Torinj : Automated Exploitation Malware Targeting Tor Users

Gérard Wagener
quuxlabs
gerard.wagener@quuxlabs.com

Alexandre Dulaunoy
quuxlabs
alexandre.dulaunoy@quuxlabs.com

Radu State
University of Luxembourg
radu.state@uni.lu

24 May 2009

Abstract

We propose in this paper a new propagation vector for malicious software by abusing the Tor network. Tor is particularly relevant, since operating a Tor exit node is easy and involves low costs compared to attack institutional or ISP networks. After presenting the Tor network from an attacker perspective, we describe an automated exploitation malware which is operated on a Tor exit node targeting to infect web browsers. Our experiments show that the current deployed Tor network, provides a large amount of potential victims.

1 Introduction

The ubiquitous computation and network infrastructure, currently deployed, is exposed to numerous risks. Recently the conficker worm, a self-spreading malicious software (malware) infected millions of machines [9]. Malware often uses multiple attack vectors. According to the authors of [9] the conficker worm can also propagate via network shares and USB sticks. Moreover some evil web pages infect visitors with malicious software [17]. Some users believe that Tor [6], an anonymous communication service, can help to mitigate against privacy and confidentiality attacks. Tor can be summarized as overlay network aiming to hide one's identity which is formally proved [7]. According to Eric Cronin [5] eavesdropping is a difficult task due to the fact that packets could be misinterpreted. An additional problem for attackers is to wiretap at a strategic point where multiple hosts can be sniffed. However, Tor simplifies traffic eavesdropping for an attacker. An attacker simply needs to install Tor exit nodes and participate in the Tor network. Another advantage for an attacker to use Tor, is its proven anonymity which is tempting to create a stealthy and anonymous command and control center for controlling the eavesdropping and the infection of machines.

In this paper, we propose a novel propagation mechanism of malicious software via Tor and the contributions of this paper are

- an estimation of the vulnerable browsers aiming to tune the web browser infection.
- a mechanism to enforce interactions with the web browsers aiming to distribute malicious payloads.

The remaining paper is organized as follows:

Section 2 describes related work and focus on potential attacks on Tor. An attacker incentive model is presented in section 3 which motivates the design and implementation of an automatic exploitation malware using the Tor network, shown in section 4. Section 5 concludes the article and announces future work activities.

2 Related work

The main purpose of Tor is to provide anonymous communication services. This is achieved by setting up an overlay network composed of entry guard nodes, relay nodes and exit nodes. A client that wants to use the Tor network connects to entry guard and then establish a circuit towards the exit nodes. In this circuit each node only knows its predecessor [6]. Profiling attacks on encrypted web proxy traffic were already studied by analyzing the exchanged number of bytes [8]. McCoy et al. studied Tor traffic [10]. They captured traffic at entry guards and exit nodes. Thus, they were able to study some clear text protocols like HTTP and telnet. The purpose of their study was to gain some insights about the Tor usage. In their study they could establish the number of different users passing through their entry guard, because they could see where they are coming from. However, when analyzing traffic from an exit node the traffic is already anonymised which makes it hard to distinguish users.

From that paper can be concluded that the most used protocol is HTTP. A threat model for the Tor network was proposed by [6] and [10]. An attacker can intercept some fraction of the traffic. She also can generate, modify, delay some traffic and can compromise a fraction of the Tor nodes. Roger Dingledine et al. described various attacks on the different Tor nodes [6] and McCoy et al. even present countermeasures to detect Tor exit nodes that are intercepting traffic [10]. Their major assumption is that the attacker is doing DNS reverse lookups in real time. Furthermore, efforts are done to wipe out sensitive information like user agents, cookies from HTTP requests. Privoxy [16] is a local proxy implementation that hides some sensitive information. An experiment, performed by Dan Egerstad [21], showed that a lot of Tor users transmit sensitive information, like account names, user names and passwords through the Tor network without an end to end encryption. Security improvements in the Tor network are described by Mike Perry [15] and especially in the area of application attacks at the exit nodes. One of the proposed improvement is to carefully distribute Tor exit nodes usage to use disjoint IP networks. Mike Perry also announced to compute checksums of carefully selected web pages in order to detect injection attacks.

3 Attacker Incentive Model

As discussed in section 2, attackers can easily eavesdrop traffic on a Tor exit node. In this paper we go a step further and propose an automated exploitation malware that is capable to infect browsers that pass through an exit node. An attacker should be able to estimate the population of vulnerable browsers and to enforce an interaction with the browsers.

3.1 Passive attacks

Besides the tools like Privoxy that try to wipe out most of this information some users still provide browser information like user agents and cookies. Many browsers set this string. Some browsers start this string by setting the browser's name followed with the version. Other browsers set the browser family first and then put the browser name between brackets. Furthermore, some browsers provide information about the underlying operating system and used libraries. This unorganized user agent naming provides us some insights about the users that are surfing via our exit node.

Furthermore, the Mitre organization hosts the Common Vulnerabilities and Exposure Database (CVE) which contains known software vulnerabilities from 1999 until now, including browser vulnerabilities. If we observe n browsers, V of them are vulnerable and for $\bar{V}$ no vulnerability was reported. Thus we can compute the browser vulnerability ratio b defined in eq. 1. If all observed user agents are vulnerable the browser vulnerability ratio becomes 1, and if no observed user agents are vulnerable b = 0.

$$b = \frac{V}{V + \bar{V}} \tag{1}$$

3.2 Active attacks

As previously described, the browser infection ratio can be computed. User agent strings can be forged.

Tools like Privoxy change user agent strings. Moreover proxies or browsers can be configured to not download external objects attached to a web site, where an attacker can place his infection payload dedicated for web browsers. Hence, a feedback from an observed HTTP traffic is desired.

An attacker can tag HTML responses for getting this feedback. Practically, an attacker can set up a man-in-the middle attack by installing a transparent proxy on the Tor exit node.

She can introduce n images or other objects in intercepted HTML documents. In case a regular browser is parsing these pages it tries to acquire the objects. Normally the URL of the object is first resolved followed by the download of the object.

We define a tag as an object that is injected in the intercepted HTML traffic and we propose two tags per HTML response.

3.2.1 Static tag injection

A static tag is a fix invisible image that is introduced in HTTP responses. The image has a dimension of 1 to 1 pixel and is invisible aiming to not distract the user looking at the HTML page. The URL of the image is fix for all users. We assume that the DNS cache on the user's machine is working correctly and that the lookup of the image domain name is only done once while the user is surfing. Thus we can count the number of different users.

3.2.2 Dynamic tag injection

The purpose of the dynamic tag is to enforce an interaction for each visited web page. In case only a static image is used, the image is normally resolved once and then it is kept in cache for all the next web pages that are visited during the life time of the browser. In order to avoid this caching mechanism an attacker can generate a unique sub domain for each injected dynamic image. Thus the machine hosting the browser is forced to do a DNS lookup. An attacker can also observe if a user comes back. In this case the user restarted her machine, reloaded her browser with a dedicated web page. In that web page, the attacker previously injected an image located on a unique sub domain. Hence, if the attacker sees more than one hit for a unique generated sub domain, she can deduce that the same user reappeared.

3.3 Attacker Information sources

By intercepting and tagging HTML documents an attacker can explore three information sources.

DNS server We assume that an attacker controls a DNS server for generating unique sub-domains for each dynamic tag. The attacker can log all the DNS queries including source IP addresses that do the DNS queries.

Web proxy The tag injection can be done by doing a man-in-the middle attack. An attacker can compromise an exit node and set up a transparent proxy for inserting the tags. From this web proxy the attacker can record all the HTTP header information like user agents or cookies.

TCP traffic After having compromised an exit node an attacker can also record all the out going traffic from the exit code. Thus she has access to the full communications of Tor users. The attacker can focus on HTTP responses, especially on the mime type of a message, aiming to tune her browser infection. For instance, if she notices that most HTTP responses are HTML documents, she could inject images in the transferred HTML documents. However, if she sees that the most transfered documents are PDF files she could launch PDF attacks.

4 Torinj : An Automated Exploitation Malware

To validate the attacker incentive model we implemented a proof of concept malware called Torinj. Torinj is composed of three components : an unmodified Tor client, an embedded intercepting proxy and a hidden C&C (command and control) channel. An overview is shown in figure 1. A standard and unmodified Tor client is integrated with Torinj providing the access to the Tor network layer. Torinj

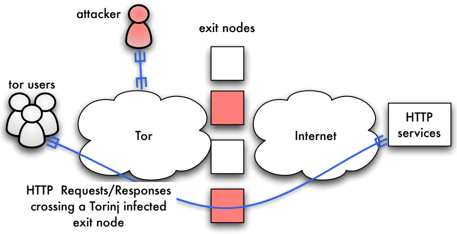
attacker
exit nodes
tor users
Tor
Internet
HTTP services
HTTP Requests/Responses crossing a Torinj infected exit node

Figure 1: An overview of the Torinj framework

behaves like any other Tor client and provides similar services like relay or exit functionalities. Torinj includes a small HTTP proxy used to intercept and relay HTTP requests. Interception and relaying are activated by the attacker using the C&C channel. The hidden C&C channel relies on hidden service protocol [20] available in Tor to provide some anonymity [13] to the command and control interface and its user. The attacker access the C&C channel of each Torinj bot through the Tor network. Torinj infection is working at the interception level and does not need to lure the users to connect to attractive services. Torinj infection is done on the unencrypted HTTP requests/responses crossing the infected exit node. The exploitation mechanism of Torinj is composed of two steps:

passive attacks, where the Torinj HTTP proxy is gathering essential information about the HTTP requests (e.g. browser user agent, Internet media type) without altering the requests;

active attacks, where Torinj is exploiting the HTTP requests by modifying the responses based on the optimal infection scenario learned by the previous step.

For further technical details we recommend to read our source code[1]

4.1 Experiment setup

During this experiment we used three different machines. On the first machine (M_1) we operated an unmodified Tor exit node (v0.2.1.14-rc). On the second machine we let run BIND [2], version 9.4.2, as DNS server and used the tool tcpdump [19] to capture all the DNS queries and responses. On the third machine (M_3) we operated an apache web server [1], version 2.2.6, hosting the transparent image simulating a malicious payload. From a legal and ethical point of view we avoided to inject malicious java script payloads like XSS-proxy or BeEF [3].

All the machines were synchronized with NTP [11] in order to have accurate timestamps. After having started to participate in the Tor network, we set up a web proxy implemented in Perl from CPAN [4] (0.23). This proxy was extended to inject tags with regular expressions. We used the tool iptables [12] to reroute the traffic, originated from the Tor exit node to the Internet, to our Perl proxy server. The DNS server was configured with a wild card that it should associate all sub-domains with the IP address of our web server. Thus inside the web proxy we can generate dynamic and static tags that always point to our web server. As information sources we used tcpdump activated on M_1 and M_2, the web server logs, the web proxy logs. The processing was done with Perl and sqlite3 [14] and a modified version of tcpick [18].

4.2 Passive attacks

We operated a Tor exit node for a period of 28 hours and we passively inspected observed HTTP headers. In this experiments we observed similar results to McCoy et al [10]. We observed that 96% of the traffic was HTTP and only 4% of the traffic was end-to-end encrypted with HTTPS.

We have also discovered 4973 different user agent strings which confirms the non existence of naming convention for user agents. We have found that only 3.2% of the HTTP requests did not have a user agent set. We assume that these browsers are not vulnerable despite they could be vulnerable versions. Moreover we did an automatic lookup of the user agent in the CVE list. Although 1845 user agents did not match an entry in the CVE list (37% of the browsers), there may be undisclosed vulnerabilities. If a version is not explicitly set for a given browser, we assume that this browser is not vul-

[1] `http://www.foo.be/torinj/`

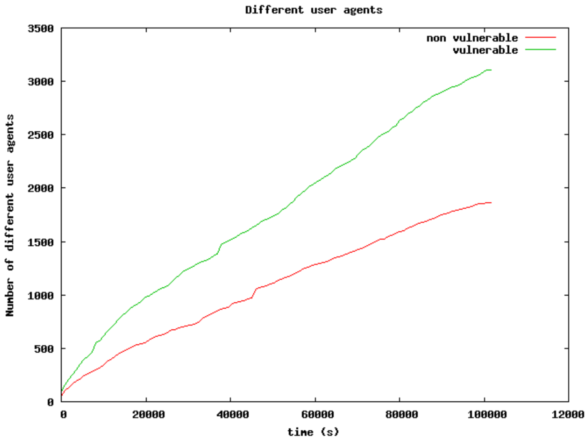

Figure 2: Vulnerable and non vulnerable user agents

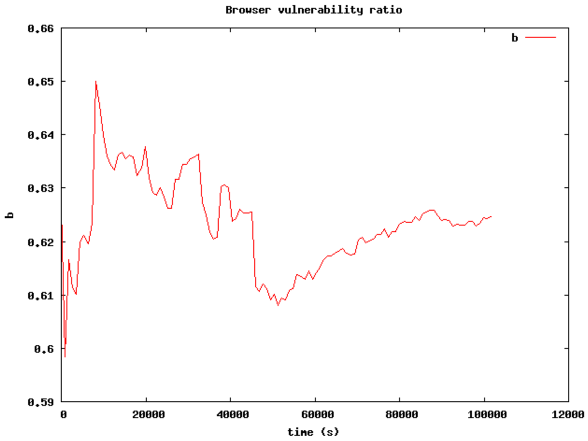

Figure 3: Browser vulnerability ratio

nerable (1.3% of the observed browsers). We found 3106 vulnerable unique user agents. Figure 2 confirms the fact that there are more vulnerable browsers than non vulnerable browsers. We measured during time intervals of 15 minutes the number of vulnerable browsers and non vulnerable browsers. The number of unique user agent strings is growing (figure 2) because most user agent strings contain version numbers, with which other browsers they are compatible, information about the underlying operating system, patch levels and used libraries. Figure 3 presents the browser vulnerability ratio, which is varying around 0.63. That means that on average 63 browsers of 100 are vulnerable according the CVE list which shows the potential of our automated exploitation malware.

4.3 Active attacks

For this experiment we set up and operated the automated exploitation malware proof of concept for two and a half hours. We have observed 391 different user agents that passed through our proxy. The proxy injected 126 static tags and 688 dynamic tags.

The purpose of the static tag is to count the different users. If a user opens her browser, the later resolves the static tag, the static tag is then kept in the user's cache and it should not be resolved again. On our DNS server we observed 126 hits for the static image and on our web server we counted 196 hits. Most of the hits on the web server were done through our proxy. However 80 of the web server hits passed trough other Tor exit nodes. When the injected tags are downloaded through our proxy, our proxy did not tag these responses again. We also have counted 391 different user agent strings. This number is higher than the number of static tag injection hits and the ratio corresponds to 32% which can be explained that only HTML documents were tagged and other mime types were directly forwarded without change.

Each HTML document having a body element is intercepted and a unique dynamic tag is injected and our proxy injected 688 tags.

4.3.1 Mime type distribution

In order to get a feedback from a user the injected tag needs to be processed by the user agent and the user agent needs to connect back to the attacker. This is often the case when HTML documents are processed. Table 1 shows the mime type distribution. Roughly a third of the traffic that goes through the proxy is composed of HTML documents.

Mime type	%
text/html	33
image/jpeg	24
image/gif	16
image/png	06
text/plain	05
Content-Type:application/x-javascript	04
Content-Type:text/css	03
Content-Type:text/javascript	03
Content-Type:text/xml	02
Others:	06

Table 1: Mime type distribution

5 Conclusion and future work

In this paper, we have described the incentive for an attacker to compromise Tor exit nodes and designed the Torinj scenario targetting the HTTP protocol. The experiments further demonstrate the viability of the Torinj prototype and the inherent interest for an attacker to compromise Tor exit nodes. Our experiments showed that 63% of the browser passing through an exit node are vulnerable according the CVE database. Moreover, we showed that interaction with the browsers can be induced by injecting tags in HTML documents. By injecting tags in HTML documents an interaction per web-page can be enforced which is necessary of malicious payload distribution. However, additional research efforts are needed to complete this proof of concept. First of all, the automated exploitation malware should be operated over a longer period of time and from different IP addresses. We already facilitated this work by making our exploitation software freely available under a GPL license. Furthermore, user agents can be carefully crafted to trick the exploitation malware. Therefore other browser finger printing techniques should be explored. We have only tested the injection in HTML documents and other mime types, like PDF, images, movies can be explored. We are also planning to improve the infection model to find effective strategies for the attacker to launch automatic infection while limiting the detection factor.

References

[1] Apache http server project. http://httpd.apache.org/.

[2] Bind. https://www.isc.org/software/bind/.

[3] Rich Cannings, Himanshu Dwivedi, and Zane Lackey. *Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions (Hacking Exposed).* McGraw-Hill Osborne Media, 2007.

[4] Cpan http::proxy. http://search.cpan.org/dist/HTTP-Proxy/.

[5] Eric Cronin, Micah Sherr, and Matt Blaze. On the reliability of current generation network eavesdropping tools. *International Federation for Information Processings*, 222(2):103–113, 2008.

[6] Roger Dingledine and Nick. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, San Diego, CA, USA, August 2004.

[7] Joan Feigenbaum, Aaron Johnson, and Paul F. Syverson. A model of onion routing with provable anonymity. In *Financial Cryptography*, pages 57–71, 2007.

[8] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pages 171–178, 2002.

[9] Werner Tillman Leder Felix. Know your enemy: Containing conficker to tame a malware. `http://www.honeynet.org/papers/conficker`. Last accessed 18 May 2009.

[10] Damon Mccoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In *PETS '08: Proceedings of the 8th international symposium on Privacy Enhancing Technologies*, pages 63–76, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] David L. Mills. Network time protocol (version 3) specification, implementation and analysis, 1992. RFC.

[12] netfilter. iptables. http://www.iptables.org/.

[13] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.

[14] Michael Owens. Embedding an sql database with sqlite. *Linux J.*, 2003(110):2, 2003.

[15] Mike Perry. Securing the tor network. `http://www.freehaven.net/~arma/SecuringTheTorNetwork.pdf`, 2007.

[16] Privoxy. http://www.privoxy.org/.

[17] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iframes point to us. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[18] Francesco Stablum. Tcpick. http://tcpick.sourceforge.net/.

[19] tcpdump. http://www.tcpdump.org/.

[20] Tor: Hidden service protocol. `http://www.torproject.org/hidden-services.html.en`.

[21] Kim Zetter. Tor researcher who exposed embassy e-mail passwords gets raided by swedish fbi and cia. http://blog.wired.com/27bstroke6/2007/11/swedish-researc.html, November 2007.