Critical Traffic Analysis on the Tor Network

Florian Platzer[1,*], Marcel Schäfer[2] and Martin Steinebach[1]

[1]*Fraunhofer SIT, Germany*
[2]*Fraunhofer USA CMA, USA*
E-mail: florian.platzer@sit.fraunhofer.de; mschaefer@fraunhofer.org; martin.steinebach@sit.fraunhofer.de
[*]*Corresponding Author*

Received 01 December 2020; Accepted 02 December 2020;
Publication 10 March 2021

Abstract

Tor is a widely-used anonymity network with more than two million daily users. A prominent feature of Tor is the hidden service architecture. Hidden services are a popular method for communicating anonymously or sharing web contents anonymously. For security reasons, in Tor all data packets to be send over the network are structured completely identical. They are encrypted using the TLS protocol and its size is fixed to exactly 512 bytes. In this work we describe a method to deanonymize any hidden service on Tor based on traffic analysis. This method allows an attacker with modest resources to deanonymize any hidden services in less than 12.5 days. This poses a threat to anonymity online.

Keywords: Tor, traffic analysis, hidden services, deanonymization.

1 Introduction

Every communication in the Internet leaves traces. To minimize these traces, there exist networks that intend to provide anonymity and prevent censorship

Journal of Cyber Security and Mobility, Vol. 10_1, 133–160.
doi: 10.13052/jcsm2245-1439.1015
© 2021 River Publishers

for its users, such as the Tor network [22]. Tor is the most popular and well-known low-latency anonymity network insisting to provide anonymity and prevent censorship not only to its users but also to services being offered. It has more than two million daily users, over six thousand volunteer-operated relay servers, called nodes, around the world and it provides about 170,000 services, so called *hidden services*, at the time of writing [21]. Anonymity is achieved by routing all traffic through so called *circuits* of a certain number of nodes connected via so called *onion routing* technique [10]. All packets are encrypted in such a way that each node only knows from which node the current packets were sent and to which node the packets have to be sent next. They don't know what the packets actually contain or what they are about. In addition, all Tor packets look exactly the same because they have a uniform size. Ideally, this means, both sender and receiver remain anonymous and don't have to fear prosecution for what they do. Not having to fear any consequences notoriously leads to the offering of many sensitive services. Examples are sharing of illegal content such as child pornography or illegal trading of drugs on black markets [2]. But also human rights and whistleblowing organizations such as Wikileaks and Goldballeaks make use of the Tor network. Its tools for anonymous messaging like TorChat and Bitmessage [13] are especially important to journalists in unstable and repressive regimes around the globe. Consequently, there is a large number of users who have trust in the technical implementation of Tor to protect their anonymity. Vulnerabilities in the Tor network will have a major influence to endanger users' anonymity and – in some cases – lives.

The release of Tor version 0.4.1.5 in august 2019 was a major improvement in the security of Tor. This version incorporated padding cells within introduction circuits [20]. Padding cells are a kind of traffic noise within a circuit. This was done to prevent deanonymization via traffic analysis. In this work we demonstrate the importance of this implementation for operators of Tor hidden services: We use the prior Tor version 0.3.3.7. for which no padding cells were implemented to demonstrate that a simple traffic analysis can have a big impact on the anonymity of networks that provide identical traffic packets without padding cells. While Tor serves us as use case example, this is not limited to Tor alone. The following analysis holds true for all networks with identical traffic packets but not providing padding cells.

In this work we focus on traffic analysis of a data channel, namely *introduction point circuit* in order to design attacks against the anonymity of Tor hidden services. This dose not allow an attacker to deanonymize a *specific* hidden service, but to deanonymize any hidden service. However, this is a threat to the entire ecosystem of hidden services on Tor. We will propose three independent methods each dealing with a different finding regarding the Tor network. Combining these three enables us to deanonymize hidden services:

- We will present a method to detect whether the Tor node under our control is part of an introduction point circuit.
- We will present a method to find out in which position of a circuit our Tor node is located.
- We will present a method that allows the recognition and association of traffic at nodes under our control by means of sent patterns.

Combining these three methods allows us to associate the corresponding onion address. As a result, we get the hidden service to an observed IP address of a hidden server.

This paper is structured as follows: In Section 2, we review previous work. In Section 3, we introduce the basics of circuits and hidden services in Tor. In Section 4, we present our approach to deanonymize hidden services. We evaluate our results in Section 5. In Section 6, we discuss limitations of our attack and its nevertheless significant relevance. The paper is concluded in Section 7.

2 Related Work

This section provides an overview over the past research about deanonymizing attacks against Tor hidden services. Several studies have demonstrated that hidden services could be deanonymized by locating Tor hidden servers. More precisely, that it is possible to get a hidden server's network address. Most of the following studies focused on one specific type of data channel, namely the *rendezvous point circuit*, in order to explore methods to deanonymize Tor hidden services. Another work [13] starts an attack using the data channel named *introduction point circuit* but considers only circuits of length 3.

Lasse Øverlier and Paul Syverson [16] described the first attacks that locate hidden servers in Tor. In order to carry out the attack, the attacker needs to control the first node in the server's rendezvous point circuit. The attacker establishes many rendezvous point circuits to the hidden service and sends a specific traffic pattern along the circuit. For identification of their own pattern, they used a timing analysis. Besides the counting of cells, they used information of the time the cells are sent or received and the direction of each individual cell in order to determine a circuit match. Finally, they used an attack as described in [27] to calculate statistics of the IP addresses that contacted the server. Øverlier and Syverson used a previous version of Tor in which no guard nodes were used. As a consequence to the attack published in that paper, entry guard nodes were added to the Tor design. The basic idea of guard nodes was presented in [26]. The entry guard nodes prevent this kind of attacks, especially predecessor attacks.

Zehn Ling et al. [14] described a method to discover hidden services based on the protocol level. For this it is necessary to control several entry nodes, a Tor client, a rendezvous point and a central server. The central server is used to record information of related cells forwarded from all of their controlled entities. Whenever their client establishes a circuit to an introduction point, the client sends a copy of the introduction cell to the central server. Also the rendezvous point controlled by the attacker and the server's entry node send cells to the central server. After the connection is established between the client and the hidden server, the client manipulates a cell and sends it via the rendezvous point to the server. The server cannot decrypt the cell correctly and immediately responds with a destroy cell to truncate the established circuit. All cells are registered at the central server and can be evaluated there. From that evaluation the attacker knows if all needed instances for this attack are in the same rendezvous point circuit to locate the hidden server. Ling et al. showed that this approach gives a probability of 100% that the hidden server is detected if the hidden server uses their entry nodes. They reached a probability of 24.42% to locate a hidden service in general. However, at the time of publishing their paper, it required at least 30 entry nodes within the Tor network, each with a bandwidth of 10 MB/s.

Alex Biryukov et al. [3] demonstrated an attack to deanonymize a hidden service using the rendezvous point circuit: As a client, an attacker requests a hidden service with a unique cookie. This request is forwarded from the server's introduction point and the hidden server establishes a circuit to the client's chosen rendezvous point. If the rendezvous point receives this cookie, it sends 50 padding cells back along the rendezvous point circuit. If the guard node of the server's rendezvous point circuit is controlled by the attacker, he can recognize this traffic signature (number of forwarded cells to establish the circuit and the 50 padding cells). Using this data the attacker is able to determine whether or not the node is selected as the server's guard node. That means, the attacker's node is directly connected to the hidden server and knows its IP address.

Rob Jansen et al. [11] have shown that it is possible to exploit sensitive information over a middle node instead of monitoring traffic at the edge of the network such as over the guard or exit node. They used a website fingerprint attack including a training phase, in which an attacker records as much sample traffic as possible that is similar to the transmission characteristics for a number of websites. They determined whether their node is a middle node and analyzed the traffic to detect onion service usage. This means that they were able to identify hidden websites services based on their traffic pattern.

Albert Kwon et al. [13] presented the first practical passive attack against hidden services. They utilized website fingerprinting as described in [5], [25], [24] and [4]. Traffic patterns of hidden services in Tor can have fingerprints. With execution of their website fingerprint attack, Kwon et al. were able to identify hidden service activities. For example they were able to observe the creation of introduction point circuits and identify the corresponding rendezvous point circuit that is used to access a hidden service. With this method they observed the number of incoming and outgoing cells as well as the duration of activity of a circuit. They were able to distinguish five different kinds of circuits: Client-IP (a circuit established from the client to an introduction point), HS-IP (a circuit established from an hidden server to an introduction point), Client-RP (a circuit established from the client to a rendezvous point), HS-RP (a circuit established from the hidden server to the rendezvous point) and general circuits. If an adversary controls a guard node, he can observe all traffic to and from an observed client or server and analyze the presence of hidden service activities.

All these methods show possibilities to deanonymize hidden services over rendezvous circuits making them known for being the vulnerable part of onion routing. These methods don't show that other circuits, such as the introduction point circuits, are prone to deanonymization as well.

3 Background

In this section we briefly describe the technical basics of Tor communication and the hidden service architecture. We mention key points of the Tor protocol which allow us to recognize in which position of a circuit our Tor node is located.

Tor is a low-latency anonymity network that aims to protect a user's privacy and anonymity. Tor is an overlay network that uses the Internet infrastructure. Clients can surf anonymously by proxying all of their traffic through the Tor network. It contains several thousands relay nodes world-wide connected via so called *onion routing* technique [10]. A public key infrastructure is used to encrypt traffic and support anonymous communication with each server. All traffic is routed through so called *circuits* that are frequently re-established. Whenever a client wants to send data, the client will split it in fix-sized packages and will encrypt every package in multiple layers. Each node of the circuit can decrypt one of these layers of encryption. As the package is routed through the circuit, at each relay node in this circuit, one of these layers will be decrypted. The last node within this circuit decrypts the last layer and receives the original message. The Tor design of circuits is such that each node in a circuit only knows its immediate predecessor and successor. It doesn't know any other nodes along the path. Thus, no relay node knows the entire network path from source IP to the destination IP address.

In Tor there are different types of circuits. In general a circuit is a path through several relays that connects a client to its destination and transfers user data traffic between them. All data packets, called *cells*, that are sent on this path are encrypted using the TLS protocol and have a fixed size of exactly 512 bytes. Clients and relays exchange cells in order to create circuits and attach streams to them. Every relay decrypts a layer of the encryption and forwards it to the next relay in this circuit. Tor uses circuits for different purposes, e.g. anonymous communication, connections to introduction points and connections to rendezvous points. Each circuit will be marked with a purpose tag and will only be used for this specified purpose.

We distinguish between the following types of circuits:

- *General Circuits* are used for anonymous communication and consist of at least three relays by default. A general circuit has the purpose tag *GENERAL*. In addition to immediately needed general circuits, a relay builds more general circuits as backup, in order to have further circuits ready when needed. They can also be used for other purposes, e.g. as introduction point circuits or rendezvous point circuits. In order to change the purpose of a circuit, a relay has to extend a further node to one of the existing general circuits.
- *Introduction Point Circuit*. Every client is able to provide a hidden service. In this case the client acts as hidden server. It will establish three introduction point circuits by default. These three introduction points are published openly in order to provide the service to users: if a client wants to access this hidden service, it will contact one of these introduction points. The contacted introduction point will forward the

received request to the hidden server. That way the identity of the hidden server remains hidden to the client. The introduction point circuits have the purpose tag *HS_SERVICE_INTRO*. If a client contacts one of the introduction points, it establishes a circuit to them and marks it with the purpose tag *HS_CLIENT_INTRO*.

- *Rendezvous Point Circuit*. If a client contacts an introduction point for the purpose of using a hidden service, it creates a further circuit to a rendezvous point of his choice. The client sends information of the chosen rendezvous point to the server's introduction point. This introduction point forwards the received information to the hidden server. With this information the hidden server likewise creates a circuit to this rendezvous point. After establishing the rendezvous point circuits, the whole traffic between server and client will flow through this rendezvous point. A rendezvous point circuit has the purpose tag *HS_CLIENT_REND* if the circuits are established by the client to the rendezvous point. If a circuit is established by the server to the rendezvous point, the circuit has the purpose tag *HS_SERVICE_REND*.

Hidden Services, also known as *Onion Services*, are services provided by so-called hidden servers. These servers will only accept incoming connections for hidden services via the hidden service protocol. Connection initiators will not be able to identify the IP address of the hidden server.

Hidden services are Internet services like web pages, ssh services, email accounts, login services etc. offered in the Tor network [16] [3]. In general, a hidden service could be any service that uses TCP.

The design of hidden services is based on Goldberg's rendezvous design [9]. It connects two circuits created by client and server on a rendezvous point.

If someone wants to offer a hidden service, he has to create a document called *descriptor*. A descriptor contains information such as the offered service, the public key of its server and the information which introduction points are used for that service [14]. This descriptor is published on a *directory server*. A directory server is a Tor relay and contains information about the Tor network nodes. Clients can ask the directory server for the descriptor to obtain the addresses of the hidden server's introduction points.

Figure 1 shows the communication channel of hidden services. We will describe the process below with an example:

If Bob wants to offer a hidden service, he has to establish at least one introduction point circuit, as shown in step (1) of figure 1. Bob asks (by

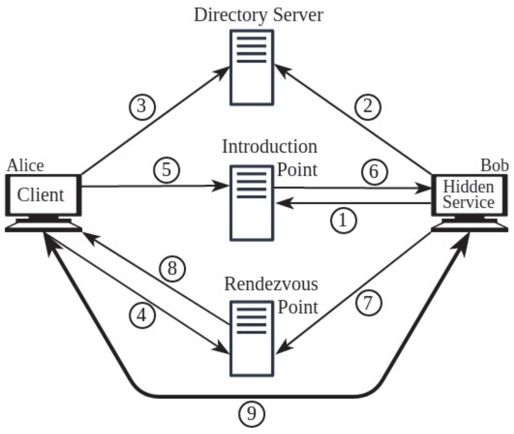


Figure 1 Hidden service architecture.

default three) chosen relays to act as his introduction points. If they accept, he will wait for requests. If one of these relays rejects Bob's request, he asks another one. Afterwards, he creates a descriptor and publishes it to a directory server (step 2).

If Alice wants to use Bob's service, she asks the directory server for the introduction points (step 3). She establishes a rendezvous point circuit to a relay of her choice (step 4). This is Alice's rendezvous point. Afterwards, she sends a request including rendezvous point information, a randomly chosen 'rendezvous cookie' to recognize Bob and the start of a Diffie-Hellman handshake to one of the introduction points (step 5). The introduction point will forward the request to Bob (step 6). Either Bob creates a circuit to the rendezvous point or he rejects Alice's request. If he creates a circuit to Alice's rendezvous point, he will send the rendezvous cookie, the second half of Diffie-Hellman handshake and the session key they now share (step 7). The rendezvous point forwards Bob's answer to Alice (step 8). Alice and Bob can communicate without knowing anything about each other through the rendezvous point (step 9). Alice does not know the IP address of Bob's server, and Bob does not know who has requested his service. Both Alice and Bob only know the IP address of the rendezvous point [16]. But the rendezvous point can recognize neither Alice nor Bob nor the data they transmit [22] or the hidden service itself [16].

Creating a circuit requires multiple cells. A cell consists of a header and a payload. The header includes a command to describe what to do with the cell's payload. Note, that two formats exist in Tor. The older "create/created" format and the newer "create2/created2" format. The same applies to "extend/extended" commands.[1] Cells are named after their commands. For example, a cell with a *relay* command is called *relay cell*. A cell with a *create2* command is called *create cell*.

Certain cell commands are utilized to create a general circuit:

- create2: A request to create a circuit to the sender of this cell.
- created2: ACK for the *create2 cell*.
- extend2: An instruction to extend the current circuit by one further node. For that, the receiver of this cell sends a *create2 cell* to the next relay. The next relay is determined within the *extend2 cell*.
- extended2: ACK for the *extend2* cell.
- relay: In case a cell includes a *relay* command, this cell is not interpreted by the node which received this cell. This node relays the cell to the next node within the circuit.
- relay_early: In order to extend a circuit by one more node the *extend2 cell* must be received within a *relay_early cell*. Relays reject any extend2 cells not received in a relay_early cell. Furthermore, if a node receives more than eight relay_early cells on a given circuit, it closes the circuit immediately. This way, the length of any circuit is limited to eight nodes. However, clients should send the first eight *relay cells* as *relay_early cells* too, in order to hide the circuit length.

In Figure 2, step 1 through step 3 show the establishment of a general circuit with three relays, called 3-hop circuit.

Additional commands are utilized to create an introduction point circuit:

- establish_intro: A request for being an introduction point.
- intro_established: An acknowledgment for the *establish_intro cell*.

Figure 2 shows all steps needed to create a 4-hop introduction point circuit (step 1 through step 5). This is the default number of nodes in introduction point circuits. Only if a node creates introduction point circuits for the first time, it will establish a 3-hop introduction point circuit. In all other cases an introduction point circuit consists of four relays.

4 Traffic Analysis to Deanonymize Hidden Services

In this section, we present our approach for deanonymizing hidden services trough traffic analysis. We figure out if a node under our control is a guard

[1]See https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. Accessed April 03, 2020.

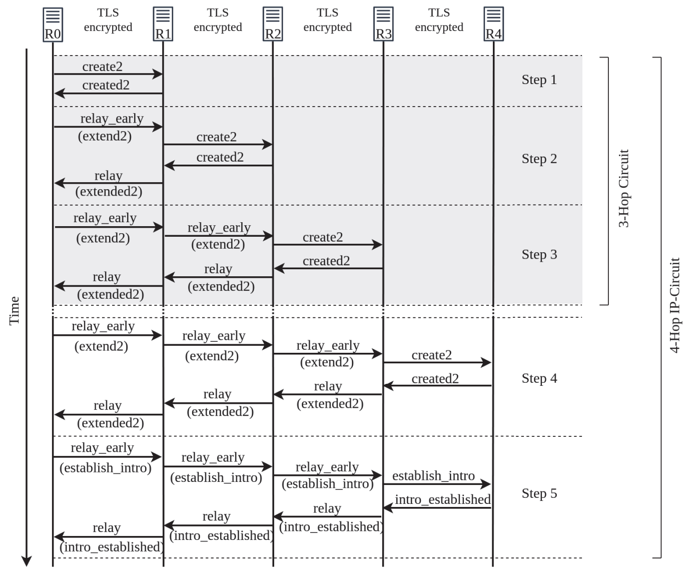


Figure 2 Creation of a Circuit. For a 3-Hop Circuit Step 1 through Step 3 is needed. For a 4-Hop IP-Circuit, Step 1 through Step 5 is needed.

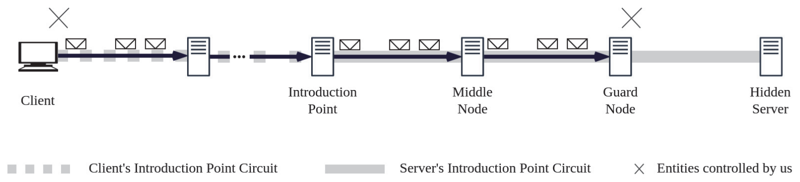

Client
Introduction Point
Middle Node
Guard Node
Hidden Server
Client's Introduction Point Circuit
Server's Introduction Point Circuit
Entities controlled by us

Figure 3 Illustration of sending traffic pattern.

node within an introduction point circuit of a hidden server. As mentioned in Section 2, a guard node is directly connected to the hidden server and thus knows its IP address. Therefore we show how we can determine the position of our relay node within the introduction point circuit (Section 4.1). Afterwards (Section 4.2), we demonstrate a method to send cells along a circuit in a time-based pattern. This we can send from any client and our relay node is able to track this sending pattern. Lastly, we present our method to deanonymize hidden services (Section 4.3). Figure 3 shows an illustration of our method for sending traffic patterns in order to determining a hidden service for a given IP address. Our client sends a traffic pattern by requesting several times a hidden service provided by a hidden server. For that, the requests are sent to the introduction point. The introduction point forwards all cells through the server's introduction point circuit and thus through the guard node under our control. Our guard node tracks the sent pattern.

4.1 Ascertain the Position Within an Introduction Point Circuit

In order to recognize the position within an introduction point circuit, we first identify the position within a circuit. Afterwards, we also identify whether the controlled node is part of an introduction point circuit.

4.1.1 Circuit position

We discovered characteristics in the traffic that are consistent for the different types of circuits and the different positions within a circuit. As every cell is encrypted and has a fixed size, observing the traffic at a relay won't tell you about its content, but it gives you information about whether a cell has been received or sent and from or in which direction it was relayed. Hence, we observed all cells flowing through our Tor node and recorded its time-stamp, direction of travel and whether the cell has been sent or received. This information suffices to obtain characteristics that identify the position of the observed relay within a circuit. This means, the order of a series of operations, i.e. sent, received, direction to or from the originator relay, can be used as a clear indicator for the position in a circuit.

A number of 18 cells is required to establish a 4-hop introduction point circuit, hence for a distinction it is sufficient to examine the first 18 cells of each circuit.

Table 1 Characters of circuit position-fingerprints

Description	Character
Cell sent	$-$
Cell received	$+$
Cell relayed to the origin node	$<$
Cell relayed away from the origin node	$>$

We encoded the operations with characters according to $-$, $+$, $<$ and $>$ for *sent*, *received*, *relayed to origin* and *relayed from origin* respectively. We encoded each cell as a two character sequence. As shown in Figure 2, the first *create cell* was sent from the origin *R0*. Thus, the *create cell* was relayed from the origin node to the first relay *R1*, which received this *create cell*. Consequently, obtained at the first relay *R1*, the character sequence +> is associated to this first *create cell*. The sequence for the response cell (*created2*) to this first cell is -<. Therefore, the sequence of both cells is +>-<. For simplification, in the following these sequences are denoted as *position-fingerprints* or simply *fingerprints*. Note though that these are not uniquely identifying.

Step 1 through step 3 in Figure 2 show a 3-hop general circuit. For a 3-hop introduction point circuit, two more cells are needed: The *establish_intro cell* and the *intro_established cell*. The overall position-fingerprint of the 3-hop introduction point circuit and obtained at the first relay (*R1*) is:

$$+>-<+>->+<-<+>->+<-<+>->+<-< \tag{1}$$

The position-fingerprint of the observed circuit from the second and the third relay is shown in Table 2.

With the same process the position-fingerprint of a 4-hop circuit can be calculated. For example, the overall fingerprint of the first relay (*R1*) in the 4-hop IP-circuit displayed in Figure 2 (step 1 through step 5) is:

$$\begin{gathered}+>-<+>->+<-<+>->+<-<+>->+<-< \\ +>->+<-<\end{gathered} \tag{4}$$

The position-fingerprints of the observed 4-hop introduction point circuit from all relays are shown in Table 2.

We can be sure that our node is the first node in a 4-hop circuit if a fingerprint shown in (4) is registered.

If the obtained fingerprints were alike those in (3) or (7), we know that our node is the last node in a given circuit.

The fingerprints (2) and (6) show that our node is a middle node of the observed circuit.

Table 2 Position-fingerprint of Introduction Point Circuits for the different relays

Circuit Type	Relay No.	Position-fingerprint	
3-hop	R1	$+>-<+>->+<-<+>->+<-<+>->+<-<$	(1)
	R2	$+>-<+>->+<-<+>->+<-<$	(2)
	R3	$+>-<+>-<$	(3)
4-hop	R1	$+>-<+>->+<-<+>->+<-<+>->+<-<$ ↲ $+>->+<-<$	(4)
	R2	$+>-<+>->+<-<+>->+<-<+>->+<-<$	(5)
	R3	$+>-<+>->+<-<+>->+<-<$	(6)
	R4	$+>-<+>-<$	(7)

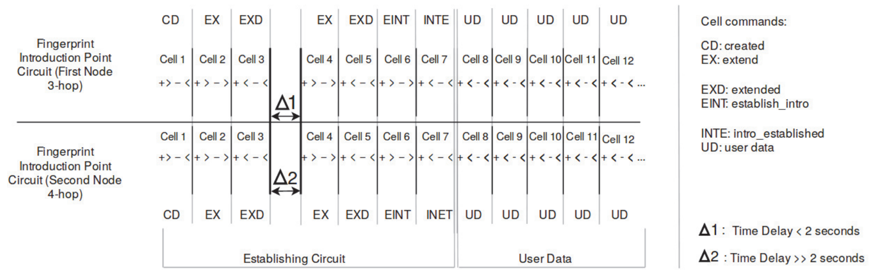

Fingerprint Introduction Point Circuit (First Node 3-hop)
Fingerprint Introduction Point Circuit (Second Node 4-hop)
CD EX EXD EX EXD EINT INTE UD UD UD UD UD
Cell 1 Cell 2 Cell 3 Cell 4 Cell 5 Cell 6 Cell 7 Cell 8 Cell 9 Cell 10 Cell 11 Cell 12
Δ1
Δ2
CD EX EXD EX EXD EINT INET UD UD UD UD UD
Establishing Circuit
User Data
Cell commands:
CD: created
EX: extend
EXD: extended
EINT: establish_intro
INTE: intro_established
UD: user data
Δ1 : Time Delay < 2 seconds
Δ2 : Time Delay >> 2 seconds

Figure 4 Comparison of Position-fingerprints of 3-hop and 4-hop Introduction Point Circuits.

In case we obtain fingerprint (1) or (5), we can not distinguish whether our node is the first one in a 3-hop circuit or the second node in a 4-hop circuit. Figure 4 illustrates such a case. For differentiation, we analyzed the time difference between receiving and sending cells of these circuits. As described in Section 3, hidden servers use 3-hop circuits whenever they create introduction point circuits for the first time. Afterwards, they utilize already established unused 3-hop circuits by extending one more node to change the purpose from *GENERAL* to *HS_SERVICE_INTRO* or *HS_CLIENT_INTRO*.

In order to identify our node as the first node in a 3-hop circuit, two facts need to be given. First, a circuit with the corresponding fingerprint (1) or (5) needs to be obtained. And secondly, all cells have to be received immediately one after another. In this case, the time delay between each cell is less than one second. However, 3-hop circuits can be extended by further cells in order to extend a fourth node. If those cells are received after a time delay of several seconds or minutes, we can associate our node as second node in a 4-hop circuit.

4.1.2 Recognizing introduction point circuits

A client decides the lifetime of a self-established circuit. However, the default setting is that circuits do not live longer than 24 hours. Circuits may be alive for different times depending on their activities. However, a circuit in Tor has a median lifetime of 10 minutes [13]. An introduction point circuit is the longest living circuit in Tor (between 18 and 24 hours[2]). After 24 hours, existing introduction point circuits are closed and new ones are established. Those are long-living connections in order to ensure a continuous availability of the hidden service through its introduction point circuit. If a circuit has a

[2]Tor 0.3.3.7 source code: or.c line 5293

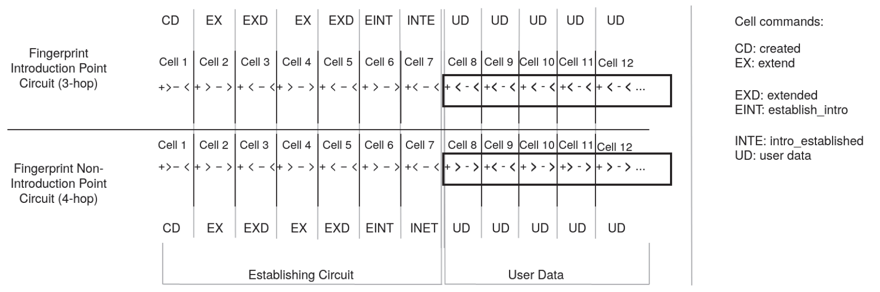


Figure 5 Comparison of Position-fingerprints of 3-hop and 4-hop Introduction Point Circuits.

long lifetime, the likelihood that this circuit is an introduction point circuit is comparably high. Another feature of introduction point circuits is, that once the introduction point circuit is established, all traffic flows in the direction of the original node, i.e. the node that initiated the communication. As a consequence, no cells will be relayed away from the hidden server. Figure 5 illustrates such a case. This is special for introduction point circuits, no other circuit in Tor exhibits this traffic pattern. Once we recognize traffic flowing from the original node back along the circuit, we can exclude this being an introduction point circuit.

4.2 Sending and Recognizing Our Own Traffic

In this section, we describe a way to send cells along an introduction point circuit, i.e. to send a traffic pattern. If our own guard node is located within this circuit, this guard node detects our sent traffic pattern.

Depending on the popularity of the hidden service, the introduction point might receive a high number of requests. A single request will be indistinguishable from the rest of the received cells. In order to recognize our sent cells, it is necessary to create a specific traffic pattern and route it through the introduction point circuit.

For the purpose of creating such a pattern, an ordinary request is sent to the hidden service that is being observed. A circuit is established to the associated hidden server's introduction point. The introduction point relays the request cells through our guard node to the hidden server. Our client establishes a rendezvous point circuit to a self-chosen relay node automatically. The established rendezvous point circuit between the hidden server and our client needs to be cut off before sending the next request. This avoids traffic

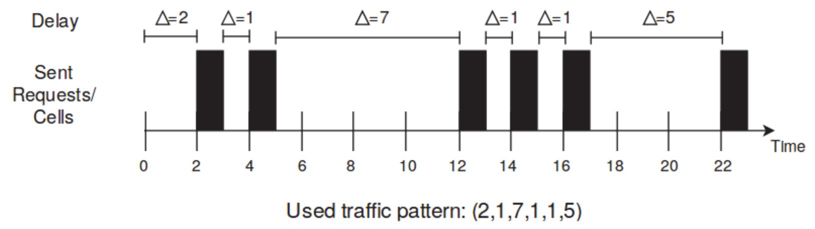


Figure 6 Example of a Traffic Pattern. Requests were sent at second 2, 4, 12, 14, 16 and 22. This results in a pattern (2,1,7,1,1,5).

flowing through the rendezvous point circuit instead of the introduction point circuit. As a consequence, our guard node will not receive cells from our client anymore. After the rendezvous point circuit has been closed, the next request for the hidden service is sent and the established rendezvous point circuit is cut again. This way, multiple cells are sent through the introduction point circuit and with that through our own guard node. After each request a certain delay is implemented before sending the next request. This delay can be modified individually over all requests and is used to implemented a traffic pattern. In other words, the pattern is the sequence of delays between each sent request. An example of a pattern of length six is shown in Figure 6.

In order to find the sent traffic pattern, our guard node observes all cells that are received from every circuit. For each received cell we calculate the difference between the timestamp of a cell with every other cell within the same circuit. As already mentioned, the traffic pattern is the delay between each sent request. Therefore, it is necessary to investigate all the calculated time differences of received cells for the sent traffic pattern.

4.3 Deanonymizing Hidden Services

In this section, a method to deanonymize hidden services is described. The prerequisites for a successful implementation of the method are:

1. We control a node within an introduction point circuit.
2. This node is a guard node.

In order to analyze whether or not these prerequisites are fulfilled the following process is used:

1. Log all incoming and outgoing cells for every circuit.
2. Obtain the lifetime of each circuit. In Section 4.1.2 it is mentioned that the median lifetime of a circuit is 10 minutes. Introduction point circuits

are alive between 18 and 24 hours. Therefore, viable candidates are circuits with a lifetime much longer than 10 minutes.
3. For all these circuits, calculate the position fingerprint as described in Section 4.1.1 based on the logged cells and identify guard nodes.
4. Evaluate the direction of received and sent cells, after finishing the establishment of each of those circuits. Viable candidates are circuits in which cells are flowing only towards the hidden server.

In addition, analyzing the IP address that our node is directly connected with could give a further hint. The IP address of each relay is publicly known within a list, also known as *consensus* [17]. If this IP address is such a publicly known Tor IP address, it will be an IP address of a Tor relay and thus not of a hidden server. In this case, our node is probably not connected to a hidden server.

Afterwards, the associated onion address of the hidden service needs to be found. This hidden service is provided by the hidden server to which a guard node is directly connected within an introduction point circuit.

1. Setup a Tor client.
2. Start a correlation attack by sending a traffic based pattern as described in Section 4.2. Use any onion address that is to be investigated.
3. On the guard node, observe all cells that belong to the selected circuits. Analyze those circuits for the sent traffic pattern.

By recognizing such a traffic pattern, it is possible to figure out that our node is the first node within an introduction point circuit of the hidden server associated with the selected onion address. Hence, the corresponding onion address of an IP address of the hidden server can be determined. As a result, a hidden service has been deanonymized.

5 Evaluation

We have implemented the proposed hidden service deanonymization approach in Section 4. In this section, we first describe the setup of our experiment and then we present our experimental results.

5.1 Experiment Setup

In order to evaluate our methods, we set up our own Tor client and a hidden service on the live Tor network. Additionally, we set up a Tor node that our

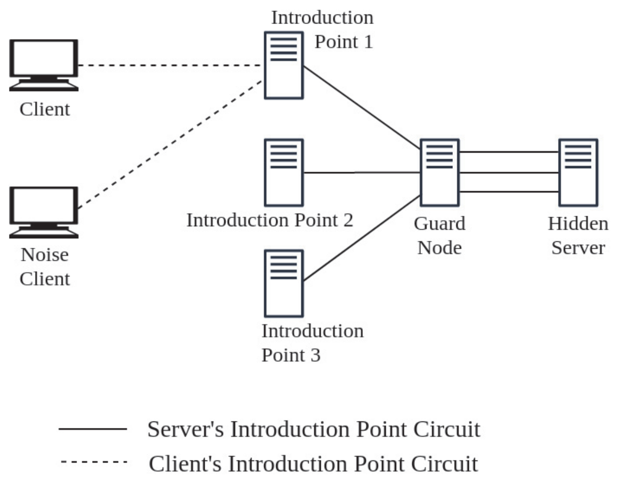


Figure 7 Experiment setup on the live Tor network.

hidden server used as guard node. On this node we monitored each incoming and outgoing cell.

With our method, we intended to recognize traffic patterns sent by us within circuits that are also used by other Tor users. To mimic this more realistic scenario that contains more “noise”, we let another client continuously send requests in order to simulate other Tor users.

With the aim of simplifying the analysis of our results, we modified the source code[3] of our client in a way that when requesting a hidden service, all cells were routed through the first introduction point only.

At the time of our experiment the most common Tor versions was 0.3.3 and 0.3.4. Neither version had implemented padding cells. The version of Tor in our experiment was version 0.3.3.7. Our setup is shown in Figure 7.

5.2 Experiment Results

Guard node detection.

In order to evaluate the detection of our guard node we calculated the position-fingerprints as described in 4.1.1. For this test case no noise client was used. We repeated the setup process 330 times. In all 330 test cases we were able to successfully confirm that our node was a guard node.

[3]Tor 0.3.3.7 source code: hs_client.c line 814, rendclient.c line 1048

IP address correlation.

For associating a hidden service to the IP address we obtained due to our guard node, we started a correlation attack against our hidden service by sending a traffic pattern.

We sent 310 traffic patterns of length eight as described in Section 4.2. Since our hidden service is unknown to other users, there is no noise on the observed introduction point circuit. No noise client was used. This means that all recognized traffic within this circuit are cells sent by us. Due to time delays in the network, it makes sense to specify a tolerance for the recognition of patterns. This tolerance could be just a few seconds. For a tolerance of two seconds we could identify 299 of our sent patterns through our guard node. This corresponds to a success rate of 96,45%.

Simulation of User Traffic.

In order to simulate other Tor users we used our noise client. With this method, we can verify whether a recognized pattern actually comes from traffic generated by ourselves or from the noise of other Tor users.

As mentioned above, it makes sense to specify a tolerance for the recognition of patterns. The higher the tolerance, the greater will be the false positive error rate (FPR). False positives means detecting certain patterns within an introduction point circuit that we did not send. An example is shown in Figure 8. Figure 9(d) shows the false positive rate for detecting a traffic pattern depending of their length. It also shows the difference between the used tolerance. Figures 9(a)–9(c) show the FPRs of hidden services with different request rates.

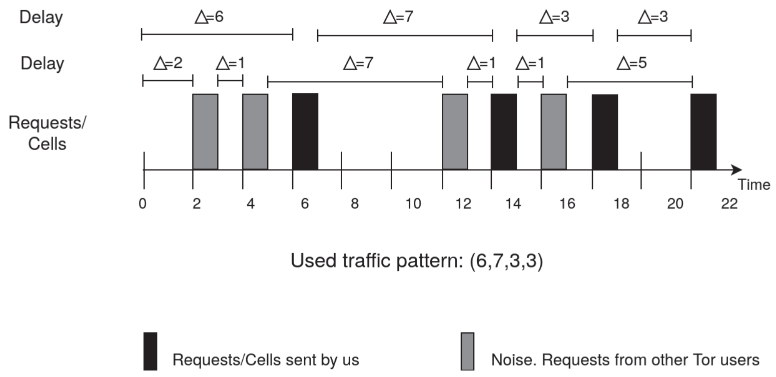
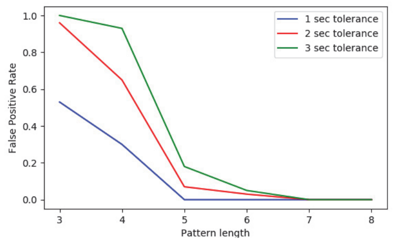
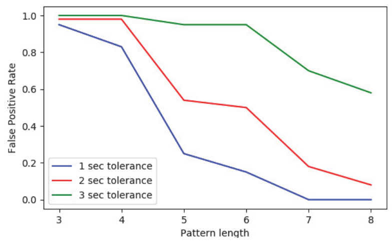

Delay
△=6
△=7
△=3
△=3
Delay
△=2
△=1
△=7
△=1
△=1
△=5
Requests/ Cells
Time
0
2
4
6
8
10
12
14
16
18
20
22
Used traffic pattern: (6,7,3,3)
Requests/Cells sent by us
Noise. Requests from other Tor users

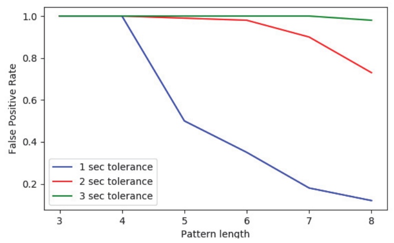
Figure 8 Example of a false positive. Sent traffic pattern (6,7,3,3) was recognized. Another traffic pattern, in this case (2,1,7,1,1,5), which was not sent by us, could be detected.



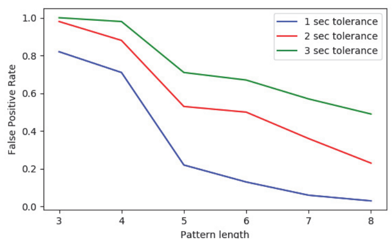
(a) FPR for services with 26,000 requests per day.



(b) FPR for services with 43,000 requests per day.



(c) FPR for services with 65,000 requests per day.



(d) FPR of (a) - (c) together.

Figure 9 FPR depending on the length of a pattern.

There are multiple works in literature about how popular hidden services can be. As maintained by Alex Biryukov et al. [2], one of the most popular non-botnet hidden service offers adult content and was requested 2,573 times per 2 hours. This corresponds to 30,876 times per day. The most well-known darknet marketplace Silk Road [6] was requested 1,175 times per 2 hours (This corresponds to 14,100 times per day). According to Owen and Savage, Silk Road was requested 8,067 times per day [18]. The Global commission on internet governance [8] claims that the most popular hidden service contains child-abuse material and was requested 168,152 times per day, [17].

Due to the different statements about the popularity of hidden services, our *noise-client* sent requests to the hidden service in randomly chosen time intervals such that the noise corresponds to either a) 26,000, b) 43,000 or c) 65,000 requests per day.

For each test case a), b), and c) we analyzed the traffic of 500 traffic patterns, without sending a particular pattern from our client.

Table 3 shows that the false positive error rate for hidden services that are requested over 26,000 times per day is 0%. This means, that none of

Table 3 False positive rates for patterns of length eight

Requests Per Day	Tolerance	False Positive Rate
26,000	2 seconds	0%
43,000	2 seconds	8.6%
65,000	2 seconds	72.8%

the 500 analyzed patterns was erroneously recognized as a sent pattern. For hidden services that are requested 43,000 times per day the false positive rate is 8.60%. The false positive rate for hidden services that are requested over 65,000 times per day is 72.80%.

Since almost all hidden services are requested less than 10,000 times per day [2], we are able to deanonymize observed hidden services with a notable high success rate.

6 Discussion and Limitations of Deanonymization

As stated in the introduction, our experiment doesn't describe an attack that is able to deanonymize any *particular* hidden service. To an already known IP address that provides a Tor service, our attack aims at finding the corresponding hidden services. This section briefly discusses its nevertheless significant relevance.

Probability of success

Guard Probability. At the time of our experiment, Tor had over 6,385 relays. 3,050 of them had the guard flag and could be used as first node for establishing a circuit [21]. According to Tor metrics [21] there were 72,522 hidden services.

The number of hidden services that select a particular guard node as their guard can be calculated using the expectation value as follows:

$$E(x) = \sum_{n=1}^{N} p_i \, x_i, \tag{1}$$

where

$$x_i = \begin{cases} 0 & \text{if node is not selected as guard} \\ 1 & \text{if node is selected as guard} \end{cases}$$

N is the total number of all hidden services. p_i is the guard probability that a particular node or hidden service chooses this guard.

The probability that a guard node will be chosen depends on its consensus weight. The consensus weight is a value that is based on bandwidth observed by the relay and bandwidth measured by the bandwidth authorities. For the purpose of traffic balancing the bandwidth of each relay is measured [1]. This measuring procedure is described in [12]. At the time of our experiment, the median bandwidth of all guard nodes was about 5 MiB/s. These nodes had a consensus weight of around 6,200[4] and therefore a guard probability of about 0.012% that a particular node chose this guard. Following the Equation (1) above, in this particular setting, the expectation value for the number of hidden services that chose this node as their guard was almost 9. The most popular guard node had an advertised bandwidth of 86.62 MiB/s and a consensus weight of 190,000. Its guard probability was 0.38%.[5] Thus, the expectation value for the number of hidden services that chose this node as their guard node was 275. Each node or hidden service will hold a chosen guard node for a random period between 30 and 60 days [3, 7]. This means that an attacker could possibly deanonymize a significant number (depending on its consensus weight) of hidden services within this time period with only one guard node.

Hidden Service Detection. From an attacker's perspective, the worst case scenario is checking all 72,522 hidden service addresses in order to find the correct hidden service when using a single Tor client to send the traffic pattern. For a pattern of length eight, the attacker has to wait about 15 seconds per pattern, if he is using a serial process. For all existing services an attacker must invest at most 12.5 days. However, if the attacker operates more than one Tor client to send the pattern, he can reduce it to a fraction of that time.

If the attack is an effort driven by a large organization or government, that could allocate a huge amount of resources, it seems realistic to deanonymize a variety of hidden services (or even any) with a high success rate.

Padding Cells. Since Tor version 0.4.1.5, hidden services add padding cells at the start of their introduction point and rendezvous point circuits. As a result, the traffic of those circuits is more similar to general traffic [20]. The following additional cells are sent:

- Two extra cells in each direction for rendezvous point circuits,
- One extra upstream cell for introduction point circuit,

[4]Note that the consensus weight may vary based on actual measurements by the bandwidth authorities.

[5]Tor Metrics: https://metrics.torproject.org/rs.html#details/3CAE19138E4E025CFB42770075DAAAF8FA0A1439, Accessed Sep 18 2019.

- Ten extra downstream cells for introduction point circuit.

Due to this implementation of padding cells, our method described in Section 4.3 will not work for the most recent version of Tor. The fingerprints described in Section 4.1.1 no longer provide us with information that we can identify the position of a Tor node within a circuit. Furthermore, padding cells prevent the detection of introduction point circuits with our method described in Section 4.1.2.

Tor as an example network. This work shows the possibility to earn critical information over a network that intends to provide both security and privacy to its users only by analyzing the network traffic. Tor is the most popular and well-known anonymity network insisting to provide anonymity and prevent censorship. Tor has implemented many security mechanisms to protect its users from deanonymization. One example is the uniform packet size to make traffic analysis more difficult. Tor is a good example to show that missing traffic noise can be a threat to the entire network through simple traffic analysis.

At the end we want to explain why we discuss research results which are seemingly outdated by the update of the Tor network and the padding mechanism. We still find the results important as on the one hand it shows that the addition of padding closed an important security gap in the protocol. This is also a lesson learned for future developments of similar networks: Traffic analysis based on timing is more common than some users imagine, for example in network data hiding and its detection. There the hidden data can be transmitted by artificial delays introduced by the stego algorithm. On the other hand the results are important to know for all users of the now obsolete version as it gives them a better idea if their anonymity could have been compromised while using Tor.

7 Conclusion

Tor hidden services establish introduction point circuits to offer their service anonymously. However, combining several traffic analysis steps, we showed that it is possible to deanonymize hidden services if there is no generated traffic noise, as was the case recently with Tor. Firstly, we showed that introduction point circuits exhibit fingerprintable traffic so that we are able to find out, whether or not a Tor node is within an introduction point circuit. Secondly, we showed a way to find out the position within an introduction point circuit. Both combined gives the opportunity to detect whether a Tor node is the guard node of a hidden service. This is critical because the guard node knows the IP address of its hidden server. But it still does not know which hidden service is based on that hidden server. We also showed that we are able to recognize traffic we sent from a client to nodes under our control using certain patterns. Sending a pattern of multiple packets to a selected hidden service will route the pattern through an introduction point circuit, and thus through a guard node. If this guard node is under our control and we are able to recognize this pattern, we know both the IP address of the hidden server and its hidden service. As a result, the hidden service is deanonymized.

We evaluated our attacks using own Tor clients and hidden services on the live Tor network. We were able to successfully identify the Tor node under our control as guard node of our own hidden service in 100% of all test cases. In 96.45% of all test cases we were able to find our sent traffic pattern on our guard node. We have shown that the false positive rate is 0% for hidden services that have been requested over 25,000 times per day. For hidden services that have been requested over 43,000 times per day the false positive rate stays as low as 8.6%. It grows to 72.80% for hidden services that have been requested over 64,000 times per day.

Acknowledgment

The joint project PANDA on which this publication is based was funded by the Federal Ministry of Education and Research under the funding codes 13N14355 and 13N14356. The authors are responsible for the content of this publication.

References

[1] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*, pages 122–134. IEEE, 2015.

[2] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. Content and popularity analysis of tor hidden services. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 188–193. IEEE, 2014.

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

[4] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238. ACM, 2014.

[5] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

[6] Nicolas Christin. Traveling the silk road: A measurement analysis of a large anonymous online marketplace. In *Proceedings of the 22nd international conference on World Wide Web*, pages 213–224. ACM, 2013.

[7] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the guards: A framework for understanding and improving entry guard selection in tor. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, pages 43–54. ACM, 2012.

[8] Centre for International Goverance Innovation. *Global Commission on Internet Goverance*, 2020 (accessed April 03, 2020). https://www.cigionline.org/initiatives/global-commission-internet-governance.

[9] Ian Avrum Goldberg and Eric Brewer. *A pseudonymous communications infrastructure for the internet*. University of California, Berkeley, 2000.

[10] David M Goldschlag, Michael G Reed, and Paul F Syverson. Hiding routing information. In *International Workshop on Information Hiding*, pages 137–150. Springer, 1996.

[11] Rob Jansen, Marc Juarez, Rafa Gálvez, Tariq Elahi, and Claudia Diaz. Inside job: Applying traffic analysis to measure tor from within. In *NDSS*, 2018.

[12] Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. Peerflow: Secure load balancing in tor. *Proceedings on Privacy Enhancing Technologies*, 2017(2):74–94, 2017.

[13] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[14] Zhen Ling, Junzhou Luo, Kui Wu, and Xinwen Fu. Protocol-level hidden server discovery. In *INFOCOM, 2013 Proceedings IEEE*, pages 1043–1051. IEEE, 2013.

[15] Karsten Loesing, Werner Sandmann, Christian Wilms, and Guido Wirtz. Performance measurements and statistics of tor hidden services. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 1–7. IEEE, 2008.

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

[17] Gareth Owen and Nick Savage. The tor dark net. 2015.

[18] Gareth Owen and Nick Savage. Empirical analysis of tor hidden services. *IET Information Security*, 10(3):113–118, 2016.

[19] The Tor Project. *Configuring Onion Services for Tor*, 2019 (accessed Mai 06, 2019). https://2019.www.torproject.org/docs/tor-onion-service.html.en.

[20] The Tor Project. *New release: Tor 0.4.1.5*, 2020 (accessed April 03, 2020). https://blog.torproject.org/new-release-tor-0415.

[21] The Tor Project. *Tor metrics portal*, 2020 (accessed December 01, 2020). https://metrics.torproject.org.

[22] Paul Syverson, R Dingledine, and N Mathewson. Tor: The secondgeneration onion router. In *Usenix Security*, 2004.

[23] Tao Wang. Website fingerprinting: Attacks and defenses. 2016.

[24] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*, pages 143–157, 2014.

[25] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM, 2013.

[26] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Defending anonymous communications against passive logging attacks. In *2003 Symposium on Security and Privacy, 2003.*, pages 28–41. IEEE, 2003.

[27] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

Biographies

Florian Platzer is a research assistant at the Fraunhofer Institute for Secure Information Technology. He is part of the PANDA project at Fraunhofer SIT. The PANDA project is an interdisciplinary project researching the darknet. Within this project he is responsible for the computer science part. Florian studied IT security at the Technical University of Darmstadt, Germany. He wrote his master thesis about deanonymization of Tor hidden services.

Marcel Schäfer serves as Senior Research Scientist for the Fraunhofer USA Center for Experimental Engineering CESE in Maryland since 2019. From 2009 to 2018 he was with Fraunhofer Institute for Secure Information Technologies SIT in Germany. With a Master's degree in mathematics from the University of Wuppertal, Germany and a PhD in computer science from the Technical University of Darmstadt, Germany, he consults and teaches for topics on dark web, privacy networks and anonymous communication, and also serves as a subject matter expert for privacy, e.g. GDPR and data anonymization. As PI, Co-PI and researcher Dr. Schäfer has lead and worked in various projects that discover new challenges and opportunities broadly spread over the fields of cybersecurity and software engineering in both the public and private sector.

Martin Steinebach is the manager of the Media Security and IT Forensics division at Fraunhofer SIT. From 2003 to 2007 he was the manager of the Media Security in IT division at Fraunhofer IPSI. He studied computer science at the Technical University of Darmstadt and finished his diploma thesis on copyright protection for digital audio in 1999. In 2003 he received his PhD at the Technical University of Darmstadt for this work on digital audio watermarking. In 2016 he became honorary professor at the TU Darmstadt. He gives lectures on Multimedia Security as well as Civil Security. He is Principle Investigator at ATHENE and represents IT Forensics and AI security. Before he was Principle Investigator at CASED with the topics Multimedia Security and IT Forensics. In 2012 his work on robust image hashing for detection of child pornography reached the second rank “Deutscher ITSicherheitspreis”, an award funded by Host Görtz.