2013 IEEE Symposium on Security and Privacy

Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization

Alex Biryukov, Ivan Pustogarov, Ralf-Philipp Weinmann
University of Luxembourg
{*alex.biryukov,ivan.pustogarov,ralf-philipp.weinmann*}*@uni.lu*

***Abstract*—Tor is the most popular volunteer-based anonymity network consisting of over 3000 volunteer-operated relays. Apart from making connections to servers hard to trace to their origin it can also provide receiver privacy for Internet services through a feature called "hidden services".**

In this paper we expose flaws both in the design and implementation of Tor's hidden services that allow an attacker to measure the popularity of arbitrary hidden services, take down hidden services and deanonymize hidden services. We give a practical evaluation of our techniques by studying: (1) a recent case of a botnet using Tor hidden services for command and control channels; (2) Silk Road, a hidden service used to sell drugs and other contraband; (3) the hidden service of the DuckDuckGo search engine.

***Keywords*-Tor; anonymity network; privacy; hidden services**

I. Introduction

Research into low-latency anonymity networks has mostly focused on sender anonymity, i.e. allowing users to connect to network resources without disclosing their network address to the destination. Nonetheless, to guarantee freedom of speech, responder privacy[1] is equally if not even more important. Allowing the people not only to access information anonymously but also to publish anonymously is an important aspect of nurturing democracy. Similarly, providing internet services without disclosing their location and owner puts a constraint on what attacks can be performed by adversaries.

Tor [7] arguably is the most popular and well-researched low-latency anonymity network, providing sender privacy for internet services to its users; additionally responder privacy can be achieved with Tor by making TCP services available as *hidden services*. While the first generation of Tor's hidden service design has been described in the original design paper, the current version of Tor is using a revised design [25]. Justifications of the design choices and attack scenarios considered are given in [15].

This paper analyzes the security of Tor hidden services. We look at them from different attack perspectives and provide a systematic picture of what information can be obtained with very inexpensive means. We focus both on attacks that allow to censor access to targeted hidden services as well as on deanonymization of hidden services. As the result we believe that many components of the current Tor HS protocol should be improved and while a short term patch may mitigate some of the problems, a more complex approach is required both in terms of efficiency and in terms of privacy.

We also study *deployed hidden services*. For instance, we apply our findings to a botnet which makes its command and control center available to bots as a Tor hidden service (we extracted its onion address by analyzing a malware sample) and extrapolate its size by counting the number of hidden service requests.

Contributions:

- We give a method to measure the popularity of any hidden service without the consent of the hidden service operator.
- We show how connectivity to selected hidden services can be denied by impersonating all of their responsible hidden services directories.
- We demonstrate a technique that allows one to harvest hidden service descriptors (and thus get a global picture of all hidden services in Tor) in approximately 2 days using only a modest amount of resources.
- We show how to reveal the guard nodes of a Tor hidden service.
- We propose a large-scale opportunistic deanonymization attack, capable of revealing IP addresses of a significant fraction of Tor's hidden services over a one year period of time.

Most of our attacks are made practical and cost efficient by two implementation deficiencies in the current versions of Tor[2]: (1) Tor relays can cheat and inflate their bandwidth in the consensus despite bandwidth measurements; this makes them more likely to be chosen by the path selection algorithm. (2) Using a technique called "shadowing" we can phase relays in and out of the consensus at will without them losing their flags, allowing us to defeat countermeasures against Sybil [8] attacks.

Ethical considerations: Attacks against Tor can be simulated in dedicated simulators such as Shadow [13]. However, deployed hidden services are not well studied. Until now there have been no statistics about the number of hidden services or their usage statistics. Henceforth, we deem experiments on the live Tor network that do

[1]Traditionally called recipient privacy in the case of mix networks.

[2]We consider versions up to tor v0.2.4.6-alpha, which was the most current version at the time of submission.

1081-6011/13 $26.00 © 2013 IEEE
DOI 10.1109/SP.2013.15

not intentionally cause degradation of the network and its services to be worthwhile and necessary to enhance the scientific understanding of hidden services.

Our goal was at no time to perform a full deanonymization of any target that was not under our control but rather to show that this would be possible. Moreover, since we are conducting anonymity research, we did not disclose information about guard nodes identified to third parties but rather discarded identifying data after the experiments.

Roadmap: We start by presenting and discussing related work in Section II. In Section III, we provide the necessary background to understand Tor hidden services and our attacks. In section IV, we show how countermeasures against Sybil attacks implemented in Tor can be circumvented.

In Section V, we show how an attacker can control the hidden service directories of any hidden service. We also demonstrate a technique that allows us to harvest hidden services quickly and efficiently. Section VI shows how to confirm that a Tor relay serves as a guard node of a given hidden service, allowing us to determine the IP address of the hidden service if the guard node is under our control. In Section VII, we show how an attacker can discover the guard nodes of a hidden services. In Section VIII, we discuss countermeasures that can be implemented to defend against our attacks. Section IX concludes the paper.

A. Examples of hidden services analyzed

A botnet using hidden services: In April 2012, an "ask me anything" thread (AMA) on the social news website Reddit appeared in which an anonymous poster, allegedly a malware coder and botnet operator, claimed to be operating a botnet with its command and control center running as a Tor hidden service [1]. The malware installed on the clients was described to be a modified version of ZeuS.

Subsequently, a thread on the tor-talk mailing list appeared [14] in which apparently the same botnet was discussed. We obtained samples of this malware and found the properties of the malware matched: just like described in the AMA thread, it was using a modified UnrealIRC 3.2.8.1 server[3] for one of the command and control channels and included a Bitcoin miner. This was the first publicly documented instance of a botnet in the wild using Tor hidden services. While there had been a talk given at DEFCON in 2011 [4] about how hidden services could be used to protect botnets from takedowns of their command and control structure, previously no such malware had been observed.

Interestingly, not one but two hidden services were operated for command and control: the standard HTTP based channel[4] that ZeuS uses for command as well as an IRC based one[5]. Furthermore, the malware creates a hidden service (on port 55080) on each install, which allows the botnet operator to use the infected machine as a SOCKS proxy for TCP connections through the hidden service. While the hidden service is constantly running, the command to enable SOCKS proxy functionality needs to be given through the IRC command and control channel. In the version of the malware we analyzed, a Tor v0.2.2.35 binary was executed by injecting it into a svchost process.

In September 2012, G Data Security described a sample of apparently the same malware in a blog post [11]; a more thorough analysis of the botnet was published by Claudio Guarnieri of Rapid7 in December 2012 [12].

Black Markets on Hidden Services: A number of black markets exist on Tor hidden services. Silk Road is by far the most widely known, even triggering requests from U.S. senators to the U.S. Attorney General and the Drug Enforcement Agency (DEA) to request it to be shut down [22].

Silk Road is a market that operates mostly in contraband goods using Bitcoin as currency. According to a recent study primarily narcotics and other controlled substances are sold on this platform [5]. This study estimates the Silk Road revenue at over USD 1.9 million per month – aggregated over all sellers – with a 7.5% cut going to the Silk Road operators.

II. Related work

The first published attacks against Tor hidden services were presented by Øverlier and Syverson in [19]. They targeted a previous version of the hidden services design in which no entry guard nodes were used. In the scenario described, the attacker needs to control one or more Tor relays; the idea being that given enough connection attempts, one of the attacker's relays will be chosen as the first hop of the rendezvous circuit established by the hidden service.

To mount the attack, the attacker establishes many rendezvous circuits to the hidden service and sends a specific traffic pattern along the circuits. She uses traffic correlation to determine if one her nodes was chosen as a part of the circuits. Once an attacker's Tor relays is chosen as the first hop of a circuit, the location of the hidden service is immediately revealed. As the result of the paper, entry guard nodes were added to the Tor hidden services specification which prevents the attack in the current version of Tor. The basic idea of guard nodes (originally named *helpers*) was introduced by Wright et al. in [17].

Valet services were proposed by Øverlier and Syverson as an extension to the hidden services concept to strengthen DoS resilience of hidden services. This is achieved by introducing an additional layer of protection for introduction points [20].

Another approach was presented in [18] and [26]. These attacks are based on the observation that the system clocks of computers drift depending on the temperature of the CPU. An attacker observes timestamps from a PC connected to

[3]Unfortunately, not the backdoored distribution by ac1db1tch3z

[4]`mepogl2rljvj374e.onion:80`

[5]`eoqallfil766yox6.onion:16667`

the Internet and watches how the frequency of the system clock changes when repeatedly connecting to the hidden service, causing its CPU load to rise. The attacker requests timestamps from all candidate servers and finds the one exhibiting the expected clock skew pattern. One drawback of this attack is that it assumes a closed-world model, i.e. the list of possible candidate servers needs to be be known by the attacker in advance. Also, the degree of the scalability of the attack is limited by the fact that the attacker needs to probe each server in the list.

Some of the building blocks for our attacks have already been mentioned in the literature. In [15], Loesing mentions that it is unavoidable for hidden services descriptors to be collected over a long period of time. However in this paper, we show how to perform harvesting fast and cheap.

The forensic problem of placing identifiable fingerprints in the log files of a machine running a hidden service trough service queries is considered in [24], [10]. This fingerprint is then used to prove that the confiscated machine in fact hosted a particular content, assuming that requests are logged.

III. Background

Tor is a low-latency anonymity network based on the ideas of onion routing and telescoping. Clients have anonymous communication to a server by proxying their traffic through a chain of three Tor relays. Specifically, prior to sending the data, a client chooses three Tor relays and uses public key cryptography to negotiate symmetric session keys with them, establishing a *circuit*. Whenever a client wants to send a piece of data he packs it into Tor cells and encrypts them with multiple layers of encryption using the session keys. As the cells travel along the circuit, each relay strips off one layer of encryption. Hence the server receives the original piece of data while each relay along the path knows only which relay it received the Tor cell from and which relay it forwarded the cell to.

Tor *Hidden Services* are a feature which was introduced in 2004 to add responder anonymity to Tor. Specifically, hidden services allow running an Internet service (e.g. a Web site, SSH server, etc.) so that the clients of the service do not know its actual IP address. This is achieved by routing all communication between the client and the hidden service through a *rendezvous point* which connects anonymous circuits from the client and the server.

The Tor hidden service architecture is comprised of the following components (see Figure 1):

- Internet service which is available as Tor hidden service;
- Client, which wants to access the Internet service;
- Introduction points (IP): Tor relays chosen by the hidden service and which are used for forwarding management cells necessary to connect the Client and the hidden service at the Rendezvous point;

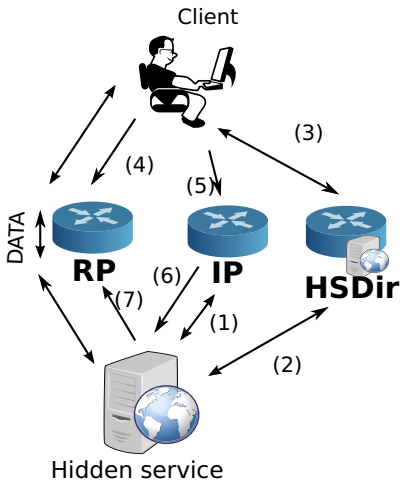

Figure 1. Tor hidden services architecture

- Hidden service directories (HSDir): Tor relays at which the hidden service publishes its descriptors and which are communicated by clients in order to learn the addresses of the hidden service's introduction points;
- Rendezvous point (RP): a Tor relay chosen by the Client which is used to forward all the data between the client and the hidden service.

A. Hidden service side

In order to make an Internet service available as a Tor hidden service, the operator (Bob) configures his Tor Onion Proxy (OP) which automatically generates new RSA key pair. The first 10 bytes of the SHA-1 digest of an ASN.1 encoded version of the RSA public key become the identifier of the hidden service. The OP then chooses a small number of Tor relays as introduction points and establishes new introduction circuit to each one of them (step 1 in Figure 1).

As the next step (step 2), Bob's OP generates two service descriptors with different IDs, determines which hidden services directories among the Tor relays are responsible for his descriptor and uploads the descriptor to them. A hidden services directory is a Tor relay which has the HSDir flag. A Tor relay needs to be operational for at least 25 hours to obtain this flag.

The hidden service descriptors contain the descriptor ID, the list of introduction points and the hidden service's public key.

B. Client side

When a client (Alice) wants to communicate with the hidden service, she needs a pointer to this service, which needs to be transmitted out of band. The pointer is the hostname of the form "z.onion", where z is the base-32 encoded hidden service identifier described above. She then computes the descriptor IDs of the hidden service (see the

expression in section III-C) and the list of responsible hidden service directories and fetches the descriptors from them (step 3).

In order to establish a connection to a given hidden service Alice's OP first builds a rendezvous circuit (step 4). It does this by establishing a circuit to a randomly chosen Tor relay (OR), and sending a `RELAY_COMMAND_ESTABLISH_RENDEZVOUS` cell to that OR. The body of that cell contains a Rendezvous cookie (RC). The rendezvous cookie is an arbitrary 20-byte value, chosen randomly by Alice's OP. Alice chooses a new rendezvous cookie for each new connection attempt. Upon receiving a `RELAY_COMMAND_ESTABLISH_RENDEZVOUS` cell, the OR associates the RC with the circuit that sent it. Alice builds a separate circuit to one of Bob's chosen introduction points, and sends it a `RELAY_COMMAND_INTRODUCE1` cell containing the IP address and the fingerprint of the rendezvous point, the hash of the public key of the hidden service (PK_ID), and the rendezvous cookie (step 5).

If the introduction point recognizes PK_ID as the public key of a hidden service it serves, it sends the body of the cell in a new `RELAY_COMMAND_INTRODUCE2` cell down the corresponding circuit (step 6).

When Bob's OP receives the `RELAY_COMMAND_INTRODUCE2` cell, it decrypts it using the private key of the corresponding hidden service and extracts the rendezvous point's nickname as well as the rendezvous cookie. Bob's OP builds a new Tor circuit ending at Alice's chosen rendezvous point, and sends a `RELAY_COMMAND_RENDEZVOUS1` cell along this circuit, containing RC (step 7). Subsequently, the rendezvous point passes relay cells, unchanged, from each of the two circuits to the other.

In this way, the client knows only the rendezvous point. Neither does the hidden service learns the actual IP address of the client nor does the client learn the IP address of the hidden service.

C. Choosing responsible HSDirs

A hidden service determines if a hidden services directory is responsible for storing its descriptor based on the descriptor's ID and the directory's fingerprint[6].

Descriptor identifiers change periodically every 24 hours and are computed as follows:

```
descriptor-id = H(public-key-id || secret-id-part)
secret-id-part = H(descriptor-cookie || time-period ||
replica-index)
```

The field `descriptor-cookie` is an optional field. If present, it prevents non-authorized clients from accessing the hidden service. The field *time period* denotes the number of days since the epoch. This is used to make the responsible directories change periodically. The *replica index* is used to create different descriptors identifiers so that the descriptor is distributed to different parts of the fingerprint range.

[6]Each Tor relay is identified by SHA-1 digest of its public key. We call this digest as the relay's fingerprint.

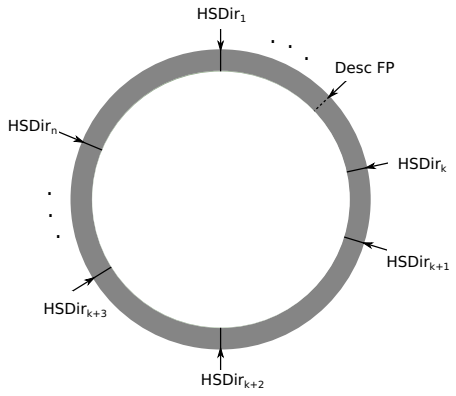

Figure 2. Tor hidden services fingerprints circular list

After computing the descriptor identifiers, a hidden service determines which directory nodes are responsible for storing the descriptor replicas. To do this the hidden service arranges the directories using their fingerprints in a closed fingerprint circle and chooses as hidden service directories the three closest relays in positive direction (fingerprint value of them is greater than the fingerprint value of the hidden service).

According to the current Tor implementation, a hidden service generates and publishes two replicas of its descriptor which results in 2 sets of 3 hidden service directories with consecutive fingerprints.

As an example, consider the circle of fingerprints depicted in Figure 2 and assume that one of the hidden service descriptor IDs is between fingerprints of relays HSDir_{k-1} and HSDir_k. In this case the hidden service directories serving the descriptor are relays with fingerprints HSDir_k, HSDir_{k+1}, and HSDir_{k+2}. The fingerprint of HSDir_k is the first following the descriptor ID. We call this HSDir relay the *first responsible hidden service directory* for the descriptor ID.

The list of all Tor relays is distributed by the Tor authorities in the *consensus* document. The consensus is updated once an hour by the directory authorities and remains valid for three hours. Every consensus document has a "valid-after" (VA) time, a "fresh-until" (FU) time and a "valid-until" (VU) time. The "valid-after" timestamp denotes the time at which the Tor authorities published the consensus document. The consensus is considered fresh for one hour (until "fresh-until" has passed) and valid for two hours more (until "valid-until" has passed). According to the

current implementation clients download the next consensus document in (FU + 45 mins; VU - 10 mins) interval.

D. Guard nodes

Being a low-latency anonymity network Tor is vulnerable to the traffic confirmation attacks: if an adversary can monitor the edges of a Tor circuit, she can confirm who is communicating. This is quite dangerous for hidden services since by design the attacker always controls one edge of the connection. If the entry nodes of the circuit were chosen uniformly from the whole set of Tor relays, the probability of the attack would approach 1 when the number of circuits to the hidden service established by the attacker increased.

In order to significantly reduce the probability of the traffic confirmation attack Tor developers introduced the concept of *entry guard nodes*. Tor initially selects a set of three guard nodes. Whenever less than two guard nodes from the set are reachable, new guard nodes are chosen. A guard node remains in the set for a random duration between 30 and 60 days. Then it is marked as expired and removed from the set. Whenever a circuit is established, one node from the set of Guard nodes is used for the first hop.

IV. Flags and bandwidth inflation

According to the current Tor specification, the maximum number of Tor relays on a single IP address that Tor authorities include to the Consensus document is 2. This restriction is enforced by the directory authorities when they cast their votes for the consensus. If more than two relays are running on the same IP address, only two relays with the highest-most measured bandwidth will appear in the consensus document. This prevents an attacker from performing the Sybil attack described by Bauer et al. [2], in which an attacker floods the network with dummy Tor relays.

However, by inspecting the Tor source code we noticed that while only two relays per IP appear in the Consensus, all running relays are monitored by the authorities; more importantly, statistics on them is collected, including the uptime which is used to decide which flags a relay will be assigned.

We call relays appearing in the consensus *active relays* and those which run at the same IP address but do not appear in the consensus *shadow relays*. Whenever one of the active relays becomes unreachable and disappears from the consensus, one of the shadow relays becomes active, i.e. appears in the consensus. Interestingly, this new active relay will have all the flags corresponding to its real run time and not to the time for which it was in the consensus. We call this technique *shadowing*.

The path selection algorithm of Tor selects nodes at random, with a probability proportional to the bandwidth advertised for the node in the consensus document. Hence it is of interest to an attacker to artificially inflate the bandwidth of her nodes, in order to increase the chance of of them being included in the path (note that although some of the attacks presented in this paper have been made more efficient using bandwidth inflation they by no means depend it).

Originally, directory authorities announced self-reported bandwidth values of the relays in the consensus document. The general concept of bandwidth inflation was first exploited in [19] to make attacks against hidden services more efficient; again the authors of [2] made use of the same design flaw to make end-to-end traffic correlation attacks feasible with a modest amount of resources.

As a reaction to these attacks, bandwidth scanners were introduced. In the current design, Tor authorities not only take into account the self-reported bandwidth values but also actively measure the bandwidth. A subset of directory authorities operate a set of bandwidth scanners which periodically choose two-hop exit circuits and download predefined files from a particular set of IP addresses (according to the current source code, there are two such IP addresses). The bandwidth of a relay shown in the consensus depends on the self-reported bandwidth B_{rep} and the bandwidth measurement reports B_{meas} by the Tor authorities. The weak point of this approach is the fact that the scanning can be reliably detected by relays that want to cheat. To inflate our bandwidth we then provide more bandwidth for authorities' measurement streams while throttling bandwidth for all other streams. This results in a high bandwidth value shown in the consensus while keeping the traffic expenses at a low level.

When doing bandwidth measurements, authorities establish two-hop circuits. Thus it is sufficient for cheating non-exit nodes to provide more bandwidth for streams which originate at IP addresses of authorities and throttle all other streams. As an improvement the attacker can take into account that for bandwidth measurements authorities download files which are known. Taking this into account, the attacker can drop circuits which carry a traffic pattern inconsistent with these downloads.

We have implemented this method of bandwidth cheating and were able to inflate the bandwidth of our relays more than ten fold; while the consensus showed bandwidth values of approximately 5000 kBytes/sec per relay, they only provided 400 kBytes/sec of real bandwidth to Tor clients each.

V. Catching and tracking hidden service descriptors

In this section we study the security of descriptor distribution procedure for Tor hidden services. We show how an attacker can gain complete control over the distribution of the descriptors of a particular hidden service. This undermines their security significantly: before being able to establish a connection to a hidden service, a client needs to fetch the hidden service's descriptor; unless it has it cached from a

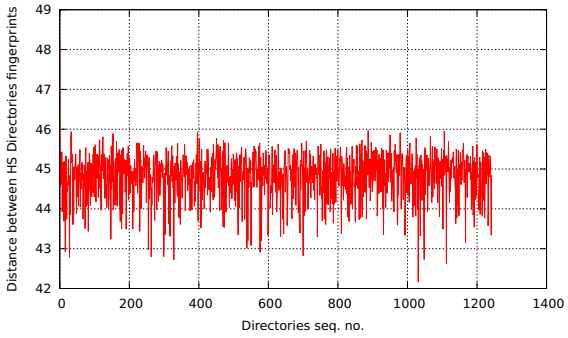

Figure 3. Distances between HS directories fingerprints, $\log_{10}$ scale.

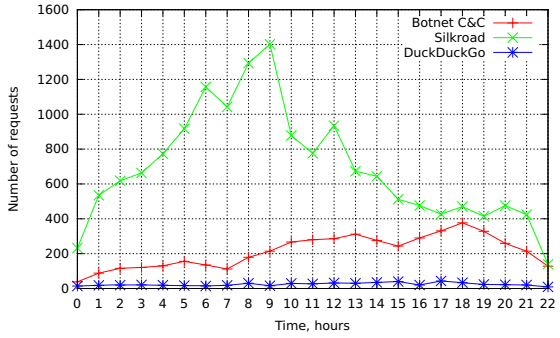

Figure 4. Hidden service descriptor request rate during one day.

prior connection attempt. Thus, should the attacker be able to control the access to the descriptors, the hidden service's activity can be monitored or it can be made completely unavailable to the clients.

A. *Controlling hidden service directories*

As mentioned in the background section, the list of responsible hidden service directories depends on the current consensus document and the descriptor IDs of the hidden service. In this subsection, we explain how to inject relays into the Tor network that become responsible for the descriptors of the hidden service. This immediately translates into the problem of finding the right public keys, i.e. the keys with fingerprints which would be in-between the descriptor IDs of the hidden service and the fingerprint of the first responsible hidden service directory.

Figure 3 shows the distances between consecutive hidden service directories (in $\log_{10}$ scale) computed for a randomly picked consensus document in November 2012. The average value is 44.8 and the minimum value is 42.16. This means that we need to find a key with a fingerprint which would fall into an interval of size $10^{44.8}$ on the average. This takes just a few minutes on a modern multi-core computer.

Just like any Tor client, an attacker is able to compute the descriptor IDs of the hidden service for any moment in the future and find the fingerprints of expected responsible HS directories. After that she can compute the private/public key pairs so that SHA-1 hash of the public keys would be in-between the descriptor ID and the fingerprint of the first responsible hidden service directory. The attacker then runs Tor relays with the computed public/private keys pairs and waits for 25 hours until they obtain the HSDir flag. When the attacker's relays appear in the consensus as hidden service directories, they will be used by the hidden service to upload the descriptors and by the clients to download the descriptors. In this way the attacker can gain control over all the responsible HS directories for a particular service by injecting 6 Tor relays with precomputed public keys. This allows her to censor a hidden service of her choice or gather its usage statistics.

As a proof of concept we used this approach to control one of the six hidden service directories of the discovered Tor botnet, the Silk Road hidden service, and the DuckDuckGo hidden service. We tracked these for several days and obtained the following measurements: (1) The number of requests for the hidden service descriptor per day (see Tables I and II) and (2) the rate of requests over the course of a day, which is shown in Figure 4 (each point corresponds to the number of hidden service descriptor requests per one hour).

Column 1 of Table I and columns 2 and 4 of Table II show the number of requests for a particular hidden service descriptor per day. Columns "Total" show the total number of descriptors requests (for any hidden services descriptor) served by the hidden service directory per day. The hidden service tracked in Table I is the IRC C&C service.

Table I
POPULARITY OF THE DISCOVERED BOTNET

Date	Botnet descriptor	Total
13 Jul	1408	6581
14 Jul	1609	2392
15 Jul	1651	4715
16 Jul	1448	6852
25 Jul	4004	6591
26 Jul	4243	4357
27 Jul	4750	4985
28 Jul	4880	7714
29 Jul	4977	9085

Table II
POPULARITY OF SILK ROAD AND DUCKDUCKGO

Date	Silk Road	Total	DuckDuckGo	Total
09 Nov	19284	27363	502	2491
10 Nov	15427	16103	549	5621
11 Nov	15185	15785	543	3899
12 Nov	15877	16723	549	10910

Descriptors are cached by the Tor process in RAM for 24 hours. Hence, as long as a computer is not restarted,

we will see at most one descriptor request every 24 hours, even if the long-lived circuit to the IRC server is repeatedly dropped; moreover suspending the computer will not cause the descriptor to be requested again. On the other hand, multiple power cycles per day lead to overcounting the size of the botnet. Hence, from Table I one can estimate that the size of the botnet was in the range 12,000 – 30,000 infected machines.

This is a very rough approximation since bots can request the descriptor several times per day, each time when the infected computer is turned on. By looking at the descriptor request rate against time we can infer that the bulk of the botnet resides in the European time-zone.

To protect a hidden service from making it unavailable, Loesing [15] proposes that hidden services periodically check if their descriptors can be downloaded from the responsible directories[7]. If some hidden service directory consistently refuses the fetch request, the hidden service files a complaint to the Tor directory authorities. The complaint includes the hidden service descriptor.

Having received the complaint, the authorities upload the descriptor to the directory and try to fetch it. If the fetch consistently fails again, the authorities remove HSDir flag from the directory. If the directory demonstrates the same behaviour for a long period of time, the relays in the IP range can be banned from obtaining the HSDir flag.

This protection is based on the assumption that it is hard to acquire fresh IP addresses. The availability of large computing platforms that can be rented on an hourly basis has made this technique ineffective, unless the entire IP ranges of such platforms are banned; new IP addresses can be easily obtained, e.g. by restarting instances on Amazon EC2.

B. Efficient harvesting of Tor HS descriptors

It is of a particular interest to collect the descriptors of all hidden services deployed in Tor. We will show how an attacker can use this collection to opportunistically deanonymize any hidden service which chose one of the attacker's nodes as one of its entry guards. The IP addresses of these hidden services can be revealed in a matter of seconds using a traffic correlation attack, as we will show later.

It is clear that an attacker can operate several hidden service directories and collect hidden service descriptors over a long period of time. However, since there were more than 1200 hidden service directories at the time of this writing it can take the attacker significant amount of time to collect enough hidden service descriptors.

To collect the descriptors of all hidden services in a short period of time, a naïve attack requires to run many Tor relays from a non-negligible number of IP addresses. Assume that a hidden service descriptor's ID falls into some gap[8] on the fingerprint circle. The hidden service uploads its descriptor to the three hidden service directories with the next greater fingerprints. This means that each hidden services directory receives descriptors with identifiers falling into two gaps preceding the hidden service directory's fingerprint. This in turn means that the attacker needs to inject a hidden service directory into every second gap in the fingerprint circle to collect all hidden service descriptors. Thus she would need to run more than 600 Tor relays for 27 hours. This requires more than 300 IP addresses, given that the attacker is allowed to run only 2 Tor relays on a single IP address.

However, given the observations in the previous section, we can collect the hidden service descriptors much more efficiently. In this subsection, we show how to reduce the number of IP addresses to approximately 50 (depending on the exact number of hidden service directories in the consensus). Our approach is based on shadow relays described in the previous section. An attacker can use this artifact of Tor's design as follows. She can rent 50 IP addresses and run 24 relays on each of them for 25 hours thus running 1200 Tor instances in total; 100 of them should appear in the consensus. The fingerprints of the public keys of the relays should fall into every second gap in the fingerprint circle. At the end of 25 hour time period all of the relays will have HSDir flags but only 100 of them will appear in the consensus and the rest will be shadow relays. The idea is to gradually make active relays unreachable to the Tor authorities so that shadow relays become active and thus gradually cover all gaps in the circular list during 24 hours.

It should be noted that the descriptor IDs of hidden services (and hence the responsible hidden service directories) change once per 24 hours and the time of the day when they change can be different for different hidden services. Since each hour the attacker covers only a fraction of the gaps on the fingerprint circle, the location of the descriptor can change from a gap not yet covered by the attacker to a gap already covered. Thus, if the attacker makes only one pass over the fingerprint circle during the day, she may not catch some descriptors. It will not happen if the attacker makes two passes during the day. Those descriptors location of which changed during the first pass to already covered gaps will be collected during the second pass (since they can change the location once per 24 hours only).

Another important point is that consensus document remains valid for a client for 3 hours, starting from its publication. According to the current implementation the clients can download the new consensus in (FU + 45 mins;VA - 10 mins) interval. Hence a hidden service can skip the consensus document which immediately follows its current consensus. This means the hidden service directories of the

[7]Note that this countermeasure is not implemented in Tor, however.

[8]A gap is defined to be an interval in the circular list of fingerprints between two consecutive HS directories

attacker should be in at least two consecutive consensus documents in order for the hidden service to learn about them.

Taken into account the aforementioned the attacker would need to control $R = \frac{N}{12*2}$ IP addresses, where N is the number of hidden service directories in the Tor network.

Note that all the relays run by the attacker can be cheap since they do not have to provide high performance. Thus the attacker will have to pay only for the additional IP addresses and very little for the traffic. The IP addresses can be acquired from Amazon EC2 accounts. This results in a low-resource attack.

C. Experimental results

We performed the attack using 50 EC2 virtual instances. During the experiment we received 59130 publication requests for different descriptor IDs. We also fetched the descriptors from the memory of running Tor instances and obtained 58389 descriptors in total[9]. Out of them there were 24703 descriptors with unique public keys. The fraction of encrypted descriptors among them was approximately 1.5%.

When computing onion addresses from the descriptors, we found the botnet C&C addresses, DuckDuckGo's hidden service and the Silk Road onion address in that set – as expected. However, we also found what looked like backup or phishing onion addresses for Silk Road, namely onion addresses with the same 8 letter prefix:

```
silkroadrlzm5thj.onion
silkroadvb5piz3r.onion
silkroadvlsu5apk.onion
silkroad5hq52m36.onion
```

We were not able to connect to `silkroadrlzm5thj.onion`, however both `silkroadvlsu5apk.onion` and `silkroad5hq52m36.onion` redirected us to `silkroadvb5piz3r.onion` which is an onion-address for Silk Road that is publicly known.

In order to verify the completeness of the harvested data we collected a sample of 120 running hidden services from public sources. Our data set missed 4 relays from this sample set. By extrapolating this result we conclude that we could have lost about 3% of hidden descriptors.

We launched a second experiment on another date in order to reduce the costs of the attack. Because of the increased number of hidden services directories on that date, we used 58 EC2 instances. We also used an improved harvesting script: in addition to storing descriptors posted by hidden services we also initiated descriptors fetches from other responsible hidden services directories if a client's request was received for an unknown descriptor. At the end of the experiment we collected 39824 unique onion addresses.

[9]Note that we fetched the descriptors from memory 3 hours after the end of the experiment. This means that by that time some of our Tor relays removed a small portion of the descriptors from their memory

In order to reduce the experiments' costs we used the following. First both shadow and active relays had reported bandwidth of 0 Bytes/sec or 1 Bytes/sec. Since the granularity of the bandwidth values in the consensus is 1 kBytes/sec, all relays used in our attack were assigned bandwidth 0 kBytes/sec in the consensus. This means that the relays used in the attack should never be chosen by clients for purposes other than hidden services descriptors fetches. This has cut the traffic costs expenses. Secondly, we launched Tor relays participating in the harvesting from cheaper EC2 instances. In the second experiment, we used EC2 micro instances which is the cheapest option. In combination with reductions in traffic costs, this allowed us to reduce the overall price down to 57 USD.

Falling back to micro instances created performance problems however. Due to limited amount of RAM, at the end of the experiment, we could not establish SSH connections to some of EC2 instances and we had to reboot them to retrieve the data. The log files indicated that system clock jumped for several times which means that we could loose some hidden services descriptors.

This experiment had inadvertent but important side-effect on the flag calculation of Tor, of which we were notified by the Tor developers; see the Appendix for more details.

VI. Opportunistic deanonymisation of hidden services

The fact that an attacker always controls one side of the communication with a hidden service means that it is sufficient to sniff/control a guard of the hidden service in order to implement a traffic correlation attack and reveal the actual location of the hidden service. In particular, an attacker can:

- Given the onion address of a hidden service with unencrypted list of introduction points determine if her guard nodes are used by this hidden service.
- Determine the IP addresses of those hidden services that use the attacker's guard nodes.
- Determine if the attacker's guard nodes are used by any of the hidden services, even if the list of introduction points is encrypted.

A. Unencrypted descriptors

In order to confirm that an attacker controls a guard node of a hidden service she needs to control at least one more Tor non-Exit relay. In the attack, the hidden service is forced to establishes rendezvous circuits to the rendezvous point (RP) controlled by the attacker. Upon receiving a `RELAY_COMMAND_RENDEZVOUS1` cell with the attacker's cookie, the RP generates traffic with a special signature. This signature can be identified by the attacker's middle node. We note that a special `PADDING` cell mechanism in Tor simplifies generation of a signature traffic which is discarded at the recipient side, and is thus unnoticeable to the hidden

service. The steps of the attack are shown in Figure 5 and are as follows:

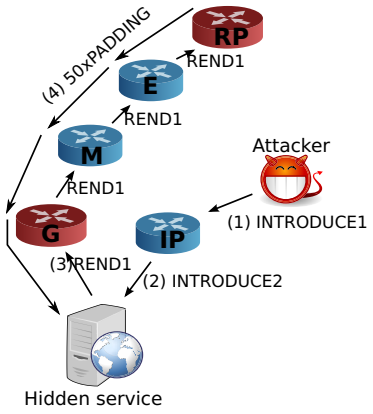

Figure 5. Revealing the guards

- The attacker sends a RELAY_COMMAND_INTRODUCE1 cell to one of the hidden service's introduction points (IP) indicating the address of the rendezvous point.
- The introduction point forwards the content in a RELAY_COMMAND_INTRODUCE2 cell to the hidden service.
- Upon receiving the RELAY_COMMAND_INTRODUCE2 cell, the hidden service establishes a three-hop circuit to the indicated rendezvous point and sends it a RELAY_COMMAND_RENDEZVOUS1 cell.
- when the rendezvous point controlled by the attacker receives the RELAY_COMMAND_RENDEZVOUS1 cell, it sends 50 PADDING cells back along the rendezvous circuit which are then silently dropped by the hidden service.
- the rendezvous point sends a DESTROY cell down the rendezvous circuit leading to the closure of the circuit.

Whenever the rendezvous point receives a RELAY_COMMAND_RENDEZVOUS1 with the same cookie as the attacker sent in the RELAY_COMMAND_INTRODUCTION1 cell it logs the reception. At the same time, the attacker's guard node monitors the circuits passing through it. Whenever it receives a DESTROY cell over a circuit it checks:

1) whether the cell was received just after the rendezvous point received the RELAY_COMMAND_RENDEZVOUS1 cell;
2) the number of the forwarded cells: 3 cells up the circuit and 53 cells down the circuit. Three cells more come from the fact that the hidden service established a circuit to the rendezvous point thus the attacker's guard node had to forward (2×RELAY_COMMAND_EXTEND + 1×RENDEZVOUS1) cells up and (2×RELAY_COMMAND_EXTENDED + 1×DESTROY) cells down. This is very important for our traffic signature since it allows us to distinguish the case when the attacker's node was chosen as the guard from the case when it was chosen as the middle.

If all the conditions are satisfied, the attacker decides that her guard node was chosen for the hidden service's rendezvous circuit and marks the previous node in the circuit as the origin of the hidden service.

In order to estimate the reliability of the traffic signature, we collected a statistics on the number of forwarded cells per circuit. We examined 748,846 circuits on our guard node. None of the circuit exhibited the traffic pattern of 3 cells up the circuit and 53 cells down the circuit. This means that the proposed traffic signature is highly reliable.

We implemented the approach to attack our own hidden service. We used a relay with a bandwidth of 500 Kbytes/s according to the consensus as the guard node and were scanning for the aforementioned traffic signature. For each RELAY_COMMAND_RENDEZVOUS1 cell receive events we collected the corresponding traffic pattern and got no false positives.

B. Encrypted descriptors

If the list of introduction points is encrypted, an attacker will not be able to establish a connection to the hidden service. Hence the attack described in the previous section does not apply. However, we can use a different method to determine if some of those encrypted hidden services use a guard node controlled by us. We will not be able distinguish between hidden services with encrypted introduction points though. On the other hand, note that results from Section V show that the number of hidden services which encrypt their introduction points is comparatively small.

To achieve this goal we do the following:

- On our guard node we look for a traffic pattern characteristic for introduction circuits (we describe this traffic pattern and how unique it is later in this section).
- We discard introduction circuits which originate at the same IP address as any of the hidden services with unencrypted descriptors.
- For all remaining introduction circuits, we mark their origins as possible locations of an encrypted hidden services.

Let us describe the characteristics exhibited by introduction circuits: The main difference between general-purpose circuits and introduction circuits is their duration. General Tor circuit stays alive either for ten minutes (if they were used by any stream), for one hour (if they did not carry any data traffic) or as long as any traffic is carried over them (this implies an open stream). In contrast, introduction circuits stay alive much longer, namely until some hop in the circuit fails or the hidden service closes the connection.

The second important difference is that after an introduction circuit is established, it does not transmit cells from the

origin. On the other hand, general-purpose circuits usually transmit traffic back and forth.

Thirdly, we can use the fact that introduction circuits are always multi-hop while some general-purpose circuits are one-hop.

In order to check how good these filters are, we launched a hidden service which established two introduction circuits through a non-guard relay controlled by us. By collecting the circuit statistics on this node for 24 hours we were able to identify our introduction circuits while having no false positives. We also did measurements on our guard node during 24 hours and identified 14 potential introduction circuits. However, we did not check if they belonged to hidden services with unencrypted introduction points.

C. Success rate and pricing for targeted deanonymizations

In early 2012 we operated a guard node that we rented from a large European hosting company (Server4You, product EcoServer Large X5) for EUR 45 (approx. USD 60) per month. Averaging over a month and taking the bandwidth weights into account we calculated that the probability for this node to be chosen as a guard node was approximately 0.6% on average for each try a Tor client made that month. As each hidden service chooses three guard nodes initially, we expect over 450 hidden services to have chosen this node as a guard node[10]. Running these numbers for a targeted (non-opportunistic) version of the attack described in Section VI-A shows us that by renting 23 servers of this same type would give us a chance of 13.8% for any of these servers to be chosen. This means that within 8 months, the probability to deanonymize a long-running hidden service by one of these servers becoming its guard node is more than 90%, for a cost of EUR 8280 (approximately USD 11,000).

Take into account that this scales well: Attacking multiple hidden services can be achieved for the same cost once the infrastructure is running.

VII. Revealing guard nodes of hidden services

As mentioned in the background section, each hidden service keeps a list of guard nodes. Revealing the guards does not immediately allow an attacker to reveal the location of the hidden service but gives her the next point of attack. This can be dangerous for a hidden service since it is supposed to be online for a long[11] time. This gives an attacker sufficient amount of time either to take control over the guard nodes or to start sniffing network traffic near the guards. Given that guard nodes are valid for more than a month, this may also be sufficient to mount a legal attack to recover traffic meta data for the guard node, depending on the jurisdiction the guard node is located in.

In this section we present an attack to reveal the guard nodes of a hidden service when the list of the introduction points in the HS descriptor is not encrypted (for the case when the list of introduction points in encrypted see Appendix B).

To do this, we use a technique similar to that presented in section VI; control over at least two Tor non-Exit relays is needed to carry it out. In the attack, the hidden service is forced to establishes many rendezvous connections to the rendezvous point (RP) controlled by the attacker in hope that some circuits pass through the second node (the middle node) controlled by the attacker. The RP generates traffic with a special signature which can be identified by the attacker's middle node. The steps of the attack are the same as in section VI.

Asymptotically, the probability that the attacker's middle node is chosen for the rendezvous circuit, approaches 1. Whenever the rendezvous point receives a `RELAY_COMMAND_RENDEZVOUS1` with the same cookie as the attacker sent in the `RELAY_COMMAND_INTRODUCTION1` cell it logs the reception and the IP address of the immediate transmitter of the cell. At the same time, the attacker's middle node monitors the circuits passing through it. Whenever it receives a `DESTROY` cell over a circuit it checks:

1) whether the cell was received just after the rendezvous point received the `RELAY_COMMAND_RENDEZVOUS1` cell;
2) if the next node of the circuit at the middle node coincides with the previous node of the circuit at the rendezvous point;
3) whether the number of forwarded cells is exactly 2 cells up the circuit and 52 cells down the circuit.

If all the conditions are satisfied, the attacker decides that her middle node was chosen for the hidden service's rendezvous circuit and marks the previous node in the circuit as a potential guard node of the hidden service.

We implemented the attack and ran it against two hidden services operated by us. In both cases the guard nodes were identified correctly, without any false positives. In the first case, the rendezvous point received around 36 000 `RELAY_COMMAND_RENDEZVOUS1` cells in 1 hour 20 minutes and the correct guard nodes were identified 8, 6, and 5 times correspondingly. In the seconds case, the rendezvous point received 16 000 `RELAY_COMMAND_RENDEZVOUS1` cells in 40 minutes and the correct guard nodes were identified 5, 2, and 1 times respectively.

We also used this approach to identify the guard nodes of the botnet hidden service. Note that in the attack described in this section an attacker can use just one middle node and send the traffic signature as a client. However it requires building rendezvous circuits which makes the attack longer. The same applies to the attack presented in section VI.

VIII. Discussion and potential countermeasures

We propose two countermeasures to make distributed storage of the hidden service descriptors more robust. The

[10] Assuming the current number of hidden services

[11] Silk Road's hidden service is already running for almost two years.

first of these prevents the directory authorities from learning the contents of hidden services descriptors they are serving. This prevents hidden services from harvesting descriptors to learn more onion addresses. Our second proposed change makes the position of the responsible hidden service directories in the directory fingerprint ring unpredictable for any hidden service. This removes the opportunity of targeting hidden service directories. Henceforth attackers can no longer precompute identity keys to target hidden services for popularity measurements and to deny service to them by selectively running relays with those keys.

Harvesting can be easily prevented by making the `descriptor-cookie` authentication [15] mandatory for all hidden services and base32 encoding the value as part of the URL together with the `permanent-id`. The downside of this change is a significantly reduced usability: instead of 16 character onion addresses the user now has to deal with onion-addresses that are 42 characters long.

In order to prevent adversaries from efficiently targeting hidden service directories we propose the following changes: For each hour, an unpredictable value is derived by the directory authorities from a shared secret. Three of these values are included in the consensus – one for each of the hours the consensus is valid.

The unpredictable value valid for the hour of the request is then included in the calculation of the descriptor ID and henceforth determines the place on the ring where the descriptor is stored. This makes it impossible for an attacker to precompute identity keys for time periods further ahead than 3 hours in the future.

Additionally, directory authorities base the decision on whether a relay is assigned an HSDir flag on the number of past consecutive consensus documents the relay has been listed in and not on the uptime of the relay. This prevents the shadowing attack we have described.

To prevent the guard nodes being revealed, one can use an additional layer of guard nodes – guard middle nodes. This countermeasure has already been proposed in [19] but is not implemented in Tor. Note that this measure will not protect against an attacker exploiting degree anomalies of the guard nodes as described in Section B.

Unfortunately, we do not see how the risk of guard nodes being able to deanonymize a hidden service having chosen them can be eliminated completely. Recent work by Tariq et al. [9] suggests that the guards compromise rate can be decreased by (1) making the guard rotation interval longer and (2) by taking into account how long nodes have been part of the network when assigning Guard flags to them. Note that this approach if not carefully implemented has a number of downsides like reduced end-user quality of experience and malicious nodes accumulating Tor users.

In regard to revealing the introduction circuits, if the attacker will not be able to collect the full list of hidden service descriptors, she will not be able to distinguish between introduction circuit of hidden services with encrypted introduction points and non-encrypted.

IX. Conclusion

We have analyzed the security properties of Tor hidden services and shown that attacks to deanonymize hidden services at a large scale are practically possible with only a moderate amount of resources. We have demonstrated that collecting the descriptors of all Tor hidden services is possible in approximately 2 days by spending less than USD 100 in Amazon EC2 resources. Running one or more guard nodes then allows an attacker to correlate hidden services to IP addresses using a primitive traffic analysis attack. Furthermore, we have shown that attackers can impact the availability and sample the popularity of arbitrary hidden services not under their control by selectively becoming their hidden service directories.

To address these vulnerabilities we have proposed countermeasures. These prevent hidden service directories from learning the content of any the descriptors unless they also know their corresponding onion address and significantly increase the resources required to selectively become a hidden service directory for a targeted hidden service.

However, note that the above suggestions are nothing more than stop-gap measures. We believe that the problems we have shown are grave enough to warrant a careful redesign of Tor's hidden services.

References

[1] Anonymous. IAmA a malware coder and botnet operator, AMA. http://www.reddit.com/r/IAmA/comments/sq7cy/iama_a_malware_coder_and_botnet_operator_ama/, April 2012.

[2] Bauer, K. S., McCoy, D., Grunwald, D., Kohno, T., and Sicker, D. C. Low-resource routing attacks against Tor. In *WPES* (2007), P. Ning and T. Yu, Eds., ACM, pp. 11–20.

[3] Biryukov, A., Pustogarov, I., and Weinmann, R.-P. TorScan: Tracing long-lived connections and differential scanning attacks. In *ESORICS 2012*, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459 of *LNCS*. Springer, 2012, pp. 469–486.

[4] Brown, D. Resilient botnet command and control with Tor. DefCon 18, https://www.defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-18-Brown-TorCnC.pdf, July 2010.

[5] Christin, N. Traveling the silk road: A measurement analysis of a large anonymous online marketplace. *CoRR abs/1207.7139* (2012).

[6] Darroch, J. N., and Seneta, E. On quasi-stationary distributions in absorbing continuous-time finite Markov chains. *Journal of Applied Probability 4*, 1 (1967), pp. 192–196.

[7] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. F. Tor: The second-generation onion router. In *USENIX Security Symposium* (2004), USENIX, pp. 303–320.

[8] DOUCEUR, J. R. The Sybil attack. In *IPTPS 2002 – Peer-to-Peer Systems, First International Worksho* (2002), P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds., vol. 2429 of *Lecture Notes in Computer Science*, Springer, pp. 251–260.

[9] ELAHI, T., BAUER, K., ALSABAH, M., DINGLEDINE, R., AND GOLDBERG, I. Changing of the guards: A framework for understanding and improving entry guard selection in Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2012)* (2012), ACM, pp. 43–54.

[10] ELICES, J., PEREZ-GONZALEZ, F., AND TRONCOSO, C. Fingerprinting Tor's Hidden Service Log Files Using a Timing Channel. In *WIFS 2011 – 3rd IEEE International Workshop on Information Forensics and Security* (2011), IEEE.

[11] G DATA SECURITY. Botnet command server hidden in Tor. http://blog.gdatasoftware.com/blog/article/botnet-command-server-hidden-in-tor.html, September 2012.

[12] GUARNIERI, C. Skynet, a Tor-powered botnet straight from Reddit. https://community.rapid7.com/community/infosec/blog/2012/12/06/skynet-a-tor-powered-botnet-straight-from-reddit, December 2012.

[13] JANSEN, R., AND HOPPER, N. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *NDSS 2012 – Proceedings of the Network and Distributed System Security Symposium* (2012), Internet Society.

[14] LIPMAN, D. H. Mailing list post: "[tor-talk] vwfws4obovm2cydl.onion ?". https://lists.torproject.org/pipermail/tor-talk/2012-June/024565.html, June 2012.

[15] LOESING, K. *Privacy-enhancing Technologies for Private Services*. PhD thesis, University of Bamberg, May 2009.

[16] MANDL, P. Sur le comportement asymptotique des probabilites dans les ensembles de etats dune chaine de Markov homogeene (Russian). *Casopis Pest. Mat. 84* (1959), pp. 140–149.

[17] MICAH, M. W., WRIGHT, M., ADLER, M., AND LEVINE, B. N. Defending anonymous communications against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (2003), pp. 28–41.

[18] MURDOCH, S. J. Hot or not: Revealing hidden services by their clock skew. In *13th ACM Conference on Computer and Communications Security* (2006), ACM Press, pp. 27–36.

[19] ØVERLIER, L., AND SYVERSON, P. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy* (2006), IEEE Computer Society, pp. 100–114.

[20] ØVERLIER, L., AND SYVERSON, P. F. Valet services: Improving hidden servers with a personal touch. In *PET 2006 – Privacy Enhancing Technologies* (2006), G. Danezis and P. Golle, Eds., vol. 4258 of *Lecture Notes in Computer Science*, Springer, pp. 223–244.

[21] POLLETT, P. K. Analytical and computational methods for modelling the long-term behaviour of evanescent random processes, June 1993.

[22] SCHUMER, C. E., AND MANCHIN, J. Press release: Manchin urges federal law enforcement to shut down online black market for illegal drugs. http://manchin.senate.gov/public/index.cfm/2011/6/manchin-urges-federal-law-enforcement-to-shut-down-online-black-market-for-illegal-drugs, Jun 2011.

[23] SENETA, E. Quasi-stationary behaviour in the random walk with continuous time. *Australian Journal of Statistics 8*, 2 (1966), 92–98.

[24] SHEBARO, B., PEREZ-GONZALEZ, F., AND CRANDALL, J. R. Leaving timing-channel fingerprints in hidden service log files. *Digital Investigation 7* (2010), pp. 104–113.

[25] THE TOR PROJECT. Tor Rendezvous Specification. https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=rend-spec.txt, accessed on November 13th, 2012.

[26] ZANDER, S., AND MURDOCH, S. J. An improved clock-skew measurement technique for revealing hidden services. In *USENIX Security Symposium* (2008), pp. 211–226.

APPENDIX

A. The Influence of Shadow Relays on the Flag Assignment

During the second harvesting experiment we accidentally revealed an important artifact of the flag assignment in Tor which is not obvious from the Tor specifications. Near the end of the experiment we were notified by the Tor developers that the Sybil attack had caused a spike in the number of relays assigned Fast flags and Guard flags (see Fig. 6)

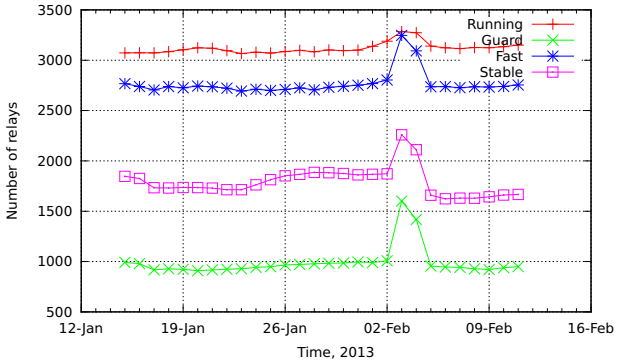

Figure 6. Increase in the number of Guard nodes.

This happened because the shadow relays were taken into account for calculating medians of the bandwidth and the uptime. From these values, thresholds are derived that determine the flag assignment of all relays. According to the Tor specification:

> ... A router is a possible Guard if its Weighted Fractional Uptime is at least the median for familiar active routers, and if its bandwidth is at least median or at least 250KB/s.
>
> To calculate weighted fractional uptime, compute the fraction of time that the router is up in any given day, weighting so that downtime and uptime in the past counts less.
>
> A node is familiar if 1/8 of all active nodes have appeared more recently than it...

In our second experiment, we caused the authorities to take into account 1392 shadow relays of bandwidth 0 Bytes/sec and 1 Bytes/sec. This significantly changed the medians for both bandwidth and uptime, which allowed many already running relays to get the Guard flag. During the harvesting experiment, this caused the number of Guard nodes to suddenly increase by 500.

The artifact has been patched in tor v0.2.4.10-alpha by ignoring Sybil relays when assigning flags. However, it is important to note that a more expensive version of the same Sybil attack is still possible. For example, an attacker could rent a large number of EC2 instances, running 2 Tor relays on each. This would enable attackers' Tor relays to decrease the value of the median for the Weighted Fractional Uptime as well as the bandwidth median, allowing to obtain the Guard flag for her relays much faster. For example, in order to inject 1200 Tor relays, an attacker would need to run 600 EC2 instances, spending only 288 USD per 24 hours.

B. Anomalies in the Tor Network Topology

If the list of introduction points of a hidden services is encrypted, it is not possible to make the hidden service establish rendezvous circuits (as was described in Section VII). In order to reveal the guard nodes of a popular hidden service in this case, an attacker can use another attack. The condition for this attack is that the hidden service has many clients which establish long-lived (approx. 1-2 hours or longer) connections. This is the case of the botnet described in previous sections; connections to its IRC hidden service are long-lived.

Our measurements show that currently only a small fraction of all hidden services use encrypted descriptors. However we believe that this is an important case to study since encrypted descriptors offer significant additional protection and in the original draft of the Tor hidden services protocol all descriptors were supposed to be hidden.

Popularity of a hidden service, i.e. a large number of clients connecting to it, creates additional load on its guards nodes. This changes the topological properties of the guard nodes in terms of their degree [12] and in terms of the decay rate of persistent connections (in comparison to the case when the guard nodes are not used by a popular hidden service). In particular:

- the degree of the guard nodes of such a service will depend on the number of clients. The deviation of a node's degree from the expected value can serve as an indication of a popular hidden service;
- if clients make persistent connections to the hidden service (which is the case with botnet where IRC channel is used) the decay rate of the persistent connections of the HS's guard nodes will look substantially different from that of other guard nodes with similar bandwidth. We expect that the decay curve is much steeper in the case of normal guard nodes.

In order to identify the guard nodes of a popular hidden service we implement the following steps. We provide two analytical models for (1) the expected degree and (2) the expected persistent connections decay rate of a "normal" guard node. We then use the scanning technique from [3] to determine the real degrees of the guard nodes and their persistent connections decay rates. Finally, we compare the predicted values with those received from the measurements and single out nodes with too high degrees and too slow decay rates. The nodes we get are the candidates for the guard nodes of the hidden service.

By *persistent connections decay rate* of a Tor relay we mean the following: Assume that at time t_0 the relay has N TLS connections with other Tor relays. The decay rate of these connections is a function of time which shows how many of them remain connected at time t.

1) Expected degree of nodes in the Tor network graph: In order to derive the expected degree of a Tor relay we use results presented in [3], section 5.2. In [3], the probability of a TLS connection between two Tor relays at a given point in time is computed as ratio of the average gap between connections to the average connection duration. We denote t_{avg} as the average circuit duration and t_{idle} as the lifetime of a connection without circuits. According to empirical results presented [3], the average duration of a circuit is 200 seconds and according to the current Tor implementation, the lifetime of a connection without circuits is set to three minutes.

Assuming a delay larger than $t_{avg} + t_{idle}$, the average delay between two circuits is computed as:

$$I = \frac{\int_{t_{avg}+t_{idle}}^{\infty} t \cdot \lambda_{a,b} \cdot e^{-\lambda_{a,b} \cdot t} \mathrm{d}t}{e^{\lambda_{a,b} \cdot (t_{avg}+t_{idle})}} = t_{avg} + t_{idle} + \frac{1}{\lambda_{a,b}}, \quad (1)$$

The probability that there is a connection between A and B at an arbitrary point in time is given by:

[12]The *degree* of a Tor relay denotes the number of TLS connections established between a given relay and other relays.

$$P_{AB} = 1 - e^{-\lambda_{a,b}\cdot(t_{avg}+t_{idle})} \cdot \frac{I-(t_{avg}+t_{idle})}{I} = 1 - \frac{e^{-\lambda_{a,b}\cdot(t_{avg}+t_{idle})}}{\lambda_{a,b}(t_{avg}+t_{idle})+1}, \quad (2)$$

where R is the current circuit arrival rate of the whole Tor network, and $p_{a,b}$ is the probability of routers A and B to form an edge in a circuit and $\lambda_{a,b} = R \cdot p_{a,b}$. The probability $p_{a,b}$ is approximated as follows:

$$p_{a,b} = 2 \cdot \frac{bw_a bw_b}{bw_{total}} \left(\frac{1}{bw_{guards}} + \frac{1}{bw_{exit}} \right),$$

where bw_{guards} is the total bandwidth of guard nodes, bw_{exit} is the total bandwidth of exit nodes, bw_{total} is the total bandwidth of the whole Tor network, bw_a and bw_b are bandwidths of routers A and B respectively. This information is obtained from the consensus document.

The expected number of open connections of a Tor relay at an arbitrary point of time is thus:

$$N_A^{avg} = \lceil \sum_{B \in T} P_{AB} \rceil,$$

where T denotes the set of all Tor relays and $|T| = n$.

We now compare the model with the observed degrees of Tor relays. We used the technique described in [3] in order determine to which other relays a given Tor relay has established TLS connections. Figure 7 shows the degrees of Tor relays sorted by their bandwidth weight from the consensus.

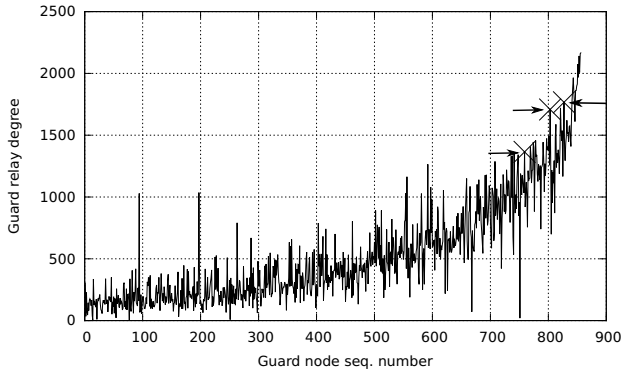

Figure 7. Degrees of Tor relays

From this figure, one can see that there are a number of nodes which deviate significantly from the average – we call these *peak* nodes. The guard nodes of the botnet which we determined in the previous section are marked by arrows and are among the peaks. This allows us to filter out quite many relays. However the number of peaks is still considerable. In the next section we show how to reduce the set of candidates of guard nodes of a popular hidden service based on the persistent connection decay rate.

2) Decay rate of persistent connections: As mentioned in [3], for an average Tor relay the decay rate of persistent connections is steep during the first hours. This is not the case if a relay is a guard node of a hidden service with persistently connected clients, such as the botnet's IRC command and control. In this case the decay rate will be determined by the bots going offline rather than by the bandwidth of the node.

In order to predict the decay rate of a "normal" Tor relay we use the following approach: We first find the expression for the duration of a connection between relay A and B and use it to determine the connection decay rate. We assume the following: 1) circuits arrive to the connections according to Poisson distribution [3]; 2) the circuit arrival rate is proportional to the bandwidth of the relay; 3) the circuit duration follows an exponential distribution. Given these assumptions, we adopt a finite state Markov chain to model the connection duration. Each state of the Markov chain represents the number of circuits carried over the connection. The chain has one absorbing state 0. We are interested in the extinction time. The number of states is finite.

We assume that at the time when we observe the connections, the system is in quasistationary state, conditioned that the extinction has not occurred. Thus the initial state distribution is a quasi-stationary distribution which always exists for finite state case (see [16] , [6]). Classical matrix theory can be used to show that a matrix containing infinitesimal transition probabilities of transient states has a dominant eigenvalue such that the corresponding left and right eigenvectors have positive entries (see [16], and [6]); the left eigenvector is the quasistationary distribution. We denote $(q_1, q_2, ..., q_N)$ as the row vector of quasistationary probabilities.

Let λ be the circuit arrival rate to a connection between two Tor relays and μ the circuit closing rate. In this case, the matrix of infinitesimal transition probabilities is:

$$\mathbf{R} = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{a} & \mathbf{C} \end{bmatrix}, \quad (3)$$

where the matrix $\mathbf{C}$ corresponds to transient states $T = \{1, 2, ..N\}$ and state 0 is absorbing. The matrix $\mathbf{C}$ can be written as:

$$\begin{pmatrix} -\lambda-\mu & \lambda & 0 & 0 & \cdots & 0 & 0 \\ \mu & -\lambda-\mu & \lambda & 0 & \cdots & 0 & 0 \\ 0 & \mu & -\lambda-\mu & \lambda & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \mu & -\mu \end{pmatrix} \quad (4)$$

The probability of extinction of a connection between relays A and B in this case can then be derived as (we use Kolmogorov forward equations to get this result):

$$p_0^{AB}(t) = 1 - e^{-\mu q_1 t}$$

The circuit arrival rate $\lambda_{a,b}$ is computed as in the previous subsection. We set the circuit closing rate as $\mu = 1/(t_{avg} + t_{idle})$, where t_{avg} is the average duration of a circuit as in [3] and $t_{idle} = 180$ seconds is the time before an idle connection would close. As stated in [3], t_{avg} depends only slightly on the pair of relays and is close to 200 seconds.

One can use numerical methods (see [21] for example) to compute the eigenvalues and eigenvectors. Note that for the cases when $\lambda_{a,b} < \mu$, one can approximate the values with an expression for infinite state Markov chain [23]. In the infinite case, the quasistationary probability for the system to be in state j is:

$$q_j = (1-\beta)^2 j \beta^{j-1},$$

where $\beta = \sqrt{\frac{\lambda}{\mu}}$. Particularly,

$$q_1 = (1-\beta)^2.$$

We apply this model to the pair of medium-bandwidth Tor relays for which the experimental data was presented in [3]. The consensus bandwidths of the relays were 1850 kBytes/sec and 4280 kBytes/sec. Both were Guard and non-Exit nodes. The comparison between the model and the data obtained from the direct measurements on one of the nodes is shown in Figure 8.

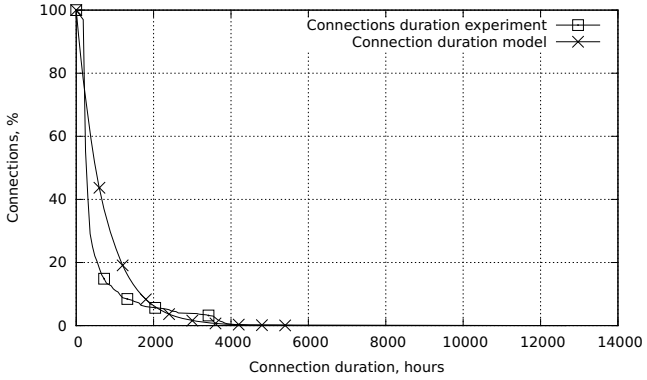

Figure 8. Connection duration model validation

Given the initial connections of a relay, we use the model for connection duration to compute the expected number of persistent connections at an arbitrary point of time for Tor relay A:

$$N_A^{pers}(t) = \sum_{B \in I} (1 - p_0^{AB}(t)),$$

where I is the set of initial connections of A.

Using the technique from [3] we obtained the persistent connections decay rate of the Tor guard nodes. We compared them with the connection decay rate predicted by the model and filtered those connections which differ from the model. Particularly, we compared the number of persistent connections after 3 hours of scanning. Out of 856 nodes 200 had a degree that exceeded the value predicted by the model. Choosing a threshold such that guard of the botnet's hidden service is included, we find that 37 nodes have a degree that is 1.4 time higher than the value predicted by the model.

Figure 9 shows the real decay rate of the botnet's guard nodes plotted against the theoretically predicted one. As one can see, the discrepancy is quite detectable.

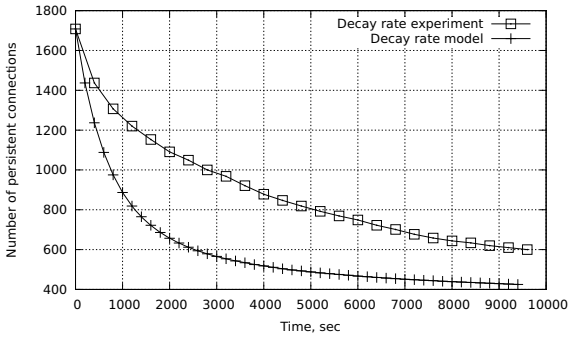

Figure 9. Decay rate of the botnet's guard 1

Figure 10 shows the observed decay rate and the predicted decay rate of another node with a degree above the average (one of the peaks in Figure 7). The majority of peaks from the previous section have this type of the decay rate, which is close to the theoretical predictions. This allows us to reduce the number of candidates to 29. Since we know the actual guard nodes of the botnet's hidden service from the unencrypted descriptor attack, we were able to check that indeed the correct guards appeared in this list of candidates.

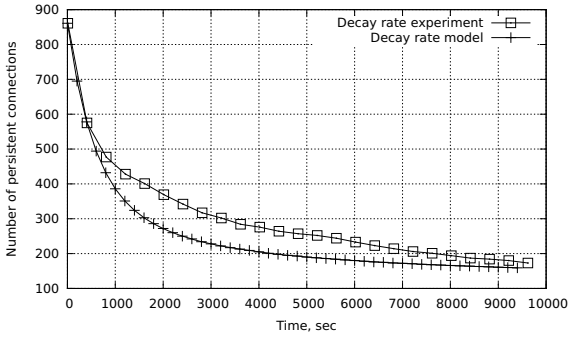

Figure 10. Common shape of the decay rate