Spoiled Onions: Exposing Malicious Tor Exit Relays[*]

Philipp Winter[1], Richard Köwer[3], Martin Mulazzani[2], Markus Huber[2], Sebastian Schrittwieser[2], Stefan Lindskog[1], and Edgar Weippl[2]

[1] Karlstad University, Sweden
[2] SBA Research, Austria
[3] FH Campus Wien, Austria

Abstract. Tor exit relays are operated by volunteers and together push more than 1 GiB/s of network traffic. By design, these volunteers are able to inspect and modify the anonymized network traffic. In this paper, we seek to expose such malicious exit relays and document their actions. First, we monitored the Tor network after developing two fast and modular exit relay scanners—one for credential sniffing and one for active MitM attacks. We implemented several scanning modules for detecting common attacks and used them to probe all exit relays over a period of several months. We discovered numerous malicious exit relays engaging in a multitude of different attacks. To reduce the attack surface users are exposed to, we patched Torbutton, an existing browser extension and part of the Tor Browser Bundle, to fetch and compare suspicious X.509 certificates over independent Tor circuits. Our work makes it possible to continuously and systematically monitor Tor exit relays. We are able to detect and thwart many man-in-the-middle attacks, thereby making the network safer for its users. All our source code is available under a free license.

1 Introduction

As of January 2014, nearly 1,000 exit relays [30] distributed all around the globe serve as part of the Tor anonymity network [10]. As illustrated in Fig. 1, the purpose of these relays is to establish a bridge between the Tor network and the "open" Internet. A user's Tor circuits—which are basically encrypted tunnels—are terminated at exit relays and from there, the user's traffic proceeds to travel over the open Internet to its final destination. Since exit relays can see traffic as it is sent by clients, Tor users are advised to use end-to-end encryption. By design, exit relays act as a "man-in-the-middle" (MitM) in between a user and her destination. This renders it possible for exit relay operators to run various MitM attacks such as traffic sniffing, DNS poisoning, and SSL-based attacks

[*]This work is the result of merging two PETS submissions. The original titles and authors were: "Spoiled Onions: Exposing Malicious Tor Exit Relays" by Winter and Lindskog, and "HoneyConnector: Active Sniffer Baiting on Tor" by Köwer, Mulazzani, Huber, Schrittwieser, and Weippl.

such as HTTPS MitM and sslstrip [22]. An additional benefit for attackers is that exit relays can be set up quickly and anonymously, thus making it very difficult to trace attacks back to their origin. While it is possible for relay operators to specify contact information such as an e-mail address,[4] this is optional and as of January 2014, only 56% out of all 4,962 relays publish contact information. Even fewer relays publish *valid* contact information.

To thwart a number of popular attacks, TorBrowser [26]—the Tor Project's modified version of Firefox—ships with the two extensions HTTPS-Everywhere [11] and NoScript [17]. While the former contains rules to rewrite HTTP to HTTPS traffic, NoScript seeks to prevent many script-based attacks. However, there is little clients can do in the face of web sites implementing poor security such as the lack of site-wide TLS, session cookies being sent in the clear, or using weak cipher suites in their web server configuration. Often, such bad practice enables attackers to spy on users' traffic or, even worse, hijack accounts. Besides, TorBrowser cannot protect against attacks targeting non-HTTP(S) protocols such as SSH. All these attacks are not just of theoretical nature. In 2007, a security researcher published 100 POP3 credentials he captured by sniffing traffic on a set of exit relays under his control [25]; supposedly to show the need for end-to-end encryption when using Tor. Section 2.1 discusses additional attacks which were found in the wild.

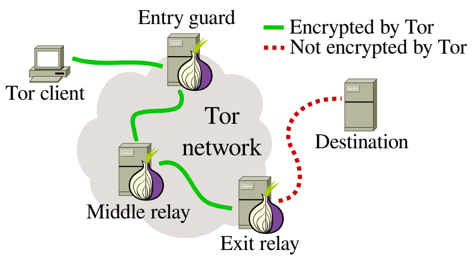


Fig. 1. The structure of a three-hop Tor circuit. Exit relays constitute the bridge between encrypted circuits and the open Internet. As a result, exit relay operators can see—and tamper with—anonymized traffic of users.

The main contributions of this paper are:

– We discuss the design and implementation of exitmap, a flexible and fast exit relay scanner which is able to detect several popular MitM attacks.
– We introduce HoneyConnector, a framework to detect sniffing Tor exit relays based on FTP and IMAP bait connections.
– Using exitmap and HoneyConnector, we monitored the Tor network over a period of multiple months in two independent studies. In total, we identified 65 exit relays that conducted MitM attacks or reused sniffed credentials.
– To detect MitM attacks against HTTPS, we propose the design and prototype of a patch for the Torbutton browser extension which fetches and compares X.509 certificates over diverging Tor circuits.

[4]Contact information is useful to get in touch with relay operators, e.g., if their relay is not configured correctly.

The remainder of this paper is structured as follows: Section 2 gives a brief background on how misbehaving relays are handled in the Tor network and gives an overview of related work. Section 3 discusses the design and implementation of `exitmap` and `HoneyConnector`, our scanners to detect malicious relays. We ran both frameworks for multiple months consecutively and present the attacks we discovered in Section 4 and discuss them in Section 5. Section 6 presents countermeasures to protect against HTTPS MitM attacks. Finally, Section 7 concludes this paper.

2 Background

The Tor Project has a way to prevent clients from selecting bad exit relays as the last hop in their three-hop circuits. After a suspected relay is communicated to the project, the reported attack is first reproduced. If the attack can be verified, a subset of two (out of all nine) directory authority operators manually blacklist the relay using Tor's `AuthDirBadExit` configuration option. Every hour, the directory authorities vote on the *network consensus* which is a signed list of all relays, the network is comprised of. Among other information, the consensus includes the *BadExit flag*. As long as the majority of the authorities responsible for the BadExit flag—i.e., two out of two—agree on the flag being set for a particular relay, the next network consensus will label the respective relay as BadExit. After the consensus was signed by a sufficient number of directory authorities, it propagates and is eventually used by all Tor clients after 24 hours have passed. From then on, clients will no longer select relays labeled as BadExit as the last hop in their circuits. Note that this does not mean that BadExit relays become effectively useless. They keep getting selected by clients as their entry guards and middle relays. Most of the malicious relays we discovered were assigned the BadExit flag after we reported them to the Tor Project. The relays which escaped the BadExit flag were either merely misconfigured or already offline when we reported them to the Tor Project.

Note that the BadExit flag is not only given to relays which are believed to be malicious. It is also assigned to relays which are misconfigured or are otherwise unable to fulfill their duty of providing unfiltered Internet access. A frequent cause of misconfiguration is the use of third-party DNS resolvers which block certain web site categories such as "pornography" or "proxy/anonymizer". Apart from the BadExit flag, directory authorities can blacklist relays by disabling its *Valid* flag which prevents clients from selecting the relay for *any* hop in its circuit. This option can be useful to disable relays running a broken version of Tor or are suspected to engage in end-to-end correlation attacks.

2.1 Related Work

In 2006, Perry began developing the framework "Snakes on a Tor" (`SoaT`) [31]. `SoaT` is a Tor network scanner whose purpose is to detect misbehaving exit relays. Similar to the less advanced `torscanner` [35], decoy content is first fetched over Tor, then over a direct Internet connection, and finally compared. Over time, SoaT was extended with support for HTTP, HTTPS, SSH and several other protocols. However, SoaT is no longer maintained and makes use of deprecated libraries. Compared to SoaT, exitmap is more flexible and significantly faster. Similar to SoaT, Marlinspike implemented tortunnel [23] which exposes a local SOCKS interface. Incoming data is then sent over exit relays using one-hop circuits. By default, exitmap does not use one-hop circuits as that could be detected by attackers which could then act honestly.

A first academic attempt to detect malicious exit relays was made in 2008 by McCoy et al. [24]. The authors established decoy connections to servers under their control. They further controlled the authoritative DNS server responsible for the decoy hosts' IP addresses. As long as a malicious exit relay sniffed network traffic with reverse DNS lookups being enabled, the authors were able to map reverse lookups to exit relays by monitoring the authoritative DNS server's traffic. By exploiting that side channel, McCoy et al. were able to find one exit relay sniffing POP3 traffic at port 110. However, attackers can easily avoid that side channel by disabling reverse lookups. The popular tool tcpdump implements the command line switch `-n` for that exact purpose. In 2011, Chakravarty et al. [5] attempted to detect sniffing exit relays by systematically transmitting decoy credentials over all active exit relays. Over a period of ten months, the authors uncovered ten relays engaging in traffic snooping. Chakravarty et al. could verify that the operators were sniffing exit traffic because they were later found to have logged in using the snooped credentials. While the work of Chakravarty et al. represents an important first step towards monitoring the Tor network, their technique only focused on SMTP and IMAP. At the time of our writing, only 20 out of all ∼1,000 exit relays allow connections to port 25. Instead, HoneyConnector focuses on FTP and IMAP. Also, similar to McCoy et al., the authors only discussed traffic snooping attacks which are passive. Active attacks remain entirely unexplored until today.

The Tor Project used to maintain a web page documenting misbehaving relays which were assigned the BadExit flag [18]. As of January 2014, this page lists 35 exit relays which were discovered in between April 2010 and July 2013. Note that not all of these relays engaged in attacks; almost half of them ran misconfigured anti virus scanners or used broken exit policies.[5] Since Chakravarty et al., no systematic study to spot malicious exit relays was conducted. Only some isolated anecdotal evidence emerged [34]. Our work is the first to give a comprehensive overview of *active attacks*. We further publish our code under a free license.[6] By doing so, we enable and encourage continuous and *crowdsourced* measurements rather than one-time scans.

[5] An exit relay's *exit policy* determines to which addresses and ports the relay forwards traffic to. Often, relay operators choose to not forward traffic to well-known file sharing ports in order to avoid copyright infringement.

[6] The code is available at http://www.cs.kau.se/philwint/spoiled_onions.

3 Monitoring Tor Exit Relays

We now discuss the design and implementation of exitmap as well as HoneyConnector which are both lightweight Python-based *exit relay scanners*. Their purpose is to systematically create circuits to exit relays which are then probed by modules which establish decoy connections to various destinations. While exitmap focuses on active attacks, HoneyConnector seeks to uncover traffic snooping. We aim to provoke exit relays to tamper with or snoop on our connections, thereby revealing their malicious intent. By doing so, we seek to discover and remove all “spoiled onions” in the Tor network. Our adversary model is thus a relay operator who exploits the fact that traffic can be modified or might be unencrypted once it leaves the Tor network. We will also show that our scanners' *modular design* enables quick prototyping of new scanning modules and exitmap's *event-driven architecture* makes it possible to scan all exit relays within a matter of only seconds while at the same time sparing their resources.

3.1 The Design of exitmap

exitmap is an active scanner that is designed to detect MitM attacks of various kinds. The schematic design of exitmap is illustrated in Fig. 2. Our tool is run on a single machine and requires the Python library Stem [32]. Stem implements the Tor control protocol [33] and we use it to initiate and close circuits, attach streams to circuits as well as to parse the network consensus. Upon starting exitmap, it first invokes a local Tor process which proceeds by fetching the newest network consensus in order to know which exit relays are currently online.

Next, our tool is fed with a set of exit relays. This set can consist of a single relay, all exit relays in a given country, or the set of all Tor exit relays. Random permutation is then performed on the set so that repeated scans do not probe exit relays in the same order. This is useful while developing and debugging new scanning modules as it equally distributes the load over all selected exit relays. Once exitmap knows which exit relays it has to probe, it initiates circuits which use the respective exit relays as their last hop. All circuits are created asynchronously in the background. Once a circuit to an exit relay is established, Tor informs exitmap about the circuit by sending an asynchronous circuit event over

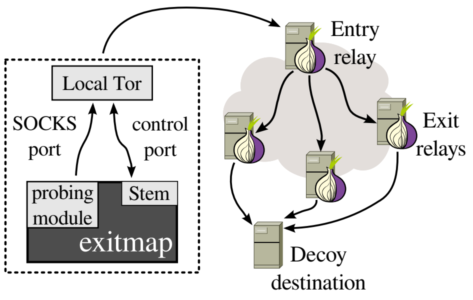


Fig. 2. The design of exitmap. Our scanner invokes a Tor process and uses the library Stem to control it. Using Stem, circuits are created “manually” and attached to decoy connections which are initiated by our probing modules.

the control connection. Upon receiving the event, `exitmap` invokes the desired probing module which then proceeds by establishing a connection to a decoy destination (see Section 3.1). Tor creates stream events for new connections to the SOCKS port which are also sent to `exitmap`. When a stream event is received, we attach the stream of a probing module to the respective circuit. Note that stream-to-circuit attaching is typically done by Tor. In order to have control over this process, our scanner invokes Tor with the configuration option `__LeaveStreamsUnattached` which instructs Tor to leave streams unattached. For performance reasons, Tor builds circuits preemptively, i.e., a number of circuits are kept ready even if there is no data to be sent yet. Since we want full control over all circuits, we prevent Tor from creating circuits preemptively by using the configuration option `__DisablePredictedCircuits`. `exitmap`'s probing modules can either be standalone processes or Python modules. Processes are invoked using the `torsocks` wrapper [36] which hijacks system calls such as `socket()` and `connect()` in order to redirect them to Tor's SOCKS port. We used standalone processes for our HTTPS and SSH modules. In addition, probing modules can be implemented in Python. To redirect Python's networking API over Tor's SOCKS port, we extended the SocksiPy module [13]. We used Python for our sslstrip, DNS, XMPP, and IMAPS modules.

Performance Hacks A naive approach to probing exit relays could be a nontrivial burden to the Tor network; mostly computationally but also in terms of network throughput. We implemented a number of tweaks in order for our scanning to be as fast and cheap as possible.

First, we expose a configuration option for avoiding the default of three-hop circuits. Instead, we only use *two hops* as illustrated in Fig. 3. Tor's motivation for three hops is anonymity but since our scanner has no need for strong anonymity, we only select a static entry relay—ideally operated by `exitmap`'s user—which then directly forwards all traffic to the respective exit relays. We offer no option to use one-hop circuits as that would make it possible for exit relays to isolate scanning connections: A malicious exit relay could decide not to tamper with a circuit if it originates from a non-Tor machine. Since we use a static first hop which is operated by us, we concentrate most of the scanning load on a single machine which is well-suited to deal with the load. Other

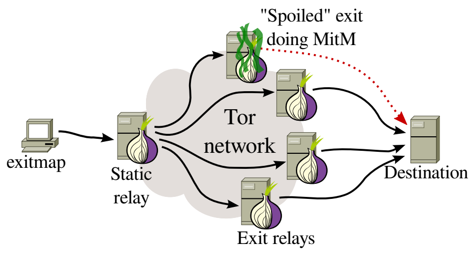


Fig. 3. Instead of establishing a full three-hop circuit, our scanner is able to use a static middle relay; preferably operated by whoever is running our scanner. By doing so, we concentrate the load on one machine while making our scanning activity slightly less stealthy.

entry and middle relays do not have to "suffer" from exitmap scans.

However, note that over time malicious exit relays are able to correlate scans with relays, thus determining which relays are used for scans. To avoid this problem, exitmap's first hop should be changed periodically and we hope that by crowdsourcing our scanner, isolating middle relays is no longer a viable option for attackers. Another computational performance tweak can be achieved on Tor's authentication layer. At the moment, there are two ways how a circuit handshake can be conducted; either by using the *traditional TAP* or the *newer NTor* handshake. TAP—short for Tor Authentication Protocol [12]—is based on Diffie-Hellman key agreement in a multiplicative group. NTor, on the other hand, uses the more efficient elliptic curve group Curve25519 [2]. A non-trivial fraction of a relay's computational load can be traced back to computationally expensive circuit handshakes. By favoring NTor over TAP, we slightly reduce the computational load on exit relays. As NTor supersedes TAP and is becoming more and more popular as Tor clients upgrade, we believe that it is not viable for attackers to "whitelist" NTor connections.

Scanning Modules After discussing exitmap's architecture, we now present several probing modules we developed in order to be able to detect specific attacks. When designing a module, it is important to consider its *indistinguishability* from genuine Tor clients. As mentioned above, malicious relay operators could closely inspect exit traffic (e.g., by examining the user agent string of HTTP requests) and only target connections which appear to be genuine Tor users.

HTTPS McCoy et al. [24] showed that HTTP is the most popular protocol in the Tor network, clearly dominating other protocols such as instant messaging or e-mail.[7] While HTTPS lags behind, it is still widely used and unsurprisingly, several exit relays were documented to have tampered with HTTPS connections [18] in the past. We implemented an HTTPS module which fetches a decoy destination's X.509 certificate and extracts its fingerprint. This fingerprint is then compared to the expected fingerprint which is hard-coded in the module.[8] If there is a mismatch, an alert is triggered. Originally, we began by fetching the certificate using the command line utility gnutls-cli. We later extended the module to send a TLS client hello packet as it is sent by TorBrowser to make the scan less distinguishable from what a real Tor user would send. Note that an attacker might become suspicious after observing that a Tor user only fetched an X.509 certificate without actually browsing the respective web site. However, at the point in time an attacker would become suspicious, we already have what we need; namely the X.509 certificate.

[7]This is particularly true for *connections* but not so much for *bytes transferred*.

[8]Note that it is also possible for modules to fetch the certificate over a direct Internet connection instead of hard-coding the fingerprint.

```
function probe( fingerprint, command ) {

    ssh_public_key = "11:22:33:44:55:66:77:88:99:00:aa:bb:cc:dd:ee:ff";

    output = command.execute("ssh -v decoy.host.com");

    if (ssh_public_key not in output) {
        print("Possible MitM attack by " + fingerprint);
    }
}
```

Fig. 4. Pseudo code illustrating a scanning module which probes SSH. It establishes an SSH connection and verifies if the fingerprint matches the expected value. If the observed fingerprint differs, an alert is raised.

XMPP and IMAPS Analogous to the HTTPS module, these two modules establish a TLS connection to a decoy destination, extract the server certificate's fingerprint, and compare it to the respective hard-coded fingerprint.

sslstrip Instead of *interfering* with TLS connections, an attacker can seek to *prevent* TLS connections. This is the purpose of the tool `sslstrip` [22]. The tool achieves this goal by transparently rewriting HTML documents while on their way from the server to the client. In particular, it rewrites HTTPS links to HTTP links. A secure login form pointing to `https://login.example.com` is subsequently rewritten to HTTP which causes a user's browser to submit her credentials in the clear. While the HTTP Strict Transport Security policy [15] prevents sslstrip, it is still an effective attack against many large-scale web sites with Yahoo! being only one of them as of January 2014. From an attacker's point of view, the benefit of `sslstrip` is that it is a comparatively silent attack because browsers will not show certificate warnings. Vigilant users, however, might notice the absence of browser-specific TLS indicators such as lock icons. Our probing module fetches web sites containing HTTPS links over unencrypted HTTP. Afterwards, the module simply verifies whether the fetched HTML document contains the expected HTTPS links or if they were "downgraded" to HTTP.

SSH The Tor network is also used to transport SSH traffic. This can easily be done with the help of tools such as `torsocks` [36]. Analogous to HTTPS-based attacks, malicious exit relays could run MitM attacks against SSH. In practice, this is not as easy as targeting HTTPS given SSH's "trust on first use" model. As long as the very first connection to an SSH server was secure, the public key is then stored by the client and kept as reference for subsequent connections. As a result, a MitM attack has to target a client's very first SSH connection. Nevertheless, this practical problem might not stop attackers from attempting to interfere with SSH connections. Our SSH module—conceptually similar to the pseudo code shown in Fig. 4—makes use of OpenSSH's ssh and torsocks to connect to a decoy server. Again, the server's key fingerprint is extracted and compared to the hard-coded fingerprint. However, compared to the HTTPS module, it is difficult to achieve indistinguishability over time. After all, a malicious relay operator could monitor an entire SSH session. If it looks suspicious, e.g., it only fetches the public key, or it lasts only one second, the attacker could decide to whitelist the destination in the future. To work around this problem, we could establish SSH connections to random hosts on the Internet. This, however, is often considered undesired scanning activity and does not constitute good Internet citizenship. Instead, we again seek to solve this problem by publishing our source code and encouraging people to crowdsource exitmap scanning. Every exitmap user is encouraged to use her own SSH server as decoy destination. That way, we hope to achieve destination diversity without bothering arbitrary SSH servers on the Internet.

DNS While the Tor protocol only transports TCP streams, clients can ask exit relays to resolve DNS records by wrapping domain names in a RELAY_BEGIN cell [9]. Once a circuit is established, this cell is then sent to the exit relay for resolution. In the past, some exit relays were found to inadvertently censor DNS queries, e.g., by using an OpenDNS configuration which blocks certain domain categories such as "pornography" or "proxy/anonymizer" [18]. Our probing module maintains a whitelist of domains together with their corresponding IP addresses and raises an alert if the DNS A record of a domain name is unexpected. This approach works well for sites with a known set of IP addresses but large sites frequently employ a diverse—and sometimes geographically load-balanced—set of IP addresses which is difficult to enumerate. Our module probes domains in the categories finance, social networking, political activism, and pornography.

3.2 The Design of HoneyConnector

HoneyConnector is a framework for establishing bait connections over Tor using unique credentials over FTP and IMAP and detecting their subsequent use to identify sniffing exit relays. The framework can be divided into several components. It consists of the HoneyConnector client which is written in Python, a copy of the Tor client, the Stem [32] library for controlling Tor connections, and a backend database for storing our bait credentials and timestamps, as well as additional exit relay information. The HoneyConnector client is responsible for creating new credentials, establishing the actual bait connection over the respective exit relays, and communicating them to the deployed services over a secure channel for creating the accounts and bait data. Furthermore, HTTPS certificates are fetched by the client and compared with the real certificates to detect MitM attacks against HTTPS. Credentials are checked for duplicates prior to using them, to prevent reusing usernames or passwords. For each protocol—in our current implementation FTP and IMAP—a virtual machine is used for hosting these services, and accessed over the Tor network using bait credentials. This makes it possible to quickly deploy multiple instances for each service. Analogous to exitmap, HoneyConnector uses the library Stem to have control over which exit relay is selected for a circuit and to check if a given exit relay's exit policy allows connections to our bait services. We manually looked for peculiarities in our bait sessions which could have been used to identify them. Afterwards, we changed the status messages sent by pyFTPdlib to match those of vsFTPd. It was not necessary to change the behavior of Dovecot as it is a common mail server found on GNU/Linux systems. Once a scan is started, the network consensus is downloaded and all exit relays are processed sequentially after random permutation.

FTP scanning Our HoneyConnector client made use of the Python library ftplib to connect to our bait FTP server. The credentials for the FTP server were generated by the HoneyConnector client, stored in a database, and then forwarded to the FTP server over a secure channel. All FTP usernames are generated by randomly choosing a prefix out of "web", "user", "ftp", "usr", or two random letters followed by a random number between 1 and 999 in combination with a randomly generated password. After sending the credentials, our client waits for 30 seconds in order to assure that the server had enough time to populate the FTP user directory. The client then connects over the Tor network to the FTP server and downloads a random file before closing the connection. After the connection was closed, the client sends a message directly (i.e., not over Tor) to the server, instructing it to delete the user from the server. On the server side we used the pyFTPdlib Python library as it allowed us to modify the source code for logging plaintext credentials; a feature which was hard to find in other FTP server software. The concatenation of username and password allowed us to identify which exit relay sniffed a given pair of credentials.

IMAP scanning For implementing our IMAP scan, we used Python's built-in library imaplib. On the server side we used Dovecot due to it being a popular IMAP server and offering the possibility of verbose authentication logging, including writing usernames and passwords to a log file. We believe that for sniffers, IMAP is more interesting than POP3 since messages are kept on the server. As a result, it is stealthier for an attacker to browse the victim's e-mails as they are kept on the server rather than being deleted after downloading them. We reused password lists from the Honeynet Project [27] instead of generating them randomly. These passwords mimic real user passwords, and we manually verified that we do not falsely count regular bruteforce attacks as reconnection (i.e. no other connection attempts within close time vicinity). We further populated all mailboxes with dummy e-mails (the exact amount was randomly chosen from {1..6000}). The e-mails do not need actual content as only the amount of e-mails in the mailbox is transferred but no content. We designed our IMAP setup analogous to the FTP setup discussed above; login credentials are first generated and then sent to the server after their uniqueness was verified. The HoneyConnector client then sleeps for a while to give the server time to populate the newly created mailbox. Subsequently, HoneyConnector simulates an e-mail client checking for new mails. Finally, the client instructs the server to delete the e-mail account and thereby terminates the bait IMAP connection.

4 Experimental Results

The following two sections present the results we obtained by monitoring all Tor exit relays over a period of several months. We begin by presenting active attacks in Section 4.1 which is then followed by sniffing attacks in Section 4.2.

4.1 Detecting MitM Attacks with exitmap

Scanning Performance On September 19th, we ran our first full scan over all ∼950 exit relays which were part of the Tor network at the time. From then on, we scanned all exit relays several times a week. Originally, we began our scans while only armed with our HTTPS module but as time passed, we added additional modules which allowed us to scan for additional attacks. In this section, we will discuss the results we obtained by monitoring the Tor network over a period of seven months.

exitmap is also useful to measure the reliability of exit relays. While running our scans, we observed that 84%–88% of circuit creations succeeded. The remaining circuits either timed out or were torn down by the respective exit relay using a DESTROY cell. The performance of our probing modules for all responsive exit relays is illustrated in Fig. 5. The ECDF's x-axis shows the amount of seconds it takes for a module to finish successfully. The y-axis shows the cumulative fraction of all exit relays. The diagram shows that all modules are able to scan at least 98% of all responsive Tor exit relays under 50 seconds. Note that it is possible to artificially slow down exitmap in order to make scans more difficult to detect. At maximum speed, it would be easier for colluding exit relays to correlate decoy connections and mark them as possibly coming from exitmap.

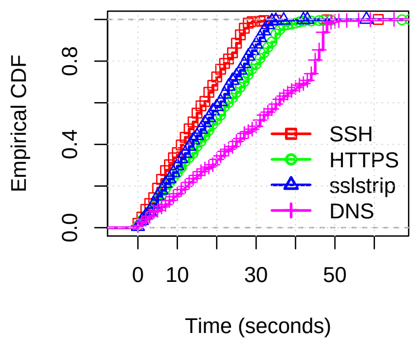


Fig. 5. The performance of some of our probing modules. The DNS module is slower because it resolves several domain names at once. All other modules can scan at least 98% of all responsive Tor exit relays under 40 seconds.

Malicious Relays Table 1 contains all *40 malicious and misconfigured exit relays* we found. We discovered the first two relays "manually" before we had developed exitmap. All data illustrated in the table was gathered on the day we found the respective attack. It includes the first 4 bytes of the relay's unique 20-byte SHA-1 fingerprint, the IPv4 addresses or netblocks the relay was found to have used over its life time, the advertised bandwidth and the country in which the relay resided according to MaxMind's GeoIP lite database. Furthermore, the relay's configuration problem or the attack it was running, the day the relay was set up and the day we discovered the relay's malicious activity.

Apart from all the conspicuous HTTPS MitM attacks which we will discuss in Section 5.2, we exposed several relays running `sslstrip`. The relay `5A2A51D4` injected custom HTML code into HTTP traffic (see Appendix B and Section 5.1). The injected HTML code was discovered by our sslstrip module which assures that the returned HTML code is exactly as expected. Besides, relays in Malaysia, Hong Kong, and Turkey were subject to DNS censorship. The relays in Hong Kong seem to have fallen prey to the Great Firewall of China's DNS poisoning; perhaps, the relays made use of a DNS resolver in China. Several domains such as torproject.org, facebook.com and youtube.com returned invalid IP addresses which were also found in previous work [21]. Finally, four relays were misconfigured as they used an OpenDNS policy which censored at least web sites in the category "pornography". The last two relays in the table ran anti virus products which broke into IMAPS sessions; presumably for content inspection. All the remaining relays engaged in HTTPS, SSH, and XMPP MitM attacks. Upon establishing a connection to the decoy destination, these relays exchanged the destination's certificate with their own, self-signed version. Since these certificates were not issued by a trusted authority stored in TorBrowser's certificate store, a user falling prey to such a MitM attack would be redirected to the `about:certerror` warning page. We will discuss some attacks in greater detail in Section 5.

4.2 Detecting Traffic Sniffing with HoneyConnector

We deployed HoneyConnector on October 13th, 2013, and after an initial testing phase of two weeks on a residential service provider network we deployed it on multiple hosting providers across Europe. The evaluation period lasted until February 10th, 2014, resulting in approximately four months overall deployment. The modified FTP server was deployed in Germany with Hetzner Hosting while the modified IMAP server was deployed with OVH. HoneyConnector can be configured to use multiple server instances of the services, but for the evaluation we decided to go with the baseline minimum of two virtual machines. We cannot ascertain whether the destination's network location was an additional incentive for sniffers, e.g., due to its IP range or hostname. Furthermore, these hosting services might be of particular interest for sniffers since (in the attackers perception) there is no central software update mechanism available to customers, the servers have high availability and high bandwidth, and are prone to misconfiguration due to inexperienced customers. The client establishing the bait connections was run locally on our own machines.

Table 1. All 40 malicious and misconfigured exit relays we discovered over a period of seven months. The data was collected right after a relay was discovered. We have reason to believe that all relays whose fingerprint ends with a † were run by the same attacker.

Fingerprint	IP addresses	Country	Bandwidth	Problem	First active	Discovery
F8FD29D0†	176.99.12.246	Russia	7.16 MB/s	HTTPS MitM	2013-06-24	2013-07-13
8F9121BF†	64.22.111.168/29	U.S.	7.16 MB/s	HTTPS MitM	2013-06-11	2013-07-13
93213A1F†	176.99.9.114	Russia	290 KB/s	HTTPS MitM (50%)	2013-07-23	2013-09-19
05AD06E2†	92.63.102.68	Russia	5.55 MB/s	HTTPS MitM (33%)	2013-08-01	2013-09-19
45C55E46†	46.254.19.140	Russia	1.54 MB/s	SSH & HTTPS MitM (12%)	2013-08-09	2013-09-23
CA1BA219†	176.99.9.111	Russia	334 KB/s	HTTPS MitM (37.5%)	2013-09-26	2013-10-01
1D70CDED†	46.38.50.54	Russia	929 KB/s	HTTPS MitM (50%)	2013-09-27	2013-10-14
EE215500†	31.41.45.235	Russia	2.96 MB/s	HTTPS MitM (50%)	2013-09-26	2013-10-15
12459837†	195.2.252.117	Russia	3.45 MB/s	HTTPS MitM (26.9%)	2013-09-26	2013-10-16
B5906553†	83.172.8.4	Russia	850.9 KB/s	HTTPS MitM (68%)	2013-08-12	2013-10-16
EFF1D805†	188.120.228.103	Russia	287.6 KB/s	HTTPS MitM (61.2%)	2013-10-23	2013-10-23
229C3722	121.54.175.51	Hong Kong	106.4 KB/s	sslstrip	2013-06-05	2013-10-31
4E8401D7†	176.99.11.182	Russia	1.54 MB/s	HTTPS MitM (79.6%)	2013-11-08	2013-11-09
27FB6BB0†	195.2.253.159	Russia	721 KB/s	HTTPS MitM (43.8%)	2013-11-08	2013-11-09
0ABB31BD†	195.88.208.137	Russia	2.3 MB/s	SSH & HTTPS MitM (85.7%)	2013-10-31	2013-11-21
CADA00B9†	5.63.154.230	Russia	187.62 KB/s	HTTPS MitM	2013-11-26	2013-11-26
C1C0EDAD†	93.170.130.194	Russia	838.54 KB/s	HTTPS MitM	2013-11-26	2013-11-27
5A2A51D4	111.240.0.0/12	Taiwan	192.54 KB/s	HTML Injection	2013-11-23	2013-11-27
EBF7172E†	37.143.11.220	Russia	4.34 MB/s	SSH MitM	2013-11-15	2013-11-27
68E682DF†	46.17.46.108	Russia	60.21 KB/s	SSH & HTTPS MitM	2013-12-02	2013-12-02
533FDE2F†	62.109.22.20	Russia	896.42 KB/s	SSH & HTTPS MitM (42.1%)	2013-12-06	2013-12-08
E455A115	89.128.56.73	Spain	54.27 KB/s	sslstrip	2013-12-17	2013-12-18
02013F48	117.18.118.136	Hong Kong	538.45 KB/s	DNS censorship	2013-12-22	2014-01-01
2F5B07B2	178.211.39	Turkey	204.8 KB/s	DNS censorship	2013-12-28	2014-01-06
4E2692FE	24.84.118.132	Canada	52.22 KB/s	OpenDNS	2013-12-21	2014-01-06
A1AF47E3	207.98.174.40	U.S.	98.3 KB/s	OpenDNS	2013-12-20	2014-01-24
BEB0BF4F†	37.143.14.176	Russia	1.54 MB/s	XMPP MitM	2013-12-16	2014-01-25
C37AFA7F	81.219.51.206	Poland	509.3 KB/s	OpenDNS	2014-02-03	2014-02-06
975ACB99	54.200.151.237	U.S.	2.73 MB/s	sslstrip	2014-01-26	2014-02-08
B40A3DC6	85.23.243.147	Finland	50 KB/s	IMAPS anti virus	2013-11-04	2014-02-10
E5A75EE1	132.248.80.171	Mexico	102.4 KB/s	IMAPS anti virus	2013-04-24	2014-02-10
423BCBCE	54.200.102.199	U.S.	702.66 KB/s	sslstrip	2014-02-13	2014-02-14
F7B4BC6B	54.213.13.21	U.S.	431.78 KB/s	sslstrip	2014-02-14	2014-02-15
DB7C7DDD	37.143.8.242	Russia	267.86 KB/s	sslstrip	2014-02-18	2014-02-18
426E8E2F	54.201.48.216	U.S.	2.25 MB/s	sslstrip	2014-02-09	2014-02-18
D81DAC47	117.18.118.136	Hong Kong	166.31 KB/s	DNS censorship	2014-01-27	2014-02-14
BDBFBBC3	209.162.33.125	U.S.	806.46 KB/s	OpenDNS	2014-03-06	2014-03-06
564E995A	67.222.130.112	U.S.	204.8 KB/s	sslstrip	2013-08-19	2014-03-13
7F2240BF	198.50.244.31	Canada	721.47 KB/s	sslstrip	2014-03-27	2014-04-04
DA7A2EDC	121.121.82.198	Malaysia	82.79 KB/s	DNS censorship	2014-03-07	2014-04-15

During the four month deployment of HoneyConnector, we registered a total of 255 login attempts with 128 sniffed plaintext credentials, tracing back to *27 sniffing exit relays*. Among all 255 login attempts, 136 were targeting FTP and 119 were targeting IMAP. From all 128 sniffed credentials, 97 were for FTP and 31 for IMAP. We observed one of the relays using two different Tor identity fingerprints for different login attempts and sniffed credentials, but since the nickname and IP address stayed the same and Tor's software version changed, we counted it only once. The identity fingerprint can be changed during software updates if no precautions are met by the operator. Overall, 2,611 distinct servers (based on the identity fingerprints) have seen bait connections, but this is considered an upper bound since there was a Tor software update from version 0.2.3 to 0.2.4 in December and there were up to approximately 1,000 exit relays online during the evaluation. Even though the HoneyConnector architecture was initially unstable, a login attempt with sniffed credentials was already registered during the very first night of stability testing (October 13th). In total, we conducted approximately 27,000 bait connection for FTP and IMAP each, resulting in approximately 54,000 plaintext credentials created by the HoneyConnector client software. A total of 0.24% of these credentials were used during reconnects by the sniffing exit relay operators.

Table 2 shows the details of all sniffing exit relays we discovered. Again, it includes the first 4 bytes of the relay's unique 20-byte SHA-1 fingerprint, the relay's bandwidth and country (also resolved using the GeoIP lite database). The triple in angle brackets represents the *1)* unique number of plaintext credentials sent, the *2)* number of different plaintext credentials used by the malicious operator (a subset of the set of unique credentials sent) as well as the *3)* total number of connection attempts conducted with these credentials. If a relay's exit policy permitted it, both IMAP and FTP were used for bait connections. Furthermore, the table shows whether the operator tried to log in using the FTP or the IMAP credentials, or both. The distribution of login attempts over the four month period can be seen in Fig. 6. FTP login attempts are shown as triangles and IMAP as squares. At most, there were ten FTP login attempts a day, whereas IMAP peaked at 33 login attempts a day. On average and across all sniffing nodes, about 60% of the bait credentials sent were used.

Another aspect is the time interval in between the bait connection made by HoneyConnector and the subsequent reconnect by the exit node operator. Fig. 7 shows the time in interval between the transmission of the bait credentials and the reconnection attempts, clustered in (non-linear) time intervals. While the light gray bars only account for the first reconnection attempt, the darker bars account for all reconnection attempts, including repeatedly using the same credentials. About 25% of all login attempts were made within the first eight hours, while half of all reconnection attempts were made within 48 hours. The shortest time period until the first observed reconnection attempt was only three minutes and ten seconds and done by the Estonian exit node "FreedomFighter". The

Table 2. All 27 exit relays which were found sniffing login credentials. The triple reads <no. of credentials sent, no. of credentials tried, no of connection attemtps>, *dynamic* refers to multiple IPs from 120.56.0.0/14 and 59.176.0.0/13

Fingerprint	IP addresses	Country	Bandwidth	Sniffed Protocol	HoneyConnection	Reconnection
`08F097F8`	58.120.227.83	South Korea	1136.64 KB/s	FTP <36,35,70>	2013-10-17	2013-10-17
`0FE41A85`	46.246.108.146	Sweden	4326.85 KB/s	FTP <1,1,6>	2014-01-20	2014-01-21
`229C3722`	121.54.175.51	Hong Kong	168.74 KB/s	FTP <2,1,14>	2013-11-04	2014-01-07
`28619F94`	dynamic	India	51.94 KB/s	IMAP & FTP <15,4,50>	2013-11-07	2013-11-13
`319D548B`	91.219.238.139	Hungary	1075.2 KB/s	FTP <2,1,47>	2013-12-24	2013-12-14
`3A484AFC`	dynamic	India	73.4 KB/s	IMAP & FTP <15,7,55>	2013-10-27	2013-10-30
`52E24E09`	dynamic	India	57.15 KB/s	IMAP & FTP <7,6,44>	2013-10-17	2013-10-18
`5761CB9C`	109.87.249.227	Ukraine	2.05 KB/s	FTP <6,2,4>	2013-11-28	2013-11-28
`5A2A51D4`	111.240.0.0/12	Taiwan	75.47 KB/s	IMAP <1,1,57>	2013-11-02	2014-01-20
`5A3B2DEC`	66.85.131.84	U.S.	512.0 KB/s	IMAP <6,2,33>	2013-11-30	2013-12-03
`6018E567`	51.35.183.211	U.K.	312.1 KB/s	FTP <1,1,6>	2014-01-24	2014-01-24
`61288460`	88.150.227.162	U.K.	353.0 KB/s	IMAP & FTP <31,3,11>	2013-11-14	2013-11-15
`6C9AAFEA`	dynamic	India	53.95 KB/s	IMAP & FTP <20,12,44>	2013-10-17	2013-10-18
`46B3ADE6`	85.17.183.69	Netherlands	234.18 KB/s	FTP <2,1,6>	2013-12-27	2014-01-09
`8450F3CA`	moved once	Germany	2938.88 KB/s	FTP <12,7,16>	2013-12-16	2013-12-16
`8A47C9B0`	100.42.236.34	U.S.	237.4 KB/s	FTP <3,1,4>	2013-12-03	2013-12-05
`9F7DBC53`	76.74.178.217	U.S.	133.57 KB/s	FTP <1,1,1>	2013-12-16	2013-12-17
`A68412BA`	moved once	U.S.	989.67 KB/s	FTP <7,5,13>	2013-12-16	2013-12-17
`AA6D6919`	85.25.46.189	Germany	59.52 KB/s	FTP <2,1,2>	2013-10-17	2013-10-19
`ADE35AA1`	dynamic	India	35.53 KB/s	IMAP & FTP <3,3,15>	2013-10-18	2013-10-18
`BF74938A`	89.79.83.166	Poland	1979.39 KB/s	FTP <7,1,7>	2013-12-23	2013-12-23
`C5398CD1`	dynamic	India	53.82 KB/s	IMAP & FTP <14,9,43>	2013-10-14	2013-10-15
`EBCA226D`	46.246.95.193	Sweden	2737.89 KB/s	FTP <1,1,1>	2014-01-21	2014-01-23
`F0AAFC6D`	dynamic	India	56.65 KB/s	IMAP & FTP <30,16,56>	2013-10-17	2013-10-18
`F0DD7385`	76.189.8.28	Canada	111.42 KB/s	FTP <1,1,21>	2013-10-14	2013-10-14
`F57E0775`	151.217.63.51	Germany	537.62 KB/s	IMAP & FTP <24,2,2>	2013-12-29	2013-12-29
`FEE8C068`	46.22.211.36	Estonia	119.51 KB/s	FTP <5,5,57>	2013-11-21	2013-11-22

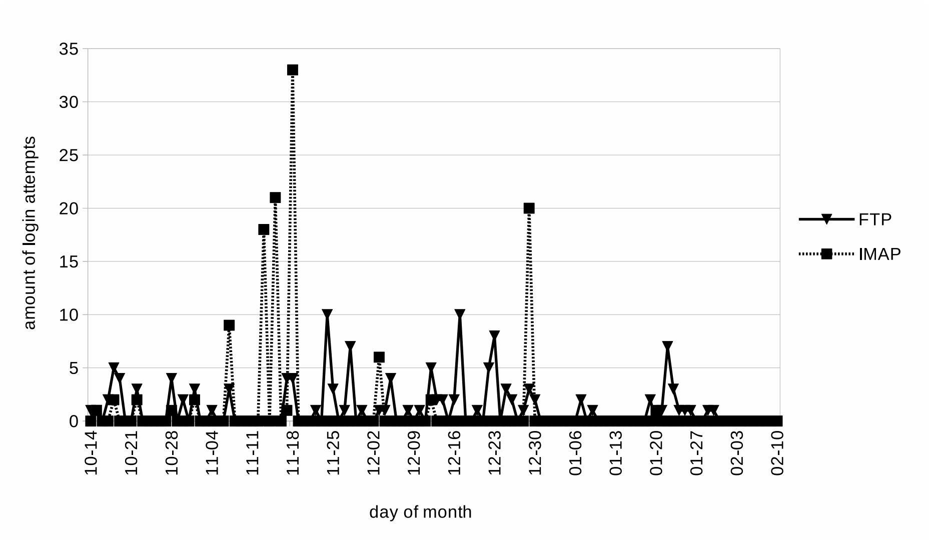


Fig. 6. The diagram illustrates the amount of rogue login attempts over time. While we did not witness any login attempts for most days, some days saw up to 33 login attempts.

longest observed time interval was related to the reconnection made by "default" located in Hong Kong, with credentials that were sent more than two months before (63 days).

5 Discussion

After having presented an overview of our results in Section 4, we now focus on and discuss several interesting aspects of our data sets. In particular, we found several instances of colluding exit relays, destination targeting, and human errors among malicious exit relay operators.

5.1 Data Set Overlap

Only two exit relays were caught by both of our scanners, exitmap as well as HoneyConnector. The first one, 5A2A51D4, was located in Taiwan and was found to sniff IMAP credentials as well as to inject HTML code (see Appendix B). While the HTML code was not malicious at the time we tested the relay, it is possible that the injected code changed over time or that the code changed depending on the HTTP Host header sent by the Tor user. The second relay which was located in Hong Kong, 229C3722, ran sslstrip as well as sniffed FTP credentials.

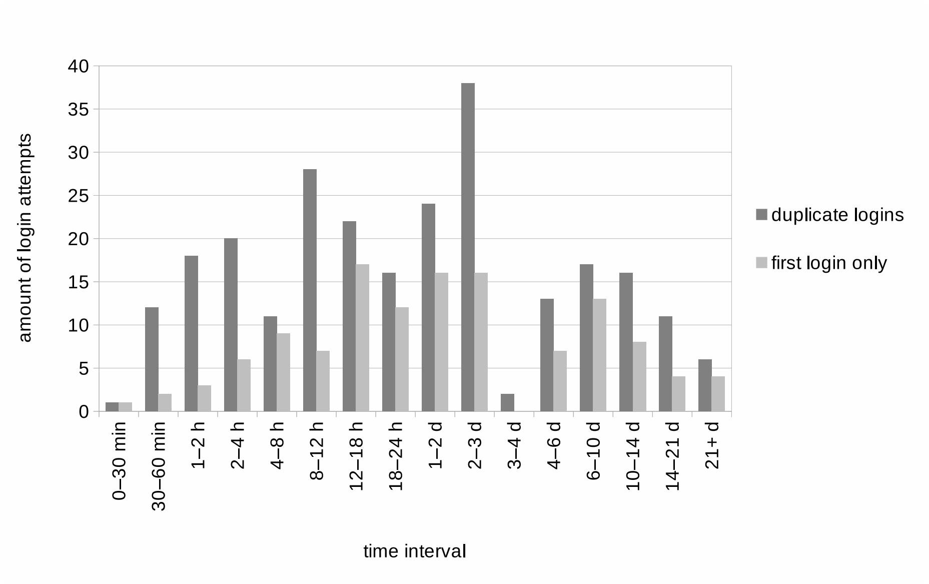


Fig. 7. The time interval between the honey connection and the login attempts.

5.2 The "Russian HTTPS Group"

Interestingly, we have reason to believe that all relays in Table 1 whose fingerprint ends with a † were run by *the same person or group of people*. This becomes evident when analyzing the self-signed certificates which were injected for the MitM attacks. In every case, the certificate chain consisted of only two nodes which both belonged to a "Main Authority" and the root certificate of all chains—shown in Appendix A—was *identical*. This means that these attacks can be traced back to a common origin even though it is not clear where or what this origin is as we will discuss later. Apart from the identical root certificate, these relays had other properties in common. First, with the exception of `8F9121BF` which was located in the U.S., they were *all located in Russia*. Upon investigating their IP addresses, we discovered that most of the Russian relays were run in the network of a virtual private system (VPS) provider. Several IP addresses were also located in the same netblock, namely 176.99.12.246, 176.99.9.114, 176.99.9.111, and 176.99.11.182. All these IP addresses are part of the netblock GlobaTel-net which spans 176.99.0.0/20. Furthermore, the malicious exit relays all used Tor version 0.2.2.37.[9] Given its age, this is a rather uncommon version number among relays. In fact, we found only two benign exit relays—in Switzerland and the U.S.—which are running the same version. We suspect that the attackers might have a precompiled version of Tor which they simply copy to newly purchased systems to spawn new exit relays. Unfortunately,

[9]For comparison, as of January 2014, the current stable version is 0.2.4.20. Version 0.2.2.37 was declared stable on June 6th, 2012.

we have no data which would allow us to verify when this series of attacks began. However, the root certificate shown in Appendix A indicates that it was created on February 12, 2013.

Connection Sampling Whenever our hunt for malicious relays yielded another result, we first tried to confirm the attack by rerunning the scan over the newly discovered relay. However, in the case of the Russian relays, this did not always result in the expected HTTPS MitM attack. Instead, we found that only every nth connection seemed to have been attacked. We estimated the exact *sampling rate* by establishing 50 HTTPS connections over every relay. We used randomly determined sleep periods in between the scans in order to disguise our activity. The estimated sampling rate is shown in Table 1 next to the respective attack in parentheses. For all Russian relays, it varies between 12% and 68%. We do not have an explanation for the attacker's motivation to sample connections. One theory is that sampling makes it less likely for a malicious exit relay to be discovered; but at the cost of collecting fewer MitM victims. Interestingly, the sampling technique was implemented *ineffectively*. This is due to the way how Firefox (and as a result TorBrowser) reacts to self-signed certificates. When facing a self-signed X.509 certificate, Firefox displays its `about:certerror` page which warns the user about the security risk. If a user then decides to proceed, the certificate is *fetched again*. We observed that the malicious exit relays treat the certificate re-fetching as a separate connection whose success again depends on the relay's sampling rate. As a result, a sampling rate of n means that a MitM attack will only be successfully with a probability of n^2 rather than n.

Who is the Attacker? An important question is where on the path from the exit relay to the destination the attacker is located. At first glance, one might blame the exit relay operator. However, it is also possible that the actual attack happens *after* the exit relay, e.g., in the exit relay's ISP, the network backbone, or the destination's ISP. In fact, such an incident was documented in 2006 for a relay located in China [7]. With respect to our data, we cannot entirely rule out that the HTTPS MitM attacks were actually run by an upstream provider of the Russian exit relays. However, we consider it unlikely for the following reasons: *1)* the relays were located in diverse IP address blocks and there were numerous other relays in Russia which did not exhibit this behavior, *2)* one of the relays was even located in the U.S., *3)* there are no other reported cases on the Internet involving a certification authority called "Main Authority", and *4)* the relays frequently disappeared after they were assigned the BadExit flag. The identity of the attacker is difficult to ascertain. The relays did not publish any contact information, nicknames, or revealed other hints which could enable educated guesses regarding the attacker's origin.

Destination Targeting While Tor's nature as an anonymity tool renders targeting individuals difficult,[10] an attacker can target classes of users based on their communication *destination*. For example, an attacker could decide to only tamper with connections going to the fictional www.insecure-bank.com. Interestingly, we found evidence for exactly that behavior; at some point the Russian relays began to target at least facebook.com. We tested the HTTPS version of the Alexa top 10 web sites [1] but were unable to trigger MitM attacks despite numerous connection attempts. Popular Russian web sites such as the mail provider mail.ru and the social networking site vk.com also remained unaffected. Note that it is possible that the relays targeted additional web sites we did not test for. Enumerating targeted web sites would mean probing thousands of different web sites. We have no explanation for the targeting of destinations. It might be another attempt to delay discovery by vigilant users. However, according to previous research [16], social networking appears to be just as popular over Tor as it is over the open Internet. As a result, limiting the attack to facebook.com might not delay discovery significantly.

5.3 The "International Sniffer Group"

A group of international exit relays in Table 2 is obviously colluding with the clear intent of sniffing credentials as the credentials that were sent over these nodes were tested in batches. Since the relays are spread over Europe and the U.S., we called it the International group, even though it is possible that they are all operated by the same single person. It consists of the five relays "Chupacabras", "AlleyCAT", "NennoExit", "Aragaun" (Previously "UMBRELLAxCORP" at the same IP address), and "ShredOwl", located in the U.S., Germany, Netherlands, and Sweden. One of the nodes, "Chupacabras", moved from Germany to the U.S. during our evaluation.

5.4 The "Indian Sniffer Group"

The second group that stuck out during our evaluation is a group of *seven Indian exit relays* in Table 2. These relays were responsible for 104 out of all 255 reconnection attempts (41%) and employed a number of distinguishable reconnect patterns that are unique to this group. All of the seven nodes within this group were operated on dynamic allocated IP addresses belonging to the ISP "Mahanagar Telephone Nigam Ltd.", and had a bandwidth between 50 and 80 KB/s. All relays ran Tor in version 0.2.3.25 on Microsoft Windows; four relays ran Windows 7, while three relays ran Windows Vista. Furthermore, the nodes seemed to change their IP address every six hours, resulting in bad uptime statistics for them. Because of the low bandwidth bundled with the poor uptime statistics, the probability of Tor exit traffic being routed over these nodes is very low.

[10]We assume of course that Tor users do not deliberately reveal their real identity, e.g., by posting on Internet forums under their real name.

Most login attempts were made by using the Mail2Web service [28] which obfuscates the real source of the connection. In fact, Mail2Web was solely used by the group of Indian nodes. However, fingerprints of Mozilla Thunderbird version 3.1.20 on Windows Vista was used over Tor on two occasions in November. For reconnecting with FTP, either Microsoft Internet Explorer or Mozilla Firefox was used. All connections made with Internet Explorer originated from one of the nodes which was running at this time which suggests that this browser used the Internet connection directly. All login attempts made with Firefox were conducted through the Tor network but from different exit relays which suggests the use of TorBrowser. The variety of software used and the number of concurrent IP addresses point in the direction that those nodes are operated by more than one individual, although not conclusively.

5.5 Who Reused the Bait Credentials?

For HoneyConnector, the majority of reconnects—145, or 57%—was conducted over the Tor network, i.e., the IP address was part of the Tor network (verified using ExoneraTor [29]) but not the relay which sniffed the credentials. This comes as no surprise since exit relay operators are expected to be familiar with the Tor network. Therefore, it is difficult to conclude who initiated the reconnect. However, 45 reconnections (18%) originated from the same IP address as the exit relay which originally sniffed the credentials. This means that the malicious operator could have used the exit relay for a direct connection (i.e., not over Tor) or Tor was manually configured to use this particular relay as exit. 16% (41) of all reconnections used the service Mail2Web. Since the servers of this service connect directly to a given IMAP server, it is not possible to assess if the user was additionally using Tor, or used this service directly. However, this service was only used by the operator or group of operators from the Indian exit nodes. In 22 cases (9%) of all reconnections, the source IP address was no Tor relay and we were unable to associated the IP address with any VPN service, meaning that the connection was likely originating from a host under the direct control of the relay operator. Within this subset, we found connections from IP addresses that belonged to hosting companies, mobile UMTS Internet services, and private home connections by consumer Internet service providers. One IP address was found to belong to a Japanese university. In two cases, the reverse DNS record of the respective IP address suggests that a VPN service was used.

The software used for the reconnections can also reveal information about the relay operator as its default configuration can be unsuitable for the Tor network [8], and improper usage of client software can lead to deanonymization [3, 4]. This includes default login credentials such as "mozilla@example.com" as password for an anonymous FTP login with Mozilla Firefox, attempts to fetch data over side channels, e.g., Mozilla Thunderbird trying to fetch an XML file containing data for automatic configuration, or simply the IP address of freely available web services such as Mail2Web. The largest amount of reconnections—117, or 46%—contained no hints or direct information on the software used.

Mail2Web was used in 41 (or 16%) reconnections, but since it is a web service connecting to IMAP accounts through a web interface, no additional information could be inferred. All credentials used were sent over Indian relays (see Section 5.4). The connections by Mail2Web were easily recognizable due to the reverse record of the respective IP addresses. The Indian nodes as well as the operator of the British node "AstralNode" used Thunderbird for IMAP reconnections (20% or 51 reconnections) since Thunderbird issues a request for an XML file during account setup containing instructions for automatic configuration. The German exit node "h0rny30c3" is also very likely to have used Thunderbird due to connection patterns, but the request for the auto configuration XML file was not found. As for FTP reconnections, we could identify the use of Firefox and Internet Explorer. The Firefox-based TorBrowser is the browser of choice in the Tor Browser Bundle (TBB), and as such the recommended way of accessing the Tor network. Firefox was used in 25 reconnection (10%), Internet Explorer was used for 21 reconnections (9%).

5.6 Human Errors During Reconnections

Using sniffed credentials is harder than it seems, and we found multiple peculiarities in our logs that we would like to share as well. Out of all 255 reconnection attempts, 31 (or 12%) were made with incorrect credentials, in most cases with apparent copy-paste errors by omitting characters at the beginning or the end of a password; specifically when punctuation or special characters were used. Other instances included multiple pastes of the same password, omitted parts of the IMAP username or typographical errors showing that these passwords were typed manually. We were also able to observe that sniffers monitoring multiple protocols can become confused as to what credentials to use for which service: the node "SuperDuperLative" for example used IMAP bait credentials for FTP reconnections, twice. The operator also tried the password with and without quotation marks in another instance. The operator of the Indian node "atlas" was mixing two different username and password combinations, trying to authenticate for one username with the password of another username. The operator of the relay "pcrrtor1" used a seemingly random password that was not sent as part of any bait connection at all from our side. The operators of "Chupacabras" and "ShredOwl" seem to have pasted the FTP URL into the wrong browser—in both cases they revealed their true IP address by using Google Chrome before switching to Firefox through Tor, clearly visible due to the default anonymous credentials tried by Firefox.

The two biggest spikes of IMAP logins seen in Fig. 6 were made by using the configuration wizard of Mozilla Thunderbird for creating a new connection to an IMAP server, in which the e-mail client uses multiple login attempts to automate setup and to verify if the connection was configured properly. The logs also indicate that further attempts were done afterwards to test and troubleshoot the configuration but since it was not possible to log in with the snooped credentials, Thunderbird would only display error messages. However, most of the recorded logins had either errors in their sequences, certain erratic-appearing client fingerprints with differing reconnection times, pointing us to the conclusion that all reconnections were conducted manually. One exception here could be the reconnections related to the relay "SuperDuperLative": most of these reconnections were made either around a static time or by processing a whole batch of sniffed credentials within a certain timeframe while using Tor.

5.7 Implications for Tor Users

A question which is of interest to Tor users is "what fraction of exit relays is malicious?". To answer this question, it is tempting but insufficient to divide our results by the total number of exit relays, e.g., $\frac{65}{1000} \approx 6\%$. This calculation is biased as it does not consider the *change of exit relays* over time. This metric is captured by the *churn rate*, i.e., the rate at which new exit relays join the network and existing ones leave.

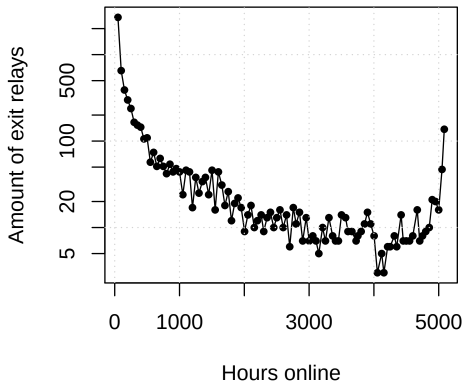


Fig. 8. The amount of hours, all 6,835 unique exit relays spent online in between Sept. 2014 and Mar. 2014. 2,698 exit relays vanished after being part of the network for 50 hours or less.

We obtained an idea of the network's churn rate by determining the amount of unique exit relays (based on the relay's identity fingerprint) which were part of the network from September 2013 to March 2014. For every unique relay, we also calculate the amount of hours, it served the network. In total, we observed *6,835 unique exit relay identity fingerprints*. The distribution—with intervals of 50 hours—is illustrated in Fig. 8. A total of 2,698 exit relays was online for only 50 hours or less in these seven months. 137 exit relays were online for 5,052 hours or more—which is close to the maximum of 5,088 hours. The diagram clearly shows that given the network's considerable churn rate, our scanners tested many more relays than the overall amount of exit relays at a given point in time. An estimate for the probability of selecting a malicious exit relay in a circuit would also require the consideration of a relay's observed bandwidth.

To protect against sniffing exit relays, end-to-end encryption should be used whenever possible. In particular, HTTPS should be preferred over its unencrypted alternative. The same applies to other protocols which have more secure TLS-based alternatives, e.g., SMTPS or IMAPS. Note that TorBrowser's

HTTPS-Everywhere extension automatically redirects the user to many HTTPS-enabled web sites whenever possible. Outside the Tor network, server operators can and should help by enabling ubiquitous encryption for all services they run, e.g., by making use of HTTP Strict Transport Security [14].

5.8 Limitations

For both our frameworks, exitmap and HoneyConnector, performing attribution is problematic, meaning that it is difficult to distinguish if the attacker is the relay operator or any other entity along the path from the exit relay to the destination. This can be for example the relay's ISP, any other ISP along the path, or a nation-state adversary. Even though it is in our opinion unlikely (due to the ease of running a malicious Tor exit relay), it cannot be ruled out entirely. Nevertheless, if such attacks seem to be run by an exit relay whereas they are in fact conducted by the network backbone, it is beneficial to all Tor users that this relay is assigned the BadExit flag.

5.9 Ethical Aspects

Due to exitmap's modular architecture, it can be used for various unintended and even unethical purposes. For example, modules for web site scraping or online voting manipulation come to mind. All sites which naively bind identities to IP addresses might be an attractive target. While we do not endorse such actions, we point out that these activities are hard to stop and will continue to happen and already happen regardless; with or without scanner. If somebody decides to abuse our scanner for such actions, it will at least spare the Tor network's resources more than a naive design. As a result, we believe that by publishing our code, the benefit to the public outweighs the damage caused by unethical usage.

6 Thwarting HTTPS MitM Attacks

The discovery of destination targeting made us reconsider defense mechanisms. Unfortunately, we cannot rule out that there are additional, yet undiscovered exit relays which target low-profile web sites. If we wanted to achieve high coverage, we would have to probe millions of web sites; and considering the connection sampling discussed in Section 5.2, this has to be done repeatedly! After all, an attacker is able to *arbitrarily reduce the scope* of the attack but we are *unable to arbitrarily scale* our scanner. This observation motivated another defense mechanism which is discussed in this section.

6.1 Threat Model

We consider an adversary who is controlling the upstream Internet connection of a small fraction of exit relays.[11] The adversary's goal is to run HTTPS-based

[11]By "fraction", we mean a relay's bandwidth as it determines how likely a client is to select the relay as part of its circuit.

MitM attacks against Tor users. We further expect the adversary to make an effort to stay under the radar in order to delay discovery. The actual MitM attack is conducted by injecting self-signed certificates in the hope that users are not scared off by the certificate warning page. Our threat model does not cover adversaries who control certificate authorities which would enable them to issue valid certificates to avoid TorBrowser's warning page. This includes several countries as well as organizations which are part of TorBrowser's root certificate store. Furthermore, we cannot defend against adversaries who control a significant fraction of the Tor network's exit bandwidth.

6.2 Multi Circuit Certificate Verification

As long as an attacker is unable to tamper with all connections to a given destination,[12] MitM attacks can be detected by fetching a public key over *differing paths in the network*. This approach was picked up by several projects including Perspectives [37], Convergence [19] and Crossbear [6]. In this section, we discuss a patch for TorBrowser which achieves the same goal but is adapted to the Tor network. Apart from NoScript and HTTPS-Everywhere, TorBrowser contains another important extension: *Torbutton*. This extension provides the actual interface between TorBrowser and the local Tor process. It directs TorBrowser's traffic to Tor's SOCKS port and exposes a number of features such as the possibility to create a new identity. Torbutton already contains rudimentary code to talk to Tor over the local control port. The control port—typically bound to 127.0.0.1:9151—provides local applications with an interface to control Tor. For example, Torbutton's "New Identity" feature is implemented by sending the `NEWNYM` signal which instructs Tor to switch to clean circuits so that new application requests do not share circuits with old requests. Torbutton already implements a useful code base for us which made us decide to implement our extension as a patch for Torbutton rather than build an independent extension.

6.3 Extension Design

Our patch hooks into the browser event `DOMContentLoaded` which is triggered whenever a document (but not necessarily stylesheets and images) is loaded and parsed by the browser. We then check if the URI of the page contains "`about:certerror`" as TorBrowser displays this page whenever it encounters a self-signed certificate. However, it is not clear whether the certificate is genuinely self-signed or part of an attack. In order to be able to distinguish between these two cases, our patch now attempts to re-fetch the certificate over at least one additional and distinct Tor circuit as illustrated in Fig. 9. We create a fresh circuit by sending `SIGNAL NEWNYM` to Tor's control port. Afterwards, we re-fetch the certificate by issuing an `XMLHttpRequest`. If the SHA-1 fingerprints of both certificates match, the certificate is probably genuine.[13] Otherwise, the user might

[12]This would be the case if an attacker controls the destination.

[13]Note that powerful adversaries might be able to control multiple exit relays, network backbones, or even the destination.

have fallen prey to a MitM attack. False positives are possible, though: large sites could have different certificates for different geographical regions. Note that we are not very likely to witness many false positives as our code is only run upon observing self-signed certificates or certificates which somehow trigger the `about:certerror` warning page.

Our extension also informs the user about a potential MitM attack. In case of differing certificates, we open a browser dialog which informs the user about the situation. A screenshot of our design prototype is shown in Fig. 10. The dialog points out that this is likely an attack and asks the user for permission to send the data to the Tor Project for further inspection. The submitted data contains the *exit relays* used for certificate fetching as well as the *observed certificates*. We transmit no other data which could be used to identify users; as a result, certificate submission is anonymous. While it is technically possible to transmit the data silently, we believe that users would not appreciate this and consider it as "phoning home". As a result, we seek to obtain informed consent.

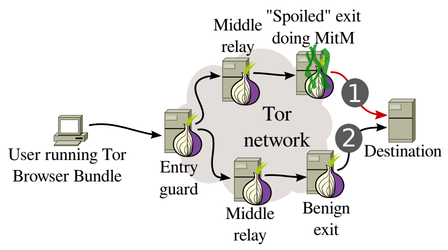
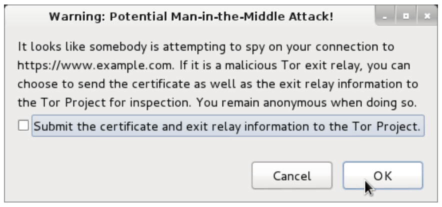


Fig. 9. A user stumbles across a self-signed certificate ❶ which could be an indication for an HTTPS MitM attack ran by a malicious exit relay. To verify if the certificate is genuine, the client re-fetches it over an independent exit relay ❷ and checks if the two certificates match or not.

6.4 Limitations

Our threat model does not consider adversaries with the ability to issue valid certificates. While our extension could easily be extended to conduct certificate comparison for all observed certificates, it would flood the Tor network with certificate re-fetches. To make matters worse, the overwhelming majority of these re-fetches would not even expose any attacks. There exist other techniques to foil CA-capable adversaries such as certificate pinning [20]. By default, our patch re-fetches a self-signed X.509 certificate only once. An attacker who is controlling a significant fraction of exit relays might be able to conduct a MitM attack for the first as well as for the second fetch. Nevertheless, we would eventually expose the


Warning: Potential Man-in-the-Middle Attack!

It looks like somebody is attempting to spy on your connection to https://www.example.com. If it is a malicious Tor exit relay, you can choose to send the certificate as well as the exit relay information to the Tor Project for inspection. You remain anonymous when doing so.

☐ Submit the certificate and exit relay information to the Tor Project.

Fig. 10. The popup window in TorBrowser which informs the user about the potential HTTPS MitM attack. The user can agree to submitting the gathered information to the Tor Project for further inspection.

attack; it would simply be a matter of time until a client selects two independent exit relays for certificate comparison.

7 Conclusions

In this paper, we revisited the trustworthiness of Tor exit relays. After developing two exit relay scanners, we closely monitored the Tor network over a period of several months. This effort led to the discovery of 65 relays which were either misconfigured or outright malicious. Interestingly, we have evidence that a non-trivial fraction of all attacks were coordinated rather than isolated. Our results further suggest that the attackers made an active effort to remain under the radar and delay detection. To protect the Tor network from malicious exit relays, we developed exitmap and HoneyConnector; easily extensible scanners which are able to probe exit relays for a variety of MitM and traffic sniffing attacks. Furthermore, we developed a patch for TorBrowser's Torbutton extension which is able to fetch self-signed X.509 certificates over different network paths in order to verify their trustworthiness. All our source code is freely available at http://www.cs.kau.se/philwint/spoiled_onions.

Acknowledgments

We want to thank Internetfonden whose research grant funded the authors from Karlstad University. This work has also been supported by COMET K1, FFG – Austrian Research Promotion Agency. Finally, we want to thank Aaron Gibson, Georg Koppen, Harald Lampesberger, and Linus Nordberg for helpful feedback and suggestions.

References

[1] Alexa. *The top 500 sites on the web.* 2013. URL: http://www.alexa.com/topsites.

[2] Daniel J. Bernstein. "Curve25519: new Diffie-Hellman speed records". In: *Public Key Cryptography.* Springer, 2006. URL: http://cr.yp.to/ecdh/curve25519-20060209.pdf.

[3] Stevens Le Blond et al. "One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users". In: *LEET.* USENIX, 2011. URL: https://www.usenix.org/legacy/events/leet11/tech/full_papers/LeBlond.pdf.

[4] Stevens Le Blond et al. "Spying the World from Your Laptop: Identifying and Profiling Content Providers and Big Downloaders in BitTorrent". In: *LEET.* USENIX, 2010. URL: https://www.usenix.org/legacy/event/leet10/tech/full_papers/LeBlond.pdf.

[5] Sambuddho Chakravarty et al. "Detecting Traffic Snooping in Tor Using Decoys". In: *RAID.* Springer, 2011. URL: http://www.cs.columbia.edu/~mikepo/papers/tordecoys.raid11.pdf.

[6] *Crossbear*. URL: http://www.crossbear.org.

[7] Roger Dingledine. *Re: Holy shit I caught 1*. 2006. URL: http://archives.seul.org/or/talk/Aug-2006/msg00262.html.

[8] Roger Dingledine and Nick Mathewson. “Anonymity Loves Company: Usability and the Network Effect”. In: *WEIS*. 2006. URL: http://freehaven.net/doc/wupss04/usability.pdf.

[9] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. URL: https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. “Tor: The Second-Generation Onion Router”. In: *USENIX Security*. USENIX, 2004. URL: http://static.usenix.org/event/sec04/tech/full_papers/dingledine/dingledine.pdf.

[11] Electronic Frontier Foundation. *HTTPS Everywhere*. 2013. URL: https://www.eff.org/https-everywhere.

[12] Ian Goldberg. “On the Security of the Tor Authentication Protocol”. In: *PETS*. Springer, 2006. URL: http://freehaven.net/anonbib/cache/tap:pet2006.pdf.

[13] Dan Haim. *SocksiPy - A Python SOCKS client module*. 2006. URL: http://socksipy.sourceforge.net.

[14] Jeff Hodges, Collin Jackson, and Adam Barth. *HTTP Strict Transport Security (HSTS)*. 2012. URL: https://tools.ietf.org/html/rfc6797.

[15] Jeff Hodges, Collin Jackson, and Adam Barth. *RFC 6797: HTTP Strict Transport Security (HSTS)*. 2012. URL: https://tools.ietf.org/html/rfc6797.

[16] Markus Huber, Martin Mulazzani, and Edgar Weippl. “Tor HTTP Usage and Information Leakage”. In: *CMS*. Springer, 2010. URL: http://freehaven.net/anonbib/cache/huber2010tor.pdf.

[17] InformAction. *NoScript*. 2013. URL: http://noscript.net.

[18] *Known Bad Relays*. URL: https://trac.torproject.org/projects/tor/wiki/doc/badRelays.

[19] Thoughtcrime Labs. *Convergence*. 2011. URL: http://convergence.io.

[20] Adam Langley. *Public key pinning*. 2011. URL: https://www.imperialviolet.org/2011/05/04/pinning.html.

[21] Graham Lowe, Patrick Winters, and Michael L. Marcus. *The Great DNS Wall of China*. Tech. rep. New York University, 2007. URL: http://cs.nyu.edu/~pcw216/work/nds/final.pdf.

[22] Moxie Marlinspike. *sslstrip*. URL: http://www.thoughtcrime.org/software/sslstrip/.

[23] Moxie Marlinspike. *tortunnel*. URL: http://www.thoughtcrime.org/software/tortunnel/.

[24] Damon McCoy et al. “Shining Light in Dark Places: Understanding the Tor Network”. In: *PETS*. Springer, 2008. URL: http://homes.cs.washington.edu/~yoshi/papers/Tor/PETS2008_37.pdf.

[25] Ryan Paul. *Security expert used Tor to collect government e-mail passwords*. 2007. URL: http://arstechnica.com/security/2007/09/security-expert-used-tor-to-collect-government-e-mail-passwords/.

[26] Mike Perry, Erinn Clark, and Steven Murdoch. *The Design and Implementation of the Tor Browser [DRAFT]*. 2013. URL: https://www.torproject.org/projects/torbrowser/design/.

[27] SkullSecurity. *Passwords*. 2011. URL: https://wiki.skullsecurity.org/Passwords.

[28] SoftCom, Inc. *Email Hosting Services*. URL: http://mail2web.com.

[29] The Tor Project. *ExoneraTor*. URL: https://exonerator.torproject.org.

[30] The Tor Project. *Relays with Exit, Fast, Guard, Stable, and HSDir flags*. URL: https://metrics.torproject.org/network.html#relayflags.

[31] The Tor Project. *Snakes on a Tor*. URL: https://gitweb.torproject.org/torflow.git/tree/HEAD:/NetworkScanners/ExitAuthority.

[32] The Tor Project. *Stem Docs*. URL: https://stem.torproject.org.

[33] The Tor Project. *TC: A Tor control protocol (Version 1)*. URL: https://gitweb.torproject.org/torspec.git/blob/HEAD:/control-spec.txt.

[34] *TOR exit-node doing MITM attacks*. URL: http://www.teamfurry.com/wordpress/2007/11/20/tor-exit-node-doing-mitm-attacks.

[35] *torscanner*. URL: https://code.google.com/p/torscanner/.

[36] *Torsocks: use socks-friendly applications with Tor*. URL: https://code.google.com/p/torsocks/.

[37] Dan Wendlandt, David G. Andersen, and Adrian Perrig. "Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing". In: *ATC*. USENIX, 2008. URL: http://perspectivessecurity.files.wordpress.com/2011/07/perspectives_usenix08.pdf.

A Malicious X.509 Root Certificate

Below, the root certificate which was shared by all Russian and the single U.S. exit relay is shown. While the domain authority.com does exist as of May 2014, it appears to be unrelated to the CA "Main Authority", the issuer.

```
Certificate:
 Data:
  Version: 3 (0x2)
  Serial Number: 16517615612733694071 (0xe53a5be2bd702077)
 Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=US, ST=Nevada, L=Newbury, O=Main Authority,
    OU=Certificate Management,
    CN=main.authority.com/emailAddress=cert@authority.com
  Validity
   Not Before: Feb 12 08:13:07 2013 GMT
   Not After : Feb 10 08:13:07 2023 GMT
  Subject: C=US, ST=Nevada, L=Newbury, O=Main Authority,
     OU=Certificate Management,
     CN=main.authority.com/emailAddress=cert@authority.com
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
    Public-Key: (1024 bit)
    Modulus:
     00:da:5d:5f:06:06:dc:8e:f1:8c:70:b1:58:12:0a:
     41:0e:b9:23:cc:0e:6f:bc:22:5a:05:12:09:cf:ac:
     85:9d:95:2c:3a:93:5d:c9:04:c9:4e:72:15:6a:10:
     f1:b6:cd:e4:8e:ad:5a:7f:1e:d2:b5:a7:13:e9:87:
     d8:aa:a0:24:15:24:84:37:d1:69:8e:31:8f:5c:2e:
     92:e3:f4:9c:c3:bc:18:7d:cf:b7:ba:b2:5b:32:61:
     64:05:cd:1f:c3:b5:28:e1:f5:a5:1c:35:db:0f:e8:
     c3:1d:e3:e3:33:9c:95:61:6d:b7:a6:ad:de:2b:0d:
     d2:88:07:5f:63:0d:9c:1e:cf
    Exponent: 65537 (0x10001)
  X509v3 extensions:
   X509v3 Subject Key Identifier:
    07:42:E0:52:A7:DC:A5:C5:0F:C5:
    AF:03:56:CD:EB:42:8D:96:00:D6
   X509v3 Authority Key Identifier:
    keyid:07:42:E0:52:A7:DC:A5:C5:0F:C5:
          AF:03:56:CD:EB:42:8D:96:00:D6
    DirName:/C=US/ST=Nevada/L=Newbury/O=Main Authority
      /OU=Certificate Management
      /CN=main.authority.com/emailAddress=cert@authority.com
    serial:E5:3A:5B:E2:BD:70:20:77

   X509v3 Basic Constraints:
    CA:TRUE
 Signature Algorithm: sha1WithRSAEncryption
   23:55:73:1b:5c:77:e4:4b:14:d7:71:b4:09:11:4c:ed:2d:08:
   ae:7e:37:21:2e:a7:a0:49:6f:d1:9f:c8:21:77:76:55:71:f9:
   8c:7b:2c:e8:a9:ea:7f:2f:98:f7:45:44:52:b5:46:a4:09:4b:
   ce:88:90:bd:28:ed:05:8c:b6:14:79:a0:f3:d3:1f:30:d6:59:
   5c:dd:e6:e6:cd:3a:a4:69:8f:2d:0c:49:e7:df:01:52:b3:34:
   38:97:c5:9a:c3:fa:f3:61:b8:89:0f:d2:d9:a5:48:e6:7b:67:
   48:4a:72:3f:da:28:3e:65:bf:7a:c2:96:27:dd:c0:1a:ea:51:
   f5:09
```

B Injected HTML Code

The following HTML code was injected by the relay `5A2A51D4` (see Table 1 and Table 2). It was appended right in front of the closing HTML tag.

```
<br>
<img src="http://111.251.157.184/pics.cgi"
 width="1" height="1">
```

When requesting the image link inside the HTML code, the server responds with another HTML document. The full HTTP response is shown below.

```
HTTP/1.1 200 OK
Date: Tue, 14 Jan 2014 17:12:08 GMT
Server: Apache/2.2.22 (Ubuntu)
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html


<HTML>
<HEAD>
<TITLE>No Title</TITLE>
```

```
</HEAD>
<BODY>

</BODY>
</HTML>
```