are used for peer-to-peer management such as initializing connections, creating circuits or destroying them. These are called *link-level* cells.

3.2 Communication unit: cells

Link-level cells and circuit-level cells have two different formats (Figure 1). The fields **CircID** and **Cmd** are not encrypted while the remaining bytes might be encrypted several times, depending on the number of relays in the circuit. All circuit-level cells have the same **Cmd** value (set to "Relay") but not the same **Rel cmd** value (the second **Cmd** field in Figure 1), defining the subtype of circuit-level cell. There are many subtypes of circuit-level cells, but they are not distinguishable by any other than the edge node. If circuit-level cells had different possible value in the **Cmd** field, the anonymity provided by Tor could be trivially broken. Indeed, an attacker could encode any message with an alternative composition of circuit-level cells of different **Cmd** value (e.g., binary encoding of an ascii message). Such a problem arose when the *relay-early* cell was designed in order to avoid the possibility to build infinite-length Tor circuits and to constrain them to a reasonable maximum size [3]. This particular circuit-level cell had a **Cmd** value different from other circuit-level cells, allowing high-confidence correlation.

3.3 Circuit construction

A Tor software in client mode constructs and destroys circuits in the same way: incrementally. It does the construction incrementally in order to protect its identity while negotiating symmetric keys with each relay, using

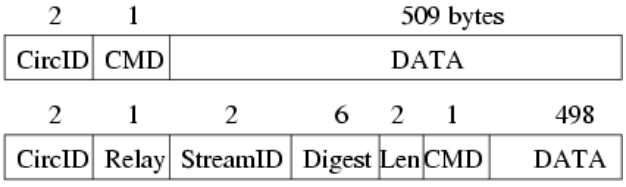


Fig. 1. Link-level cells and circuit-level cells structure. Image from [22].

a one-way authenticated key-exchange protocol called *NTor* [27], in which only relays are authenticated. The Tor client and each hop exchange a few cells to establish the circuit. The Tor client (let us call it Alice) starts by sending a link-level *create* cell towards the first relay (let us call it OR1). OR1 responds with a *created* cell, and the crypto material contained in both cells are used to derive the symmetric keys (one for each direction). At this point, the established circuit contains one hop. To extend the circuit, Alice sends a circuit-level *extend* cell symmetrically encrypted towards OR1 with the necessary information to contact the next relay (let us call it OR2). Then, OR1 proceeds like Alice in the first step: it sends a *create* cell towards OR2 with the crypto material from Alice. *NTor* prevents OR1 from reconstructing the shared key and also prevents an active man-in-the-middle attack. In order to extend the circuit towards a third relay or further, Alice plays the same game with the edge relay of the circuit: telling it where to send a *create* cell and waiting for the *extended* cell going backward. The main difference with the third relay extension is the fact that *OR1* does not recognize the cell. Instead, the cell is wrapped with *OR1*'s key material and forwarded.

In order to destroy a circuit, Alice sends a link-level *destroy* cell towards the first hop, which will be echoed along the circuit. This process can be seen as an incremental destruction starting from the beginning of the circuit. Alice can also destroy part of the circuit using a circuit-level *truncate* cell encrypted with a number of layers according to the receiver (2 for the second, 3 for the third, etc.). Upon reception of a *truncate* cell, the relay sends a *destroy* cell to the next relay and answers with a *truncated* cell.

3.4 Tor Routing Protocol: if I do not get it, I drop it

Tor separates identification from routing by creating circuits of 4 nodes, including the Tor client and three relays. More generally speaking, circuits can have different

sizes depending on their purpose. Most of the time, Tor uses a circuit of 4 nodes to reach well-known network services, such as web servers. Sometimes, Tor builds circuits of 5 nodes to fetch a descriptor or to meet the requirement of an internal Tor behavior. For any circuit construction permitted by the Tor protocol, we can differentiate two types of nodes at the protocol level: edge nodes and non-edge nodes. This difference matters for one of the main functionalities of Tor: handling cells when they arrive. We want to understand how Tor deals with the difference in cell types, positions of the Tor instance on the circuit (edge or non-edge) and directions (inbound or outbound) of the cell.

Function *circuit_receive_relay_cell*() [14] shows two separate behaviors depending on the Tor instance position in the circuit. If the Tor component is not on the edge of a circuit, either it forwards the cell or it recognizes it (meaning that it is a link-level cell and that the decryption succeeded). If the Tor component recognizes a cell in a non-edge position but it does not know what to do about it, the Tor protocol drops it. The same behavior happens when an unwanted circuit-level cell is received at an edge position. Except for corner-cases which induce a tear-down, most of the unwanted cells are dropped. This feature is also explicitly authorized, with the *relay drop* cell type. This is a circuit level padding cell currently used to self-test relay bandwidth. The edge node always drops such cells without further investigation. A third and last case of ignored cells happens when an unknown subtype of *relay data* cell is received at the edge of a circuit. Tor could tear down the circuit but does not, to enforce forward compatibility.

Dropping unwanted packets is a typical behavior in network protocols. Tor does the same but does not start any further investigation to preserve its security properties. Past attacks have been known to rely on this behavior. The best example so far is the CMU-FBI attack [3] where the attacker injects a signal from an edge node and then, the relays on the path read the signal (Figure 2). In this case, the signal was a sequence of *relay data* versus *relay-early* in the inbound direction allowing the attacker to encode any wanted message in binary mode. The intended behavior was that a *relay-early* cell type was not meant to be received in the inbound direction (towards the client) but not explicitly unauthorized. At the end of the path, the Tor client drops the unwanted *relay-early* cell without further investigation, leading to the deanonymization exploit as the binary signal can be read along the circuit by colluding relays.

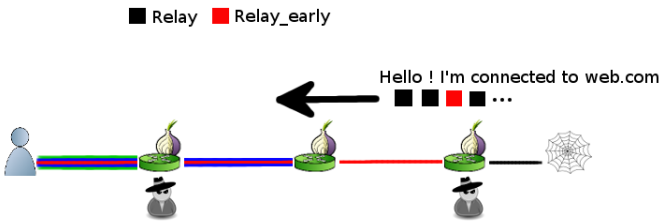


Fig. 2. Relay versus relay-early signal attack

In the following sections, we show how to exploit the dropping behavior to conduct new attacks which impact anonymity of onion services and may deanonymize Tor users.

4 Side-channel guard discovery of onion service

All the anonymity exploits introduced in this paper rely on the dropping behavior of the Tor routing protocol. Our first attack aims to discover the entry node used by any onion service in no more than a few days and without running any relay in the network. This method does not reveal the location of a hidden service directly but gives the relay on which the hidden service is always connected, which considerably narrows down the work needed towards a full deanonymization.

4.1 Attack overview

In order to create Tor circuits towards an onion service on-demand, we need its onion address and a modified Tor client [9]. Figure 3 shows how the attack proceeds. The objective is to put the HS Guard at saturation long enough to be noticeable in the reported relay bandwidth statistics. Currently, reported relay bandwidth statistics are an average of the relay consumption every 4 hours. Therefore, we have to perform at least 8 hours of constant data flow to be sure of having a 4-hours interval under our saturation flow.

In this view, we create many circuits towards the onion address with our client having the option *UseEntryGuards* set to false. By design of the hidden service protocol and the path construction, all the circuits go through the HS Guard. Then, we send *relay drop* cells or any other cell type that would be dropped in the other edge (the onion service). We used *relay drop* cells

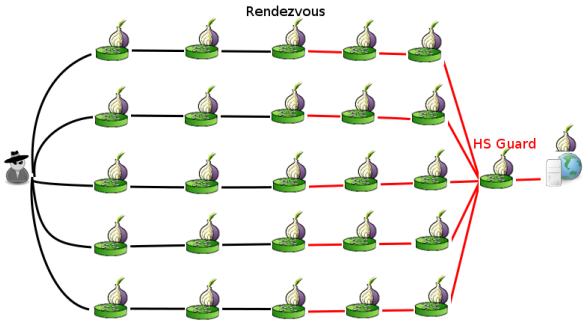


Fig. 3. Guard discovery technique: relay drop attack against an onion service. Black links are built by the adversary. Red links are built by the onion service.

because they do not raise any log message. We could use any other cell that is dropped without a tear-down of the circuit, however all the others create a [Warn] log message. Using these cells might raise the suspicion of the onion service's operator but might also be a problem for low-storage onion services: we can fill the memory with log messages at the cell rate we send.

Our cells are sent in a round robin fashion through all opened circuits to split the load over all relays. After 24 hours, all routers should have reported their extra-info descriptor with the relay bandwidth history. The one router (normally having the guard flag) that reaches a peak of saturation within the 8 hours interval would be, in theory, the HS Guard. But in practice, some other guards might saturate during the same time frame and lead to false positives for the following reasons:

- The consensus weight is computed from over evaluated bandwidth measurements, which makes a particular relay more heavily loaded due to the higher weight received.
- The relay operator lies about the advertised bandwidth and since the consensus weight takes into account the advertised bandwidth, it is possible to induce a high consensus weight [36].
- The guard is overloaded due to its length of service, since each time a new Tor user appears, he potentially selects that relay and keeps it as his guard.
- The guard is overloaded due to the high traffic generated by the particular hidden service it serves.
- The guard is overloaded due to the natural variance: sometimes, relays can saturate due to loud circuits. However, the longer the attack, the smaller the chance that the natural variance could account for false positives.

Evaluating how many false positives we get and engineering some parameters of the attack to retrieve the HS Guard with few false positives is the result of the analysis presented in Section 4.4. The likelihood of successfully retrieving the HS Guard is intuitively related to the spare resources of the entry position. Due to the following reasons, we believe that the spare resources needed are usually available (this is verified in Section 4.4):

- The Tor network runs at an average of 50% of exit capacity.
- The bandwidth capacity of the entry position, even after bandwidth-weights computation [46], is higher than the bandwidth capacity of the exit position. Therefore, even if the exit nodes were constantly used at 100%, the guard nodes would not saturate on average.
- The onion space traffic represents less than 1% [6] of the total traffic. Therefore, they cannot consume all spare resources in guard and middle positions.

Even if, on average, the entry position has spare resources, some guards may fall short. That would not allow us to detect their increased consumption of bandwidth due to the attack. Evaluating the chance to miss the HS Guard and engineering some parameters to retrieve it is also part of our work detailed in Section 4.4.

4.2 Ethical considerations

Since we wanted to test our attack on the real Tor network, we thought of a responsible way to carry on the research. To that end:

- Our guard discovery attack was applied to a dummy onion service that we created.
- We ran the entry node used by our onion service.
- We did not overload our relay too much (Figure 5), making sure that Tor users passing by this relay at the same time would not be disturbed by our attack. Characteristics of our relay are detailed in Section 4.3.2.
- We did not overload other relays by design (Figure 3).

– We did not inject any modified relay into the network.

4.3 Attack experimentation

We developed a modified Tor client [9] and extended the Tor control protocol to interface our functionalities with a python script to obtain a simple, yet powerful tool to schedule the attack on onion services of our choice. The main options are the bandwidth we want to inject, the number of circuits we want to maintain connected to the onion service and the onion address of the target.

First, we discovered a weakness in the Tor counters implementation that we could exploit to make this guard discovery method easier. A general behavior of Tor is to increment the counters before adding the cell to the output buffer. This choice makes the counters not quite accurate in the normal use of circuits since a circuit might be killed while some cells are in this buffer, "ready" to be relayed. But it is worse in our scenario. With the set-up in Figure 3, we can create an asymmetry between read/written bandwidth statistics if a condition is fulfilled. If the bandwidth of the onion service is smaller than the bandwidth of the HS Guard, then we can fill the output queue of the HS Guard with our cells until the function *circuits_handle_oom*() is called. This function would kill the circuits and remove the cells from the queue, which creates a unique asymmetry between read and written bytes in the reported bandwidth statistics. We re-create a circuit when one is killed and re-fill the HS Guard output queue to generate the asymmetry. This bug can be exploited by combining several issues: 1) Protocol issue: we can massively send cells that are not constrained by any flow control (neither at the circuit level or the stream level). 2) Counter issue: read/write counters are synchronized only when the protocol behaves as it should behave and nothing prevents their de-synchronization.

This bug was reported to and acknowledged by the Tor project.

4.3.1 Interaction between the HS guard capacity and the onion service capacity

Currently, if we apply our attack against an onion service with a capacity inferior to the spare resources of its guard, the bug described above is triggered. That is, if we inject more than the onion service network capacity, the HS guard buffers at TCP level first. Then, once the TCP send buffer is full, the HS guard buffers at the circuit level queue. This buffering fills the memory until *circuit_handle_oom*() is called to kill the circuits and recover the memory from its queues. In the meantime, the cell payload is counted in the read statistics and ignored in the write statistics. As a result, a unique asymmetry between read and write can be observed in the bandwidth history. The size of this asymmetry is directly dependent on the bandwidth injected by the malicious Tor client (the rate in which we fill the HS guard queues and induce the oom killer).

If the capacity of the onion service is superior to its guard, then we can make the guard saturating as described in Section 4.1. In our experiments, we explore in details this situation since the outcome is less trivially identifiable than the counter-exploitation. We show that the identity of the HS Guard can still be easily retrieved by saturating it. Our method is the same: we send relay drops, or any other cells that are dropped by the onion service and that are not constrained by the circuit/stream level flow control.

4.3.2 Proof of concept

Using Chutney

Ethically speaking, we cannot reproduce the bug described above in the real Tor network since the OOM killer of Tor on the targeted guard might kill circuits from legitimate clients to recover its memory. We used Chutney [44], a tool to set-up a local and private Tor network and ran an experiment over it. Figure 4 shows one of the relays having a large discrepancy between read and write values during the attack, giving a unique pattern to the relay. This relay was indeed the primary guard selected by the onion service.

On the real Tor network

We can experiment the second situation on the real Tor network, when the onion service has the highest capacity, since the only impact would be an increased consumption of available bandwidth in our relay.

Figure 5 shows the results of an injection of 720 KB/s split into 70 circuits during 8 hours. The average utilization of our targeted entry relay was about 300 KB/s with an advertised bandwidth of 1 MB/s. During the attack, the relay recorded a peak of 964 KB/s of utilization during 4 hours, which is the average legitimate use of our relay plus our attack bandwidth. This experiment is a proof of concept, and we could main-

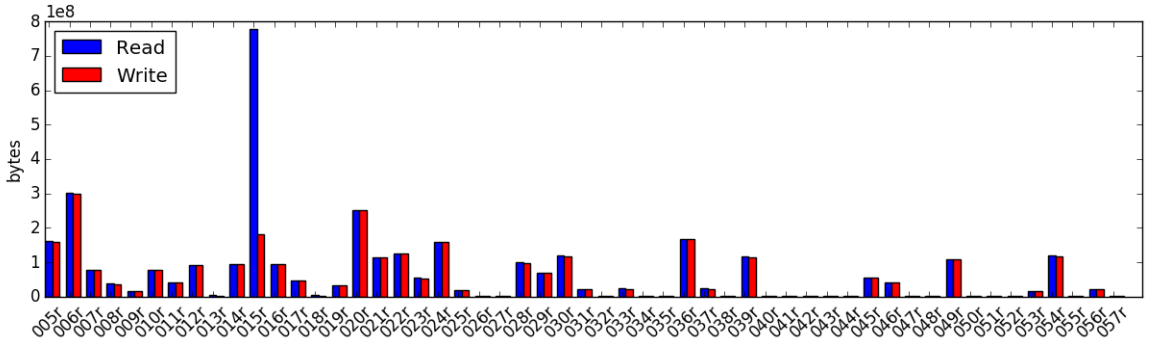


Fig. 4. Proof of concept of the guard discovery attack using *relay drop* cells and exploiting the counter weakness. The private network was composed of 5 authorities, 50 non-exit relays, 3 exit relays and 1 onion service. 015r was used as the HS entry guard during the attack by the onion service and shows a large discrepancy between public read and write counters.

tain the load a few 4-hours intervals more to facilitate the extraction of the right relay from the history in a case of real guard discovery attack. We investigate this in Section 4.4.1.

It might be possible to overload the HS guard without using our modified Tor client but using some legitimate clients and trying to generate enough traffic while acting normally. However, it would depend on the application running on the top of the onion service. That might be easy for a file sharing application such as OnionShare [11] but more difficult for other types of application, like Ricochet [10], an instant messaging software.

Our method has three advantages compared to overloading the HS Guard using the onion service's running application (1 and 2) and previous attacks (3): 1) It is application-independent, meaning that there is no interaction with the application layer and the extra load due to the attack would not be visible to an application-level statistics observer. 2) It uses the downstream bandwidth of the onion service instead of the upstream bandwidth when creating an overload by, for example, downloading files. The downstream bandwidth of network links is higher in some scenario (e.g., an onion service at home not connected with optical fiber). 3) It does not require injecting any relays into the network (compared to the known guard discovery through a Sybil attack).

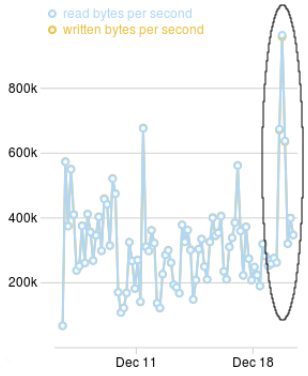


Fig. 5. Proof of concept of our relay drop guard discovery attack on the real Tor network on December 18. 720 KB/s of relay drop type cell, split into 70 circuits, have been pushed during 8 hours. This guaranteed to include a full attack window of 4 hours (1 observation) since we assume that we do not know when each interval starts as it is different for each relay. Image from Atlas [4].

4.3.3 Vanguards' proposal

This attack is not the only way to perform an efficient guard discovery of onion service. Proposal #247 called Vanguards [8] solves the known Sybil attack in which someone signs up a bunch of middle nodes and makes client connections towards the onion service, until one of his middle nodes is used in the onion service circuit, next to the HS Guard. This proposal suggests using a restrictive set of relays in the second and third positions of HS circuits. Doing so, it mitigates our attack in some scenarios. If the cumulative bandwidth of the second set of relays (or the third set) is smaller than the bandwidth of the HS guard, then the HS guard might not saturate, or they might all saturate. When the HS guard does not saturate, the attack is somewhat mitigated since we now rely only on observing value peaks, which would also be visible on the second (or third) set of guards. But, this is unlikely since those nodes are also guard-flagged and spread the load of the attack between themselves. Moreover, the proposal suggests skewing(even more) the selection distribution of such nodes towards top relays, which would probably reduce the chance to mitigate

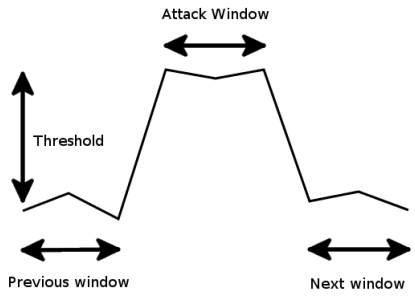


Fig. 6. Abstract view of OnionShape's measurement windows

the attack. Finally, if we do not manage to extract only one relay (the HS guard), it will succeed due to the fast rotation of those relays. Each time a rotation is performed, we can try again.

4.4 Going further with the descriptors history

So far, we demonstrated that it is possible to overload the guard of an onion service, whatever application is running on top of it. In a real attack scenario, we need to recognize the correct guard among all the relays running on the network. We developed OnionShape [9], a program able to crawl the history of relay descriptors [7] and to look for such attack pattern. OnionShape proceeds by sliding, side by side, three windows containing measurements, for each relay in the network (Figure 6). These windows slide during the period we consider and apply a test to flag any relay that matches a distribution matching the one described in Figure 6. Our objective is to find the suitable value of parameters for which we can retrieve our relay. The chosen parameters are the threshold overhead (compared to the previous history of bandwidth), the duration of the attack (in a number of reported measurements) and the statistical tools used to compare the middle window to the others.

4.4.1 Methodology

We ran OnionShape over an entire month of descriptors with different values for the parameters. Each relay flagged positive by OnionShape within the attacking window time (e.g., when an attack is supposed in progress) is a false positive. Considering the characteristics of the Tor network, we want to identify what would be the impact of our parameters on the success rate, such as the attack duration. Doing so, we would increase our chance of success if we were about to launch a relay drop attack. We would like to prevent a false negative and minimize the false positives. False positives are easier to deal with, as we can extract the right relay from them by replaying the attack and getting the intersection within the false positive sets obtained, until only one relay remains (details in Section 4.4.4).

In this analysis, we want to answer the following questions:

- How can we be sure to extract the right guard?
- What it the impact of having an onion service's spare resources smaller than its guard spare resources? (considering that the bug described in Section 4.3 is solved).
- Would it help to modify the current interval of 4 hours between two reported bandwidth measurements?
- How exactly does our confidence increase if we use multiple observations?

First, we start with the simplest statistical tool we could use: the mean. We compare the average value in the attacking window to the average value in the previous and next windows. If we obtain too many false positives, we can move to higher distribution moments.

Assumption 4.1. The average bandwidth relayed by the HS Guard is always higher during a relay drop attack than the previous and next considered windows.

Justification. The bandwidth induced by the relay drop attack fills the spare resources of the HS Guard. This assumption is always valid unless the guard operator reduces its relay's token bucket during the attack windows, starts hibernating, goes offline, etc.

Assumption 4.2. The variance computed over the measurements in the attacking window is always smaller than the variance in the previous and next considered windows.

Justification. When filling the spare resources of the guard, the relay drop attack should inject enough bandwidth to make the guard reaching its token bucket limit. The guard should then provide constant measurement values (i.e., values at saturation) in the history. Again, this assumption becomes false if the HS Guard operator

modifies its token bucket limit during the attack, starts hibernating, goes offline, etc.

Our analysis is divided into three parts: 1) we evaluate how the parameters of our classification influence false positives and false negatives considering that the onion service's capacity is higher than the spare resources of its guard. 2) We consider that the onion service's capacity might not be higher than the one from its guard and we evaluate its impact on our attack, considering that the counter weakness is solved (i.e., we assess our attack for different onion service's bandwidths). 3) We discuss higher intervals and how several observations help to extract the right relay.

4.4.2 Analysis - Onion Service's capacity higher than its guard's spare resources

Figure 7 shows the average number of false positives using OnionShape on December 2016 history. We flagged a relay positive in Figure 7a's plain lines when the following conditions were true:

$$\begin{aligned} mean(prev\_win) \times thresh \leq mean(attack\_win) \\ mean(next\_win) \times thresh \leq mean(attack\_win) \end{aligned} \quad (1)$$

And when the variance is used, we add the following conditions without any thresholds:

$$\begin{aligned} variance(prev\_win) > variance(attack\_win) \\ variance(next\_win) > variance(attack\_win) \end{aligned} \quad (2)$$

Considering a threshold equal to 1.0 and with our assumption 4.1, we cannot get any false negative (i.e., missing the targeted guard). The false negative could happen if we increase the threshold because it depends on the targeted guard's spare resources. Intuitively, a shape induced (for a given threshold) in the bandwidth graph like in Figure 6 is less likely to be observable as the threshold increases, since the spare bandwidth of the guard is not infinite. However, in practice, guards have spare resources as shown in Figure 8a displaying the bandwidth utilization of guard relays and unflagged relays during December 2016. We can observe that 80% of guard relays are used below 50% of their capacity, and nearly 100% of guard relays are used below 80% of their capacity. Regarding unflagged relays, their utilization is even smaller: almost 100% of them are used below 60% of their capacity. Therefore, to reduce the set of probable guards, we might increase the threshold gently, but relative to the average observed load on relays.

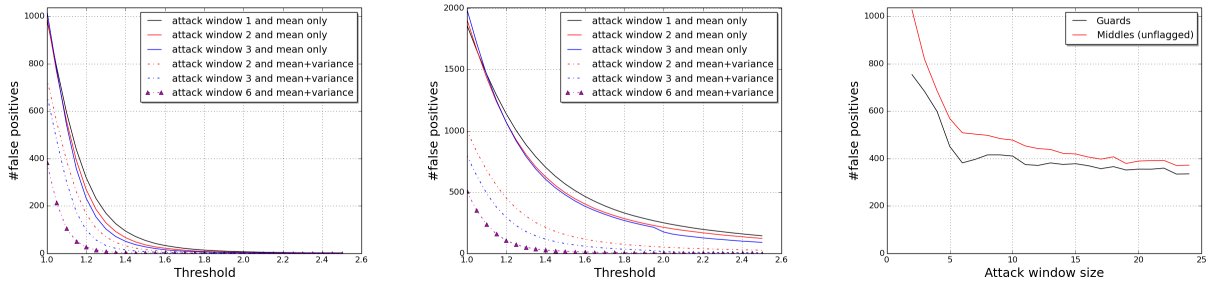
From OnionShape's results in Figure 7a, we see with the reduction of false positives that, as we increase the interval (the attack window), the differentiation increases: we have fewer candidates by expanding the attacking window. This figure also confirms our expectation over threshold variation: the false positives are reduced when we increase the threshold. However, increasing the threshold also increases the probability of a false negative (Figure 8). This result is explained by looking at Equation 1 and the actual load on relays in Figure 8a: for some relays, as the threshold increases, Equation 1 becomes false due to the lack of spare resources. Finally, as we continue to increase the threshold, the equation becomes false for all relays.

Plain lines in Figure 7a are testing assumption 4.1 only and show to be risky: the HS Guard must be used under 50% of its available capacity to be detected and distinguished from the others (prob of detection considering the probability distribution of guard relays: $\approx$ 66%). Dashed lines add the variance test from assumption 4.2 to flag a relay positive. For an attacking window size of 3 reported measurements (i.e., 12 hours), a threshold of 1.6 is enough to extract the guard if it was running below 62.5% of its advertised capacity before and after the attack (prob of detection: $\approx$ 83%). Most guards are running below 50% of their advertised capacity, hence a false negative is possible but not likely ($\approx$ 17%). The threshold can be reduced to 1.4 if the attack duration moves to 6 reported measurements (24 hours) (prob of detection: $\approx$ 94%).

Figure 7b shows the results over the unflagged relays instead of the guard-flagged relays. Due to the higher number of relays, the number of false positives is higher for the same threshold. These results give the wrong intuition that an onion service would be less vulnerable to our guard discovery attack if it chooses its entry node among the unflagged relays (i.e., using EntryNodes option). However, a false negative is less likely to happen with unflagged relay (Figure 8b); hence the adversary can increase the threshold comparatively to guard-flagged relays. Moreover, we also noticed a security flaw [12] and using the EntryNodes option is now discouraged.

Figure 7c shows how the number of false positives evolves with the size of the attack window, for a threshold of 1.0. Starting from 6 reported measurements, the attacker does not gain much additional utility when increasing the duration of its attack.

The variation through time of the threshold is also studied. In Figure 9, we computed the threshold to obtain at most 5 false positives at every moment of De-

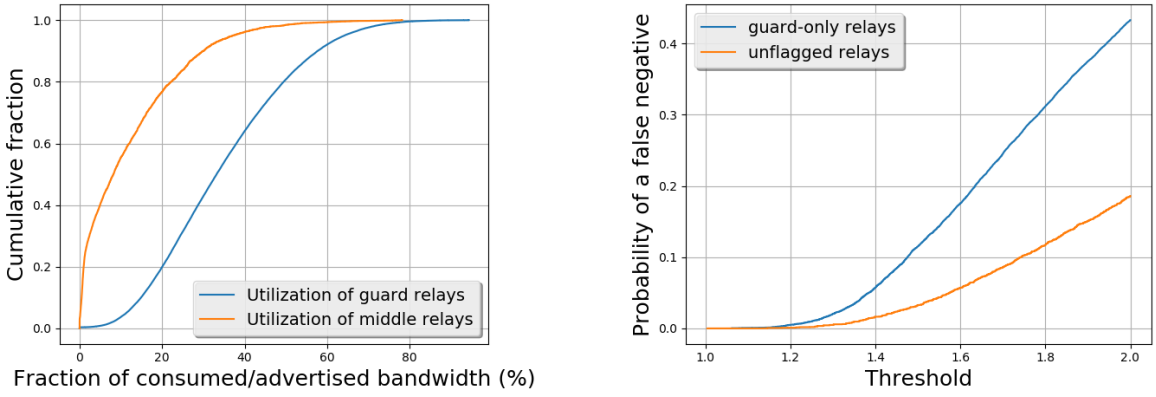


(a) Guards only - Searching for an acceptable threshold

(b) Unflagged only - Searching for an acceptable threshold

(c) Guards and Unflagged - Searching for an acceptable attack duration

Fig. 7. OnionShape's result over December 2016 network history. We measure the average false positives for varying parameters to answer research questions of 4.4.1. Figure 9 complete this result by showing the variance of these measurements.



(a) Cumulative fraction of relays utilization, where the utilization is defined by the fraction of consumed bandwidth over the advertised bandwidth - Data from December 2016

(b) Risk of a false negative w.r.t. the chosen threshold. This graph considers that the assumptions 4.1 and 4.2 hold and that the **onion service capacity is higher than its guard spare resources**.

Fig. 8. Evaluates the load on relays and computes the probability of a false negative from this evaluation (8b).

cember 2016. If the standard deviation of such obtained distribution is high, then it would mean that choosing a standardized value of the mean as an appropriate statistical tool in a real attack scenario could be ineffective. Given the results of Figure 9, increasing the attacking window size decreases the standard deviation and should reduce the likelihood of an unexpected outcome for a given choice of threshold.

4.4.3 Analysis - With different onion service's bandwidths

In the previous section, we analyzed the efficiency of our guard discovery attack considering that the onion service's bandwidth was higher than the spare resources of its guard. Indeed, if the onion service bandwidth is smaller, the adversary knows that she could win with strong confidence due to the counter design flaw explained in Section 4.3. But, what if this counter issue was not exploitable anymore? In this section, we consider this issue solved by the Tor developers provided all guards are up to date with the fix.

We computed the probability of a false negative in Figure 8 for a given threshold by:

$$\frac{\sum ConsWeight_i}{TotConsWeight} \; \forall i : \frac{Consumed\ Bw_i}{AdvBw_i} > \frac{100}{threshold} \tag{3}$$

Which gives the probability of a false negative for a given threshold if the onion service bandwidth is higher than any guard's spare resource.

To evaluate the same probability considering the effect of the onion service capacity, we evaluate how the load on guards induced by different onion service capac-

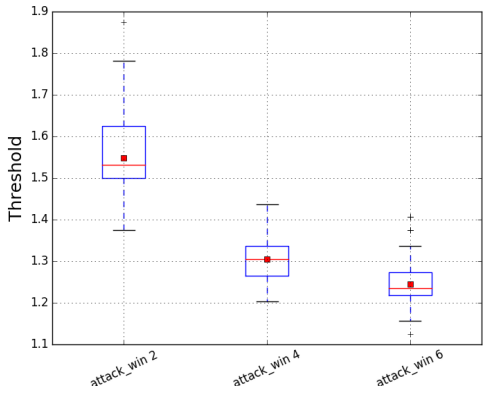


Fig. 9. Variation of the threshold through time to obtain a maximum of 5 false positives, given an attack window of size 2, 4 or 6 measurements.

ities could be detected by our threshold-based classifier, considering that the relay drop attack is trying to fill all onion service's available bandwidth ($OSBw$)

$$AdvBw_i = max(AdvBw_i, Consumed\ Bw_i + OSBw) \tag{4}$$

Then we re-compute equation 3 for different $OSBw$. Figure 10 shows our results and gives insights about the probability to miss the extraction of the entry relay depending on the bandwidth of the onion service. The 100MB/s line matches perfectly the probability from Figure 8. It captures the fact that having such downstream bandwidth would be higher than any current guard's spare resource. The 10MB/s is more interesting, as it is a usual downstream bandwidth when renting a small VPS or having the onion service operated at home behind a usual 100Mb/s downstream bandwidth. The probability of a false negative is close to a 100MB/s set-up since 10MB/s fills a large fraction of almost any guard's spare resource, which will be detected by our classifier. Finally, the more the onion service decreases its spare capacity, the safer it is from our guard discovery.

4.4.4 Analysis - Interval size and effect of multiple observations

We simulated a larger interval by merging 4-hours reports when analysing the load and computing the probability of a false negative from it. Figure 11 shows that increasing the interval gives a smaller probability for a false negative. This result can be intuitively explained by understanding why we could obtain a false negative: when the guard is already too much overloaded by legitimate use during our attack. This event is less likely to happen as we increase the time of the reported measurement (and the duration of the attack).

Changing the interval size impacts the false positive computation exactly as the attack window size impacts the false positive in Figure 6 and Figure 9 (i.e., it reduces false positives as the interval increases). Therefore, increasing the interval does not help more than increasing the time needed to extract the right relay. That is, increasing the interval costs more time to the adversary but makes her more confident at each observation.

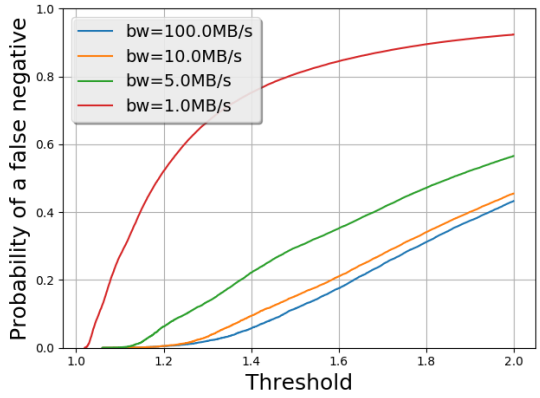
The second question we try to answer in this section is to investigate how our confidence about the extraction of the right relay increases with multiple observations. The idea is to compute the intersection between the set of relays obtained at each observation. If the threshold used is low enough, we should obtain only one relay after several observations with the probability of success being $P_s = (1-p)^n$ with p the probability of a false negative and n the number of observations. Figure 12 gives some insights for an attacker performing an attack per day during 12 hours. We can observe how typically the size of the set of candidates is reduced at each observation. This Figure is plotted from the output of OnionShape for a few days of observation in December 2016. Choosing other days in the history would give different values for #false positives but would show the same steep decrease as the number of observations increases.

Finally, the attacker would choose her threshold with respect to the probability of a false negative he is willing to accept. If the attacker is able to obtain a precise measurement of the onion service's bandwidth, she can speed up the process to recover the HS Guard by potentially selecting a higher threshold if the bandwidth measured is large enough. If she is not, choosing a low threshold such as 1.1 would still allow her to retrieve the right entry guard in a few days with a probability of success $\approx 96\%$ against a few MB/s onion service.

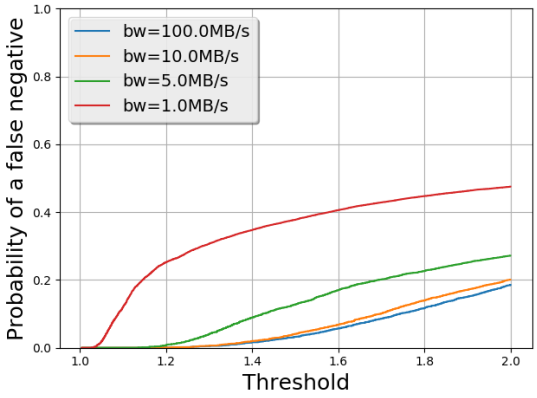
4.5 Countermeasures

Fixing the counter weakness

A fix involving discounting bytes when *circuit_handle_oom*() is called (one of the possible options) is currently (mid-2017) under discussion at the Tor project. While it fixes the trivial guard discovery attack presented in Figure 4, it does not solve all the problems.



(a) Assuming that the onion service entry relay is a guard-flagged relay (default behavior)



(b) Against unflagged relays: If the onion service uses the option EntryNodes and configures a relay without a guard flag for its entry position.

Fig. 10. Gives the probability of a false negative that our OnionShape's classification outputs depending on the bandwidth of the onion service.

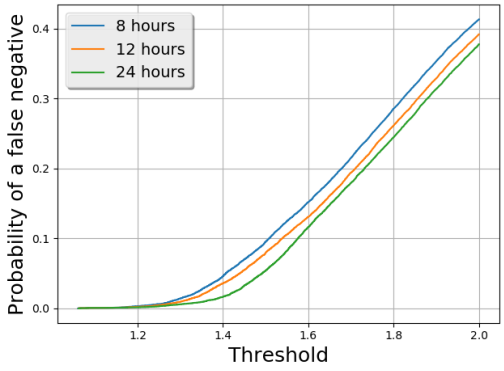


Fig. 11. Gives the probability of a false negative that our OnionShape's classification outputs depending on the length of the reported interval. The probability for the current 4 hours interval was given in Figure 8b

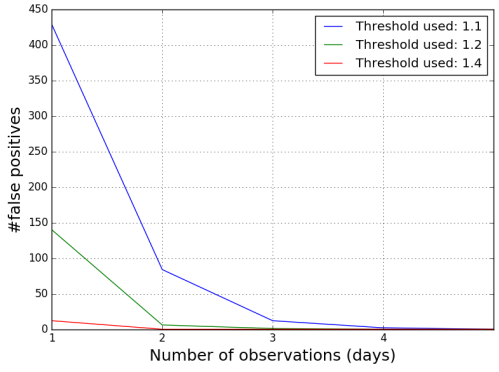


Fig. 12. Confidence of the adversary on multiple observations

Volume analysis

Given that the attack needs a high throughput to succeed, the onion service might count dropped cells and rate limit them by killing circuits once the limit is exceeded. This volume analysis does not preclude the adversary to legitimately use the application on the top of it and generate the same overload. Moreover, this idea has to be carefully put in perspective with the forward compatibility currently designed in the protocol. More details in Section 6.

Increasing the interval size

Currently being 4 hours, increasing it to 8, 12 or 18 hours does not help more than slightly increasing the cost of the attack. Expanding it to more than 18 hours would generate more problems to tackle: 1) extra info descriptors in which measurements should be included are uploaded every 18 hours; hence some descriptors would miss measurement values and Tor-related applications using descriptors would be impacted. 2)If Tor crashes/restarts, it loses the current bandwidth accounting value meaning we would potentially lose more counted bytes if the interval increases, impacting the reliability of these reports.

Removing public bandwidth report

We may also remove the side-channel by not allowing guard relays to publish their measurements and replacing them by a network-wide aggregation using PrivCount [34]. However, eliminating public bandwidth report would cause many issues to Tor-related applications that use them.

User-side countermeasure

One appealing countermeasure for an onion service operator who deeply care about the "hidden" property of his service would be to configure a bandwidth rate limitation (i.e., using BandwidthRate option) as close as possible to the average bandwidth consumption of his service. The onion service operator could use an application-level statistics observer over a time frame to estimate the right BandwithRate value. This would impact the performance of the service, though.

Discussion

Among the possible countermeasures, we believe that removing the side-channel and educating end-users to properly configure their hidden service are the most appealing ones. Some recommendations about configuring an Onion Service would be to: 1) Configure the bandwidth as close as possible to an average estimation of the legitimate use. 2) Avoid using the option EntryNodes to set up an unflagged relay as an entry node. Alongside the basic reasons for not using unflagged relays as an entry node (stability, bandwidth, etc.), our analysis showed that the relay discovery would still be easy. 3) Avoid running an onion service as well as a Tor relay on the same instance. This would induce a trivial deanonymization of the onion service by an adversary sending cells that are dropped by the onion service because a large discrepancy between read and write bytes could be observed in reported measurements of the relay.

4.6 Conclusion

As soon as the counter weakness is fixed, this 1-day guard discovery attack moves to a few-days guard discovery attack with a high probability of success. Some limitations exist in our analysis. First, the precision of our attack success depends itself on the precision of the self-advertised bandwidth measurements of each relay. Secondly, we assume a graceful behavior of relay and relay operator in overloaded conditions. Finally, the classification method that we used is elementary but proves to be sufficient to demonstrate the effectiveness of our attack. An attacker could run more advanced machine learning training methods on the real Tor network, which is likely to lead to even more efficient classification techniques. We did not explore this avenue due to the ethical concerns that such kind of training raises. We also think that a training method would be complex on a private Tor network given the fact that it must look like the real one or we would obtain a classifier that works well against a private network but might not work against the real one. OnionShape achieves good results with a safer and simpler analysis.

Our congestion attack remains usable as a building block for attacks involving other observation channels. Solving this problem would involve re-thinking flexibility and forward compatibility in-depth. More insights about this problem are discussed in Section 5 and Section 6.

5 Using forward compatibility to create the dropmark attack

The anonymity exploit described in Section 4 combines the malleability of the protocol and a side-channel to discover the entry node of an onion service. We may also try to build end-to-end correlation attacks based on the flexibility of the Tor protocol. We consider a model where the adversary controls relays and wants to efficiently spot if traffic going through one of his guards is coming out on one of his exits. A few papers tried this, resulting in the tagging attack [22, 26] for instance. This attack modifies the data flow at the entry node in the outbound direction to generate an integrity error which is detectable along the circuit by the other compromised relay during decryption but also induces a tear down of the circuit at the edge on non-colluding exits. Closer to what we have designed in this section, Biryukov *et al.* performed circuit construction fingerprinting [18] in order to deanonymize onion services. This attack, like ours, uses the fact that the Tor protocol is gentle with most unrecognized cells. The default behavior is to ignore them with emphasis on an architecture design promoting forward compatibility: every cell with a *Cmd* (Figure 1) or an unrecognised *Rel cmd* is silently dropped. In this section, we use the dropping behavior of the Tor protocol to create a traffic confirmation tailored to the Tor network.

5.1 Silently dropmarking Tor flows

Given the Tor protocol, we can send from an edge node any number of cells that would be silently dropped by the other edge. In order to avoid adding latency to our victim flow and easily extract the mark that we add, which we call a *dropmark*, we target timing intervals where flows are used to be idle. Considering what happens when a service is requested through Tor, such as connecting to a particular website: a *begin* cell is sent from the Tor client to the circuit, which triggers a DNS resolution at the edge. When the DNS resolve succeeds and the connection is set up with the desired service, the edge relay sends a *connected* cell in the inbound direction (towards the client). The next cells that would be sent towards the client would be the response to the GET request that the client has issued.

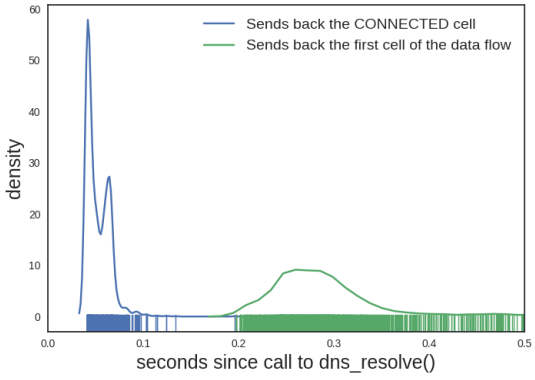


Fig. 13. Density plot of the connected cell and the first data cell induced by 2361 web requests handled by an exit relay in Shadow. Truncated to 0.5 sec.

Figure 13 shows a density plot of the connected cell and the first data cell sent towards the client. This plot results from 2361 web connections that an exit relay performed in the Shadow network simulator [33]. These connections reach many different web server locations simulated to induce a lookalike response compared to web servers on the Internet. The idle time, during which no cell goes towards the client, corresponds to the round-trip-time (RTT) of the Tor circuit, plus the RTT of the edge connection. This is true only if the request is not sent optimistically before the client receives the *connected cell*. If the request is sent optimistically (this feature is even documented in Tor's code as a way to speed-up the HTTP protocol), then the idle time corresponds to the RTT of the edge connection. This window is still large enough to encode our dropmark with relay drop cells or any other cell that would be silently dropped by the other edge.

5.1.1 The dropmark attack

We present the dropmark attack, a way to carry along the circuit one bit of information without adding latency to the victim flow. This attack needs an edge-and-relay colluding model and two characteristics within the low-latency anonymizer:

1. The protocol should silently drop unrecognized packets
2. The circuit must be idle at some moment (no cover traffic)

We conjecture that the dropmark attack impacts any low-latency anonymity network that has these two characteristics and we demonstrate the effectiveness of this attack on the Tor network.

The encoding part of the dropmark attack happens at an edge node of a circuit. This edge node could be an exit relay, an onion service (presumed honeypot), an HSDir, an intro point or a rendezvous point. For simplicity, we consider only the exit relay in the following part the paper and in our experiments. As shown in Figure 13, we take advantage of the idle window that exists on any Tor circuits each time a *relay begin* cell is sent towards the exit relay that carries the desired IP address or domain name to connect to. Upon the reception of the *relay begin* cell, as soon as the dns_resolve() function is called, we log the IP address and we send 3 relay drop cells towards the client (again, any other type of relay cell that would be silently dropped is appropriate).

Figure 14 gives the intuition of the decoding function that would decide whether a dropmark is embedded in the flow or not. The decoding function (on the guard relay) considers the first few cells of the circuit in the inbound direction and verifies this observation: if 3 cells have the same timing arrival (more or less a few milliseconds) within the first four cells, we flag the flow as having a dropmark.

In order to confirm our intuition, we implemented our dropmark attack in Tor and ran simulations in Shadow.

(a) When no dropmark is sent by the edge relay (classical circuit behavior)

(b) When a dropmark is sent along the circuit by the edge relay

Fig. 14. Simplified view of cell timings flowing towards the client from the perspective of a guard. We construct a distinguisher (the decoding function) from a timing analysis.

5.1.2 Experimentations

We simulated some web page downloads to 20 servers from the Alexa dataset [13] in Shadow. Each web client downloading a webpage should use a fresh new circuit. Doing so, it captures the Tor browser behavior that uses a new circuit for each web address. While we simulate web traffic and test the dropmark attack against it, we conjecture that any type of application traffic flowing through Tor is subject to the dropmark attack. Figure 13 shows that the dropmark attack takes advantage of the protocol used in the transport layer (TCP), hence common to all services using TCP and independent of the application layer. We observe our relays at the circuit level with custom logging events.

We also tested different situations where 1) No exit sends the dropmark but all guards apply the decoding function. 2) All exits send the dropmark and all guards apply the decoding function. The first situation aims at assessing the number of false positives that the decoding function might raise. The second situation aims at evaluating the number of false negatives. We test these two situations under a light loaded network and under a network loaded similarly to the real Tor network to evaluate how the congestion could impact the efficiency. To simulate a light loaded Tor network, we use 200 web clients, 50 relays and 20 web servers based on CollecTor archives [7] from March 2017. Altogether, they push ≈25% of the exit total capacity. To simulate a loaded Tor network, we use 450 web clients, 50 relays and 20 web servers based on the same archives. Altogether, they push ≈ 50% of the exit total capacity.

In the first situation with the light loaded network, where no dropmarks are sent, 12972 circuits resulting from the web clients' Tor daemon activity have been tested among the guards. In total, 1 false positive was detected and the fraction of HTTP transfer errors over successful transfers was 37/13030 ≈ 0.3%. Looking in the log details, we saw that the *connected* cells were received at the same timing as the data cell, raising a lookalike dropmark fingerprint. This situation might happen if the middle relay is under congestion. Notice that the number of transfers is higher than the number of observed circuits among the guards. Moreover, we observed that some *begin* cells with the same destination are sent along the same circuit when the timing between them is too close. We tried to prevent that by setting MaxCircuitDirtiness to 1 second but somehow, some streams are attached to the same circuit despite the dirtiness. We investigated the problem and opened a ticket on the bug tracker [35]. As far as our simulations are concerned, this is not an issue: those streams can just be ignored.

In the second experiment, we increased the overall congestion in the network by using 450 web clients instead of 200 and consuming ≈ 50% of the exit capacity to match the current Tor network load [5]. In this simulation, 26245 circuits were tested among the guards. In total, 9 false positives have been detected but the fraction of transfer errors over successful transfers raised to 1448/25869 ≈ 4.3%. This result seems to indicate that the increased network congestion does not impact the dropmark attack. Indeed, the congestion happens mainly in the exit position due to the scarcity of exit bandwidth while the dropmark attack should be impacted by congestion in the middle position. If we were lacking middle bandwidth, the impact could have been noticed.

In the second situation with the light loaded network, where every exit relays send dropmarks for all connections, no false negatives were detected and 13173 circuits seen among the guards were tagged as having a dropmark. Increasing the size of the network to 450 web clients, we obtained 41 false negatives over 29322 tested circuits with a fraction of HTTP transfer errors over successful transfers of 1226/26609 ≈ 4.6%. Most of those false negatives were due to circuit failures.

Finally, this method shows good results in a loaded network with ≈ 99.86% true positive rate and ≈ 0.03% false positive rate and is not perturbed by timeouts that could happen in a network (such as a web server not answering a connection request). Moreover, this method is not based on the application layer and does not perturb it, leading to successful correlation even if a few bytes are exchanged between the source and the destination. Nothing prevents an adversary from pairing this attack with other methods such as Rainbow [32] or passive timing analysis for even more reliability. Apart from the FBI-CMU unpublished *relay early* confirmation attack,

this attack is the first correlation method known to be efficient and that presents such an advantage.

5.2 Countermeasures

The dropmark attack can be performed with other RELAY cell types, such as a RELAY_DATA with a wrong stream id (dropped with an info level log message), RELAY_RESOLVE or any unused relay command number (dropped with a warn level log message). This list is not exhaustive and the available code allows testing some of them. Currently, the range of visibility in the log messages depends on the cell used for the dropmark. The visibility is from none (RELAY_DROP) to the notice level and info level. A first idea would be to log a warning message for all of these events since the adversary would rather have one left at a higher level (less visible). It is however common to receive a RELAY_DATA cell with a wrong stream id even though it is not suitable for a warn level log message as these will overload the logs with no possibility to differentiate a dropmark attack from a legitimate message at the notice log level. For example, using Tor browser with log info enabled during 2 minutes and visiting the front page of 3 websites (facebook.com, google.com and 9gag.com) gave us hundreds of such messages[2].

We conjecture that there exists a whole family of possible active attacks based on the flexibility of the Tor protocol. Designing countermeasures based on timing analysis, such as raising a warning or killing the circuit is pointless because the attacker might find another timing window to send his dropmark or another way of taking advantage of the protocol flexibility.

As it is, we do not have any countermeasure that would not break forward compatibility (killing the circuit where we received something unexpected breaks forward compatibility) or one of the original Tor's goal. We open the discussion in Section 6 for further research direction regarding the interaction between flexibility and security of anonymous network protocols that would help to fix this problem.

6 Discussion of possible further works

In the Tor protocol, forward compatibility is the act of ignoring an unknown cell or ignoring a known cell that is not supposed to be received (e.g., an unknown *relay* Cmd value or a *relay-early* in the inbound direction). Section 4 and Section 5 showed how forward compatibility could be exploited to retrieve an onion service's guard (combined with a side-channel) and to correlate with high probability without taking advantage of the user's data flow. Forward compatibility is a desirable feature in network protocols. Even more when the network is distributed since many different versions can compose the overall network.

Tor is not supposed to protect against end-to-end traffic correlation [22] but it is designed and improved in order to maximize the attacker's work under the classical resource model. We believe that an end-to-end correlation like our dropmark attack or the relay-early confirmation attack should be prevented as long as the countermeasure does not bring any dissuading effect on another goal of Tor. The question is then: can we come up with a workaround fix to the bandwidth congestion attack described in Section 4.1 and the end-to-end correlation attack from Section 5, in a way that will not impact the ease of deployment, usability, flexibility, and resistance to censorship of the Tor protocol?

Further work should offer more insights as to how Tor's protocol flexibility can be used to design attacks. Indeed, forward compatibility is one of the many flexible implementation choices in Tor. More insights might be gained by recasting prior attacks from the literature in light of this flexibility issue. If such further work shows that flexibility has a fundamental impact on the security of anonymous network protocols, it should come with a way to set a new trade-off flexibility/security that asserts what kind of attack the system is safe from. Moreover, it might be useful to design some adaptive trade-off that could be changed by the network itself (directory authorities in Tor's case). We conjecture that this problem goes beyond Tor and might be interesting to take into account for further anonymous network designs.

7 Code and data reproducibility

We made available on Github all the code, data, and a step-by-step tutorial that can be used to reproduce

2 [info] connection_edge_process_relay_cell(): data cell dropped, unknown stream (streamid 19412).

our graphs and numerical values from Section 4 and Section 5 [9].

8 Conclusion

In this paper, we show how a common network protocol feature, the dropping of unexpected packets, can be used against anonymous communication systems. In the Tor protocol, the dropping of unexpected packets is specified in order to enable flexibility and forward compatibility. We exploit this feature to facilitate the task of retrieving the identity of an onion service. We also show, with the dropmark attack, that we can be practically very close to the theoretical perfect and instantaneous traffic confirmation attack model usually considered in research papers. Even if anonymity can be broken in some situations, this paper does not claim that the Tor design is broken. However, it sheds more light over the fact that ensuring server-side anonymity is complex and that traffic confirmation can indeed be considered perfect and instantaneous in practice. Finally, we open the discussion regarding a trade-off between flexibility and security that could be a better approach to defeat our bandwidth congestion attack, the dropmark attack, previously known attacks, and defending Tor against yet unknown threats that could exploit Tor's flexibility. The lessons would be valuable for any high throughput low-latency anonymous communication system proposal.

Acknowledgement

We would like to thanks the anonymous reviewers and our shepherd Rishab Nithyanand for valuable feedbacks. We are also thankful to the Tor project members: Roger Dingledine for helpful thoughts, and George Kadianakis for awarding us a bug bounty during the disclosure process. This bounty is now used to run extra Tor relays. This work was partially supported by the Innoviris/C-Cure project and by the European Commission and the Walloon Region through the FEDER project USERMedia (convention number 501907-379156).

References

[1] Tor's specifications. https://gitweb.torproject.org/torspec.git/tree/. Accessed: 2017-05-20.

[2] Generic, decentralized, unstoppable anonymity: The phantom protocol. http://www.magnusbrading.com/phantom/phantom-design-paper.pdf, 2011. Accessed: 2017-05-20.

[3] Relay early confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack, 2014. Accessed: 2017-05-20.

[4] Atlas: web application to inspect details of currently running relays. https://atlas.torproject.org/, 2016. Accessed: 2016-12-20.

[5] Capacity of the Tor network. https://metrics.torproject.org/bandwidth-flags.html, 2017. Accessed: 2017-05-20.

[6] Consumption of hidden services in mbit/s. https://metrics.torproject.org/hidserv-rend-relayed-cells.html, 2017. Accessed: 2017-05-20.

[7] Data-collecting service in the Tor network. https://collector.torproject.org/, 2017. Accessed: 2017-05-20.

[8] Defending against guard discovery attacks using vanguards. https://gitweb.torproject.org/torspec.git/tree/proposals/247-hs-guard-discovery.txt, 2017. Accessed: 2017-05-20.

[9] Github repository regarding data and code of this paper. https://github.com/frochet/dropping_on_the_edge, 2017.

[10] Ricochet: Anonymous instant messaging for real privacy. https://ricochet.im/, 2017. Accessed: 2017-05-20.

[11] Securly and anonymously sharing files with onionshare. https://github.com/micahflee/onionshare, 2017. Accessed: 2017-05-20.

[12] Ticket 21155: Client's choice of rend point can leak info about guard(s) of misconfigured hidden services with entrynodes option. https://trac.torproject.org/projects/tor/ticket/21155, 2017. Accessed: 2017-05-20.

[13] Top-1000 alexa data set. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, 2017. Accessed: 2017-05-20.

[14] Tor source code, function circuit_receive_relay_cell(), line 176. https://gitweb.torproject.org/tor.git/tree/src/or/relay.c?h=release-0.2.9, 2017. Accessed: 2017-05-20.

[15] M. AlSabah and I. Goldberg. Performance and security improvements for tor: A survey. *ACM Computing Surveys (CSUR)*, 49(2):32, 2016.

[16] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

[17] A. Barton and M. Wright. Denasa: Destination-naive as-awareness in anonymous communications. In *Proceedings of the 16th Privacy Enhancing Technologies Symposium (PETS 2016)*, July 2016.

[18] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. Trawling for Tor Hidden services: Detection, measurement, deanonymization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013.

[19] P. Boucher, A. Shostack, and I. Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[20] Z. Brown. Cebolla: Pragmatic IP Anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*, June 2002.

[21] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Proceedings of the European Symposium Research Computer Security - ESORICS'10*.

Springer, September 2010.

[22] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[23] M. Edman and P. F. Syverson. AS-awareness in Tor path selection. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, pages 380–389. ACM, November 2009.

[24] N. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on Tor using long paths. In *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[25] N. Feamster and R. Dingledine. Location diversity in anonymity networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*, October 2004.

[26] X. Fu, Z. Ling, J. Luo, W. Yu, W. Jia, and W. Zhao. One cell is enough to break tor's anonymity. In *Proceedings of Black Hat Technical Security Conference*, pages 578–589. Citeseer, 2009.

[27] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, pages 1–25, 2012.

[28] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[29] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How much anonymity does network latency leak? In *Proceedings of CCS 2007*, October 2007.

[30] A. Houmansadr and N. Borisov. Swirl: A scalable watermark to detect correlated network flows. In *Proceedings of the Network and Distributed Security Symposium - NDSS'11*. Internet Society, February 2011.

[31] A. Houmansadr and N. Borisov. The need for flow fingerprints to link correlated network flows. In *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS 2013)*, July 2013.

[32] A. Houmansadr, N. Kiyavash, and N. Borisov. Rainbow: A robust and invisible non-blind watermark for network flows. In *Proceedings of the Network and Distributed Security Symposium - NDSS'09*. Internet Society, February 2009.

[33] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the Network and Distributed System Security Symposium - NDSS'12*. Internet Society, February 2012.

[34] R. Jansen and A. Johnson. Safely measuring tor. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS '16)*, October 2016.

[35] Jaym. Circuit dirtiness is inconsistant with maxcircuitdirtiness. https://trac.torproject.org/projects/tor/ticket/23374#ticket.

[36] A. Johnson, R. Jansen, N. Hopper, A. Segal, and P. Syverson. Peerflow: Secure load balancing in Tor. *Proceedings on Privacy Enhancing Technologies*, 2017(2), 2017.

[37] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, Washington, D.C., 2015. USENIX Association.

[38] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[39] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright. Timing attacks in low-latency mix-based systems. In A. Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004.

[40] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

[41] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In N. Borisov and P. Golle, editors, *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*. Springer, June 2007.

[42] A. Nambiar and M. Wright. Salsa: A structured approach to large-scale anonymity. In *Proceedings of CCS 2006*, November 2006.

[43] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos. Compromising anonymity using packet spinning. In *Proceedings of the 11th Information Security Conference (ISC 2008)*, September 2008.

[44] T. Project. The chutney tool for testing and automating tor network setup. https://gitweb.torproject.org/chutney.git.

[45] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[46] F. Rochet and O. Pereira. Waterfilling: Balancing the Tor network with maximum diversity. *Proceedings on Privacy Enhancing Technologies*, 2017(2), 2017.

[47] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of ESORICS 2003*, October 2003.

[48] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3-4):355–377, 2004.

[49] V. Shmatikov and M.-H. Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of ESORICS 2006*, September 2006.

[50] P. Syverson, M. Reed, and D. Goldschlag. Onion Routing access configurations. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 34–40. IEEE CS Press, 2000.

[51] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[52] X. Wang, S. Chen, and S. Jajodia. Tracking anonymous peer-to-peer voip calls on the internet. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 81–91, November 2005.

[53] X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 116–130, May 2007.

[54] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao. DSSS-based flow marking technique for invisible traceback. In *Symposium on Security and Privacy*, pages 18–32. IEEE, 2007.

[55] zzz (Pseudonym) and L. Schimmer. Peer profiling and selection in the i2p anonymous network. In *Proceedings of PET-CON 2009.1*, pages 59–59, March 2009.