2010 Fourth International Conference on Network and System Security

Breaking Tor Anonymity with Game Theory and Data Mining

Cynthia Wagner, Gerard Wagener, Radu State, Thomas Engel
University of Luxembourg
FSTC, SECAN-Lab
Luxembourg
Email: {cynthia.wagner, gerard.wagener, radu.state, thomas.engel}@uni.lu

Alexandre Dulaunoy
SES-S.A.
Luxembourg
Email: a@foo.be

***Abstract*—Attacking anonymous communication networks is very tempting and many attacks have already been observed. We consider the case of Tor, a widely-used anonymous overlay network. Despite the deployment of several protection mechanisms, we propose an attack originated from only one rogue exit node. Our attack is composed of two elements. The first is an active tag injection scheme. The malicious exit node injects image tags into all HTTP replies, which will be cached for upcoming requests and allows different users to be distinguished. The second element is an inference attack that leverages a semi-supervised learning algorithm to reconstruct browsing sessions. Captured traffic flows are clustered into sessions, such that one session is most probably associated to a specific user. The clustering algorithm uses HTTP headers and logical dependencies encountered in a browsing session. We have implemented a prototype and evaluated its performance on the Tor network. The article also describes several counter-measures and advanced attacks, modeled in a game-theoretical framework and their relevancy assessed with reference to the Nash equilibrium.**

I. Introduction

Anonymous communication systems have emerged as potential solutions to provide privacy and anonymous web access. The main idea behind most of such systems is an overlay network used to mix traffic and so, defeat attacks based traffic correlation analysis. Tor [7] is the best known approach due to its large deployment base and proven security services. Several prior articles [6], [19], [14], [32], [18], [3], [28], [29], [30] have shown Tor's vulnerability to a class of disclosure attacks which detect whether one host is communicating with another. In most research, the threat model assumes that a malicious entity can control either a Tor entry node or a pair (entry, exit) of nodes. In some cases, attackers inject data or tags into user-related flows to trigger information disclosure [24]. Several protection mechanisms have been proposed, like probing exit nodes [26] or forging browser-related information.

This article presents two main contributions, a Data Mining driven solution to recover the browsing history of Tor users and optimal configuration settings based on game theory for Tor users and operators, as well taking malicious nodes into account.

Following the discussion in [21], only trusted entry nodes can be selected in a Tor network. Therefore, we propose a proof of concept malware called *Torinj*, targeted to infect trusted Tor nodes with the aim of reconstructing overall network activity. We aim to detect the number of distinct users and to reconstruct the browsing history for each user. Our malware *Torinj* is targeted against Tor exit nodes because, we expect a larger vulnerable population of exit nodes than entry nodes and expect them less protected. The added value of *Torinj* is its ability to recover a user browsing history even when a trusted entry node is used. We do not specifically address the disclosure of users' identities or locations, but focus on extracting user-specific behaviors. We describe a practical attack that leverages a semi-supervised learning algorithm and a simple traffic injection scheme for this purpose. Several defensive measures exist, like enhancing user privacy by manipulating user agent fields or by testing the exit node. These actions can be countered by an attacker — she can become more stealthy to capture and reconstruct as many sessions as possible. We will assess the efficiency of these strategies with the aid of game-theoretical concepts.

The article is structured as follows: In section II we present a global view of our architecture with related attack scenarios. We introduce some background material and in section III we detail the semi-supervised clustering algorithm of Tor flows. Section IV describes some defense and advanced attack strategies for Tor. Section V presents the experimental results for our approach. Section VI discusses related work and section VII summarises the article.

II. Data mining a Tor session

It is challenging to reconstruct Tor sessions when only one exit node can be controlled. The attacker can observe pairs of outgoing requests and incoming replies but is unable to differentiate individual sessions. Thus, anonymous browsing under this threat model is analogous to hiding in the crowd. Consequently, it is natural to aim to isolate individual browsing histories and build individual user-related trace histories. We describe a new attack (see Fig.1) that leverages a Data Mining technique together with aspects of the HTTP protocol to cluster and extract individual HTTP sessions relayed over a malicious exit node. In this analysis, we consider only Web traffic. Several users (in the example, we consider only 2 users) use 2 Tor entry nodes which build tunnels ending at an exit node. We assume this exit node controlled by an attacker, so she can observe the traffic (5 flows) between the exit node and the 3 servers. The objective of the attack is, first to establish that only 2 users (`A` and `B`) are currently routed through the exit nodes and secondly to reconstruct the browsing history for each of them. In this case, $flow_{A1}$, $flow_{A2}$ and $flow_{A3}$ are used for user `A`, while $flow_{B1}$

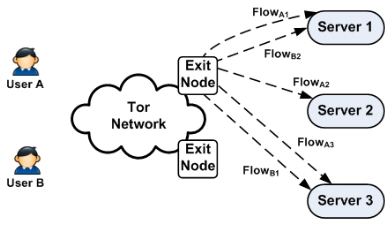


Fig. 1. Typical attack scenario

and $flow_{B2}$ are tagged for user B. The practical attack (see Fig.2) on Tor is executed as follows,

1) User A issues one (or several) HTTP request(s) to *server 1*. This traffic is contained in one flow: $flow_{A1}$. The malicious exit node forwards the request(s) to *server 1* and performs a static tag injection. The static tag is a fixed invisible image that is introduced in HTTP responses. The image has a size of 1x1 pixel and is invisible, aiming to not distract the user while browsing the HTML page. The image URL is unique for every injection. This is done by generating a universally unique number that is used to name the image file. We assume that the browser cache on the user's machine is working correctly and that the image download is done only once. The file containing the image is hosted on an attacker controlled server, called *tagging server*. For illustrating, we assume the server URL to be *www.taggingserver.com*.
2) The browser of user A requests the tag by connecting to the tagging server to download it. Even if another Tor exit node is used to retrieve the tag, the incoming traffic ($flow_{A2}$) is intercepted by the attacker. This step in the attack (see Fig.3) is essential to logically link $flow_{A2}$ and $flow_{A1}$. The attacker uses the image URL to link the initial injection performed on the flow $flow_{A1}$ with the incoming request. The tagging server replies with a redirection URI *www.taggingserver.com/static.png* — which is a static URL. This reply is provided via the *301 Moved Permanently* HTTP response code [13]. It will also set a unique cookie for the domain *www.taggingserver.com* if no cookie was provided in the request. The browser of user A will retrieve the file *static.png* and cache the new URI, which is the same for all replies performed in this stage. The advantage of this redirection is, when an URI is retrieved via *301 Moved Permanently*, its response codes are locally cached by the browser. Setting the cookie depends on the browser's capability to accept third party cookies, but it can also work without setting the cookie, if an additional synchronization step between the tagging server and the exit node is performed. This step is needed to associate the flow used to download the image tag with the initial request/response. Obviously, these cookies are not taken into account when clustering the traffic. This injected traffic is not considered when assessing the performance of the Data Mining algorithm.
3) User A issues one (or several) HTTP request(s) to *server 2*, contained in $flow_{A3}$. The malicious exit node forwards the request(s) to *server 2* and performs a tag injection. The tagging server replies with a redirection URI *www.taggingserver.com/static.png*. In this redirection, no cookie is set for the domain *www.taggingserver.com* because the request already includes the cookie received in the previous step. This reply is provided via the *301 Moved Permanently* HTTP response code. However, the browser of user A will not retrieve the file *static.png* because it is already cached locally — the user agent will use the locally-cached version instead. Due to the missing download and the provided cookie, the attacker learns that the incoming traffic and the associated flow $flow_{A3}$, are related to a user that has already been observed.
4) User B issues one (or several) HTTP request(s) to *server 1*. This traffic is contained in $flow_{B1}$. The malicious exit node forwards the request(s) to *server 1* and performs a tag injection.
5) The browser of user B requests the static tag by connecting to the tagging server to download it. The incoming traffic ($flow_{B2}$) is intercepted by the attacker. The processing is similar to the previous case. Here, a download of *www.taggingserver.com/static.png* is performed and no cookie for the domain *www.taggingserver.com* is included, the attacker can infer that $flow_{B1}$ and $flow_{B2}$ are associated with a user that has not been observed yet.
6) Both users (A and B) continue their browsing sessions and will be tracked.

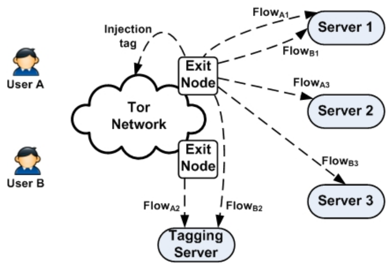


Fig. 2. Injection Attack

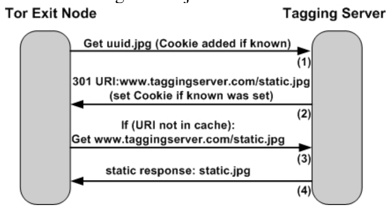

Tor Exit Node
Tagging Server
Get uuid.jpg (Cookie added if known)
(1)
301 URI:www.taggingserver.com/static.jpg
(set Cookie if known was set)
(2)
If (URI not in cache):
Get www.taggingserver.com/static.jpg
(3)
static response: static.jpg
(4)

Fig. 3. Flow A_2 interaction details

The key idea behind this attack is to have a set of flows for which we definitely know that the associated users are different: these are the flows targeted at the tagging server, from which the 1x1 image is downloaded and the cookies are issued. Additionally, we assume that connections by one user will share a set of features despite the Tor infrastructure. Such features relate to user agent-specific settings (e.g. accepted language, version, name), and to outgoing

HTTP requests (cookies, destination URIs). At last we suppose that users follow a normal browsing behavior — i.e. after retrieving a HTML page, the next connection will be targeted towards an URI which is included in the initial web page. The problem that has to be solve now is: *we need to cluster a large number of data where we are certain that a small subset belongs to different clusters*. We try to solve this problem by calculating a relevant distance function defined over pairs of flows. This function considers content-related similarities, user agent-specific settings and HTTP-specific protocol elements. A semi-supervised clustering algorithm takes this distance into account when performing the clustering.

III. Semi-supervised Clustering Algorithm

Semi-supervised learning approaches focus on data sets with only a small amount of labeled data and many unlabeled data samples. Methods such as *k-means* [4], a traditional clustering method, assume that adjacent data samples tend to have similar labels with the result that they propagate their labels to unlabeled data samples.

Our algorithm is based on the idea of the semi-supervised clustering algorithm [33], known as a *label propagation* (Fig.4) algorithm with few minor changes. The main idea is to construct a fully-connected graph, where only some nodes are labeled. Each label represents the class name of the node. We assume C different classes. Each class is initially represented by a flow that was tapped at the tagging server. The edges between two nodes have associated weights, which depend on the distance between two nodes and are controlled by the weight factor σ that is useful for the label propagation quality. All labels are propagated iteratively to unlabeled regions and nodes are either labeled or unlabeled data samples. Unlabeled nodes are iteratively estimated class-fellowship probabilities that give the probability that an item is associated with a class. At the end of the iterations, each unlabeled node is allocated to the most probable class, that is the class with highest class-fellowship score. At the beginning there are only a few flows for which the user is known. These are the flows that have been used to retrieve the image file from the tagging server and where cookies have been set. The aim is to classify the remaining flows by clustering them on a per user class base. This is done by the clustering algorithm, which uses a distance function between pairs of flows to compare them.

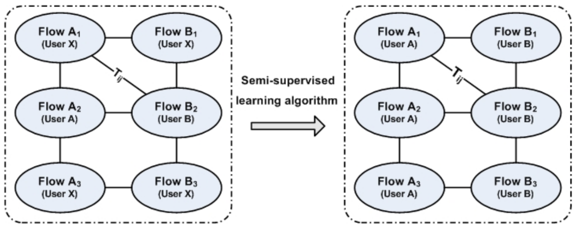


Fig. 4. Semi-supervised learning algorithm

We have a set R^D composed of labeled data (x_1, y_1),...,(x_l, y_l) and unlabeled data (x_{l+1}, y_{l+1}),..., (x_{l+u}, y_{l+u}) with $l{\ll}u$, where Y_L=$\{y_1, ..., y_l\}$ are the class labels of the labeled data and Y_U=$\{y_{l+1}, ..., y_{l+u}\}$ yet to be observed. D is the dimension of the input space. We assume that the number of classes C is known and that all classes are represented in the labeled data samples [33]. Let X=$\{x_1, ..., x_{l+u}\}$ be the different flows, and let us assume that each x is a D-dimensional vector, that is $x_i{\in}R^D$. We now want to estimate the class labels of the unlabeled samples Y_U from the data items X and their class labels Y_L.

The various distances used in this algorithm have to be defined. We assume two flow-class tuples (x_i, y_i) and (x_j, y_j) which have to be compared. In this case y_i or y_j represent the respective class labels and $D = 3$. To calculate the distances, the flow x_i is decomposed into a set of requests and responses, $Flow_i$=$\{(req_{i1}$,$rep_{i1})$,...,$(req_{ik}$, $rep_{ik})\}$. The same is done for flow x_j, $Flow_j$=$\{(req_{j1}$,$rep_{j1})$,...,$(req_{jl}$, $rep_{jl})\}$.

We define $d(i, j)$ to be the distance between the two pairs (req_{ip}, rep_{ip}) with p= (1 to n) and (req_{jq}, rep_{jq}) with q= (1 to m).

$$\begin{aligned} d_{ij} = d(Flow_i, Flow_j) = d_1(req_{ip}, req_{jq}) \\ +d_2(req_{ip}, rep_{jq}) + d_2(req_{jq}, rep_{ip}) \end{aligned} \tag{1}$$

where $d_1(req_{ip}$,$req_{jq})$ is the Levenshtein[1] distance d_L of particular parameters, as the field content of *Cookie* (Co), URL, User agent (UA) and Accepted Languages field (AL).

$$\begin{aligned} d_1(req_{ip}, req_{jq}) = d_L\Big((req_{ip[Co]}, req_{jq[Co]}) \\ +d_L(req_{ip[URL]}, req_{jq[URL]}) + d_L(req_{ip[UA]}, req_{jq[UA]}) \\ +d_L(req_{ip[AL]}, req_{jq[AL]})\Big) \end{aligned} \tag{2}$$

Equation (2) calculates the Levenshtein distance between the requests ip and jq, which is the sum of the numerical distances of the Co, UA and AL fields.

Distance $d_2(req_{ip}, rep_{jq})$ reflects whether or not a requested URL from ip is contained in reply jp,

$$d_2(req_{ip}, rep_{jq}) = \begin{cases} \alpha & \text{if } req_{ip[URL]} \text{ substring of } req_{jq} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

with $\alpha{\in}\mathbb{N}$, and α=1. α can be selected in order to tune the algorithm and considers the weight given to a logical link of browsing activity. The formula for $d_2(req_{ip}, rep_{jp})$ can also be applied for distance $d_2(req_{jq}, rep_{ip})$, to see whether or not a requested URL from jp is in reply ip.

The labeled and unlabeled data samples are represented in a fully-connected graph, where the edge between nodes i, j is weighted. To calculate the edge weight w_{ij}, we refer to the distances and estimated weights are scaled by σ, a function width scaling parameter.

$$w_{ij} = exp(-\frac{d_{ij}^2}{\sigma^2}) = exp(-\frac{\sum_{d=1}^{D}(x_i^d + x_j^d)^2}{\sigma^2}) \tag{4}$$

Node labels are propagated through the edges to all other nodes. As in [33], we define a $(l + u){\times}(l + u)$ transition matrix T, where T_{ij} gives the probability of a transition from node i to j.

$$T_{ij} = P(i \rightarrow j) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{jk}} \tag{5}$$

[1]Levenshtein distance [9]: Edit distance for measuring the difference between two strings or how many operations are needed to change one string into another.

We define a $(l+u)\times C$ label matrix Y, where a row reflects the label probability distribution of a node. For instance, the element Y_{ic} is the probability Y_i for flow i belonging to class $c\in C$. Initially these probabilities are initialized with $1/C$ for the unlabeled data samples.

The label propagation algorithm as it is used by [33] has three steps.

1) *Propagate $Y \leftarrow TY$. All nodes propagate their labels.*
2) *Row normalization of Y. This maintains a probability distribution.*
3) *Clamping of labeled data. We clamp the label distribution of labeled data to Y_{ic}=1, if item Y_i had an initial label of c. This step assures that initial labels are maintained.*

These steps are repeated until Y converges. It has been shown in [33] that the label propagation algorithm always converges.

To evaluate the labeling algorithm, we use class sensitivity, class specificity and the number of correct predictions [2]. In a multi-class prediction problem with C classes, a $C\times C$ contingency matrix Z=z_{ij} is used, where z_{ij} gives the number of times a sample belonging to class i is put in class j.

Class sensitivity gives the value of correctly predicted samples belonging to class i to x_i=$\sum_j z_{ij}$, the total number of samples associated to class i [2].

$$Q_i = \frac{z_{ii}}{x_i} \tag{6}$$

Average class sensitivity ϕ scales the sensitivity for all classes according to the number of classes C.

$$\phi = \frac{\sum_1^c Q_i}{C} \tag{7}$$

Class specificity gives the ratio of correctly-predicted samples in class i to y_i=$\sum_j z_{ji}$, the total number of samples predicted to be in class i [2].

$$Q_i = \frac{z_{ii}}{y_i} \tag{8}$$

Average class specificity ψ scales the specificity of all classes by the number of classes C.

$$\psi = \frac{\sum_1^C Q_i}{C} \tag{9}$$

The final evaluation parameter is the quality Q_{Total}, the value of all correct predictions made.

$$Q_{Total} = \frac{\sum_i z_{ii}}{N} \tag{10}$$

where N = $\sum_{ij}$ z_{ij} = $\sum_i$ x_i = $\sum_i$ y_i.

IV. Defense and attack strategies

A Tor user can deploy two main types of defensive measures to counter an injection attack. These can detect the malicious node and announce it as a rogue in the global Tor Directory. For this, a user connects to a known web server and retrieves one particular web page. A hash function checks whether the reply has been tampered with. A previously-stored hash value is compared with the current one, if a mismatch is detected, then the injection is revealed. This idea was proposed in [22] in the wider context of security improvements for the Tor network. Another proposal consisted in carefully distributing Tor exit nodes usage so as to use disjoint IP networks. We have not considered the latter proposal in the current game. Regarding the case of bad Tor exit nodes, Torscanner [26] is in use. The current process for adding a `BadExit` flag is done by the authorities managing the Tor directories, but up to now, we have not found a router providing a `BadExit` flag.

Once a rogue node has been announced, no other user will use it as an exit node and thus, for an attacker the game is literally over. The attacker can however minimize the probability of being detected by performing fewer injections to improve her stealth, i.e. instead of injecting the image tag into all HTTP replies, the attacker can use a probabilistic scheme and tag only a subset, i.e. injecting into 30 to 10% of the replies only. Thus, we deduce that the probability of being detected is indirectly proportional to the injection probability.

Another set of measures directly attack the Data Mining approach. Users can manipulate the HTTP headers that disclose information about their browser. The user agent field can be spoofed or completely removed. Privoxy[2] allows the specification of a list of user agents that can be used for this purpose. Another header that can be changed or removed is the accepted languages, but it is not reasonable to assume a random choice for this purpose, since popular web sites use this field to provide customized content/layout. For example, English speaking users hardly access the Russian website of Google, except in order to remain anonymous.

We model the game between an *attacker* and *Tor users* using the definitions proposed in [10]. Let a 3-tuple $\Gamma = (N, (A_i, R_i)_{1\leq i\leq n})$ be the game between *Tor users* and the *attacker*, where

- N is a set of n players
- A_i is a finite strategy set $(a_i \in A_n)$
- R_i: A$\rightarrow\mathbb{R}$ is payoff function, where $A = A_1\times ...\times A_n$

The game has two players, N=$\{$*Tor user*, *attacker*$\}$. All *Tor user*s act as one single collective player. The set A_1 corresponds to the *Tor user* actions and A_2 to the *attacker*'s actions. We define the *Tor user* strategy as a set of actions $A_1 = \{a_{1,0}, ..., a_{1,3}\}$, where

- $a_{1,0}$: Every user works normally — the web client is used without any privacy protection mechanisms
- $a_{1,1}$: Everybody modifies the user agent — for each request, a user agent is randomly selected and spoofed in the requests
- $a_{1,2}$: Everybody deletes user agent — no information whatever is leaked to the attacker
- $a_{1,3}$: Everybody deletes accepted languages — no language-specific information is leaked

The *attacker* strategy is defined as a set of vectors A_2=$\{(a_{2,\theta})\}$ where θ=$\{$0.1,0.2,...,0.5$\}$ is the injection probability. An attacker plays by choosing the probability of injecting a frame into a given HTTP reply.

One purpose of game theory is to find the optimal strategy profiles for players, resulting in the computation of a Nash Equilibrium that leads either to a pure or a mixed strategy. According to [10], the set of probability distributions over a strategy set A_i is a mixed strategy set

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

for a player i.

$$\Delta(A_i) = \left\{ q_i : A_i \rightarrow [0,1] \mid \sum_{a_i \in A_i} q_i(a_i) = 1 \right\} \qquad (11)$$

where $\Delta(A_i) \equiv Q_i$, $Q = \prod_i Q_i$.

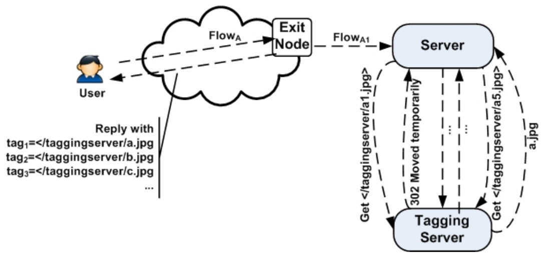
The expected payoffs for a player i from a strategy profile q are $\mathbb{E}_{a \sim q} = \sum_{a \in A} q(a) R_i(a)$, such that $q(a) = \prod_{i=1}^{N} q_j(a_j)$ iff a Nash equilibrium results from $\forall q_i \in Q_i, \mathbb{E}[R_i(q_i^*, q_{-i}] \geq \mathbb{E}[R_i(q_i, q_{-i})]$.

A Nash equilibrium in the context of a *Tor system* game means that neither the *Tor user* nor the *attacker* can increase their expected payoffs, assuming that neither player changes her strategy during the game.

Computing payoffs: The payoff for each player should model her gain as well as possible. Two different goals are important for an attacker. First, she should reconstruct sessions as accurately as possible by maximizing Q_{Total}. Secondly, she should remain as stealthy as possible and thus perform only least possible tagging. Previously we have shown that the more an attacker tags, the higher the probability of being detected. The payoff function for the attacker, p_a, can be defined as, p_a = (1-θ)$\times Q_{Total}$. For the *Tor user*, the payoff p_u is a measure of the achieved privacy and can be defined as, p_u = (1 -Q_{Total}).

A. Advanced attacks

When a Tor user deploys a user agent-changing strategy, the impact on the clustering is immediate. Due to the user agent field, the distance component will be biased and members belonging to the same class will be miss-classified due to large distances. Thus, for an attacker, it is natural to learn the list of user agents used by an individual user. This would bias the distance function towards weighting out large differences in the user agent field. There is a straightforward extension that an attacker can perform which leverages the processing of HTTP error messages. According to the HTTP specification [13], a server can reply with a *HTTP Error 302 - Moved temporarily* error message, that includes an alternative URL to which the redirection should occur. The web browser immediately retries the alternative URL. If this happens, the user agent field in the request is changed.



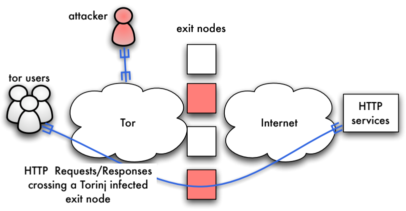
Flow$_A$
Exit Node
Flow$_{A1}$
Server
User
Reply with
tag$_1$=</taggingserver/a.jpg
tag$_2$=</taggingserver/b.jpg
tag$_3$=</taggingserver/c.jpg
...
Get </taggingserver/a1.jpg>
302 Moved temporarily
Get </taggingserver/a5.jpg>
a.jpg
Tagging Server

Fig. 5. Tagging server attack with HTTP Error message

From an attacker perspective this can be used to discover the list of user agents in use by a user. Similar to the basic attack, an image tag (i.e. *www.taggingserver.com/uuid.jpg*) is injected in a reply that is being relied by the exit node. The user's browser does not have the image in the cache and will connect to the tagging server to retrieve it. For this, the client spoofs the user agent header by using one of the values from its list. The tagging server receives a request for the resource *uuid.jpg*. Instead of sending back the corresponding file, the server will generate a universally unique identifier *uuid1*, reply with a *HTTP Error 302 Moved temporarily* message and provide the alternative URL: *www.taggingserver.com/uuid1.jpg*. Thus, the client will contact the tagging server *www.taggingserver.com* and ask for *uuid1.jpg*. For this request, another value for the user agent list is used and hence disclosed. The tagging server replies with one more *HTTP Error 302 Moved temporarily* error and an additional alternative URL: *www.taggingserver.com/uuid2.jpg*. This process can be repeated up to 5 times without detection by the user. The fifth request can be answered by sending back the image file. The use of a universally unique identifier is required to differentiate user-initiated requests. Obviously the file *uuidx.jpg* has not to exist in real and *x* is used by the tagging server to count *HTTP Error 302 Moved temporarily* messages that have been sent back already.

To differentiate between new users and users already observed, the final download of the image file and a cookie tracking mechanism is required. New users will download the file and will have a cookie set, while previously observed users will rely on the locally-cached data and will use the domain cookie in all requests (see Fig.5). In the end, the attacker has learned five values used to scrub the user agent header. Large lists can be retrieved by generalizing the attack and having more than one tag injection. Starting with the first request, n different image tags are injected. If these are not cached, the client will retrieve them from the tagging server. Each individual request will reveal one value used to scrub the user agent field. For each individual image tag up to five alternative URLs will be provided and thus disclose a total of $5n$ user agent values.

V. Experimental results



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

Fig. 6. An overview of the `Torinj` framework

We have implemented a proof of concept *malware* called `Torinj` (see Fig.6), composed of three components: a unmodified *Tor* client, an embedded intercepting proxy and a hidden C&C (command and control) channel. A standard, unmodified Tor client is integrated within Torinj providing access to the Tor network layer. Torinj behaves like any other Tor client and provides similar services including relay or exit functions. It includes a small HTTP proxy used to intercept and relay HTTP requests. Interception and relaying are activated by the attacker using the hidden C&C channel that relies on the hidden service protocol [25] available

General Information	Data set									
	1	2	3	4	5	6	7	8	9	10
Request/Response pairs	402	411	404	462	416	401	408	408	420	406
User-Agents	11	10	10	8	12	11	12	7	6	4
Cookies	43	62	48	50	81	51	66	44	95	48
URLs	56	160	133	109	167	144	182	128	148	156
text/html	0.520	0.198	0.227	0.380	0.450	0.185	0.228	0.215	0.231	0.211
text/css	0.280	0.001	0.042	0.005	0.027	0.015	0.045	0.030	-	0.075
application/x-javascript	0.055	0.061	0.070	0.017	0.042	0.053	0.128	0.110	0.058	0.067
application/shockwave-flash	0.045	0.005	0.022	-	0.022	-	0.032	0.038	0.010	0.052
image/jpeg/png/gif/x-icon	0.0308	0.67	0.601	0.413	0.381	0.555	0.463	0.545	0.564	0.491
text/javascript	0.020	0.015	0.002	0.012	0.020	0.035	0.010	-	0.014	0.035
text/plain	0.018	0.022	0.017	0.014	0.030	0.02	0.020	0.012	0.091	0.022
application/xml	0.005	0.007	0.007	-	-	0.01	0.005	0.013	0.002	0.012
text/xml	0.002	0.005	0.005	0.007	0.015	0.002	0.002	0.01	0.002	0.005
Miscellaneous Content	0.004	0.007	0.002	0.131	0.011	0.002	0.010	0.009	0.004	0.004

TABLE I
GLOBAL STATISTICAL INFORMATION

		Sensitivity, Specificity and Q_{total} for 3 different θ											
Data set	classes	θ = 0.1				θ = 0.2				θ = 0.3			
	C	ϕ	ψ	Q_{total}	N	ϕ	ψ	Q_{total}	N	ϕ	ψ	Q_{total}	N
1	7	0.469	0.522	0.795	357	0.534	0.824	0.886	306	0.616	0.823	0.877	285
2	9	0.354	0. 486	0.647	357	0.454	0.713	0.712	320	0.541	0.716	0.737	285
3	8	0.466	0.537	0.536	358	0.521	0.75	0.644	323	0.549	0.747	0.676	275
4	6	0.448	0.536	0.643	364	0.478	0.541	0.693	332	0.523	0.55	0.780	273
5	5	0.551	0.664	0.723	357	0.604	0.692	0.810	315	0.708	0.796	0.838	290
6	8	0.538	0.527	0.852	352	0.522	0.6	0.873	314	0.557	0.728	0.882	289
7	11	0.307	0.589	0.608	362	0.33	0.685	0.701	308	0.422	0.685	0.711	270
8	7	0.323	0.485	0633	349	0.314	0.508	0.719	317	0.405	0.512	0.764	259
9	4	0.495	0.626	0.728	371	0.448	0.92	0.70	313	0.576	0.947	0.812	271
10	4	0.507	0.688	0.820	360	0.638	0.689	0.818	302	0.696	0.719	0.918	269

TABLE II
SENSITIVITY ϕ, SPECIFICITY ψ, Q_{total}, THE NUMBER OF CLASSES C AND THE NUMBER OF SAMPLES N FOR DIFFERENT THRESHOLDS θ

in Tor to provide some anonymity [19] to the C&C interface and its user. The attacker accesses the C&C channel of each Torinj bot through the Tor network. For testing we used three machines. The first machine M_1 ran an unmodified Tor exit node (v0.2.1.14-rc). M_2 ran BIND[3](v.9.4.2), as DNS server with tcpdump[4] to capture all DNS queries/responses. M_3 had an Apache[5] web server (v.2.2.6), hosting the transparent image simulating a malicious payload. From a legal and ethical point, we avoided injecting malicious JavaScript payloads like XSS-proxy[6] or BEEF [5]. The machines were synchronized with NTP [16] for precise timestamps. Connected to the Tor network, we set up a web proxy implemented in Perl[7](v.0.23) and extended it to inject tags. We used iptables[8] to reroute traffic originated from the Tor exit node to our Perl proxy server via the Internet. Inside the web proxy we generated tags. As information sources we used tcpdump on M_1 and M_2, web server logs and web proxy logs. The processing was done using Perl, sqlite3 [20] and a modified version of tcpick [23].

A. Passive attacks

We ran a Tor exit node for a period of 28 hours and passively inspected captured HTTP headers. We observed similar results to [15], 96% traffic is HTTP and only 4% traffic is end-to-end encrypted with HTTPS. Our injection attack works only for HTTP replies that have the associated MIME [17] type set to `text/html` and consequently we measured the proportion in real traffic. As shown in Table I, about 30% of all HTTP replies have the MIME type set to `text/html`. Over this timeframe, we injected 627 reply messages and observed 879 unique download requests from the tagging server. Our exit node provided 2Mbit/s bandwidth and we assume our attack undetected because the exit node did not get blacklisted.

B. The hidden exit node

We aimed to assess the accuracy and precision of our attack. For this, we installed a hidden exit node and asked students in class to use it. We performed ten different experiments with total control of the browsing history. By this, we knew exactly which URLs belonged to which user session and thus, we were able to compare our results with reality (see Table I). Further, we investigated the impact of the tagging process. While tagging all HTTP requests having the MIME type set to `text/html` is possible, an advanced attacker could be less invasive and tag only a subset of the HTTP requests. For instance she could use a probabilistic scheme driven by a threshold, i.e. for each HTTP reply, a random number (between 0 and 1) is generated and if it is less than a predefined threshold then a HTML injection is performed. Tagging has been done for three different threshold values, the injection probabilities θ. The outcomes are given in Table II. We learned that 30% of the HTTP responses contained HTML documents, so an attacker can set $\theta = 0.3$ ($\sigma = 1$) as maximal value, which

[3]https://www.isc.org/software/bind/
[4]http://www.tcpdump.org/
[5]httpd://www.apache.org
[6]http://sourceforge.net/projects/xss-proxy/
[7]http://search.cpan.org/dist/HTTP-Proxy/
[8]http://www.iptables.org/

gives the best prediction quality and can be explained that more request/responses are known. To show the robustness of our method, we calculated the standard deviation for our 10 data sets, which is 0.08.

C. Defense strategies: playing Tor games

We applied the game theoretical model to data set 7, calculated the payoffs (see Table III) and computed the Nash equilibrium using the Gambit library [27]. We obtained an equilibrium (see Table IV) with two pure strategies. A pure strategy can be defined as a set of actions providing a complete definition of a user's strategy, allowing to say what move a player could make in any situation. This result shows that under game assumptions, the best strategy is to randomly change the user agent. This is unexpected since we were anticipating a different outcome. It seemed far better to delete the user agent field and thus provide no information at all. We believe that the random spoofing of user agents adds more noise and so disrupts the clustering process.

θ	Payoff values for attacker p_a and user p_u							
	$a_{1,0}$		$a_{1,1}$		$a_{1,2}$		$a_{1,3}$	
	p_u	p_a	p_u	p_a	p_u	p_a	p_u	p_a
0.1	0.42	0.52	0.77	0.20	0.42	0.53	0.45	0.45
0.2	0.36	0.51	0.70	0.24	0.32	0.55	0.32	0.54
0.3	0.23	0.54	0.65	0.24	0.29	0.5	0.27	0.51
0.4	0.30	0.42	0.62	0.23	0.26	0.45	0.22	0.47
0.5	0.23	0.38	0.53	0.22	0.23	0.39	0.26	0.37

TABLE III
PAYOFFS FOR ATTACKER p_a AND *Tor* USER p_u

Nb	Attacker					Tor User			
	0.1	0.2	0.3	0.4	0.5	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
1	0	0	1	0	0	0	1	0	0

TABLE IV
NASH EQUILIBRIUM TABLE

We have studied the impact of changing the user agent list for a end user population. In data set 7 we varied the proportion of users changing their user agent. We considered a list of 11 user agents. Changing the user agent is implemented by randomly choosing one value on a per-request basis. We simulated this strategy using real data by directly changing the user agent field in the raw data and ran a simulation with $\theta = 0.3$ ($\sigma = 1$). We observed that the accuracy is strongly impacted when all users change their user agent, which resulted in a loss in classification quality (Q_{total}).

D. The influence of changing the weight control factor σ

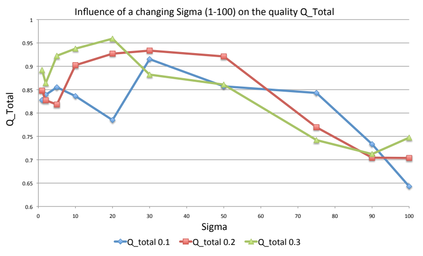


Fig. 7. The influence of σ on Q_{Total}

The weight factor σ influences the label propagation quality as it is responsible for the width scaling of the weighting function. A good choice is therefore essential for achieving good classification results. In [33], σ is estimated by using the Kruskal algorithm [8] to create minimum spanning trees as a heuristic to set the σ-parameter. We used a different method. We referred to a grid search evaluation to estimate the best values for σ for maximizing the classification quality Q_{Total}. The grid search experiment uses tag-injection values θ={0.1,0.2,0.3} refing a tagging of 10-30% of all data. We observe that the classification quality Q_{Total} changes with different σ-values and results improve the more data is tagged (see Fig.7). Best results are obtained for weight control factors ranged between 30-50 in our case.

VI. RELATED WORK

The main purpose of Tor is to provide anonymous communication services by setting up an overlay network with entry guard, relay and exit nodes. A client connects to the entry guard and sets up a path towards the exit nodes. In this circuit, each node only knows its predecessor [7]. Profiling attacks on encrypted web proxy traffic have been studied by analyzing the number of bytes exchanged [12]. In [15], they captured traffic at entry guards and exit nodes to get insights into Tor by studying cleartext protocols like HTTP and Telnet. They detected that HTTP is the most frequently used protocol and they determined the number of users passing their entry guard, because they could see the user location. Since traffic from an exit node is anonymized, it becomes hard to differentiate users. [7], [15] gave a thread model for Tor. An attacker can intercept traffic fractions or generate, modify or delay traffic and compromise Tor nodes. In [7], various attacks on different Tor nodes classes are described and [15] even presents countermeasures against traffic intercepting exit nodes with major assumption on attackers performing DNS reverse lookups in real time. Furthermore, efforts are made to erase sensitive information like user agents and cookies from HTTP requests by using local proxy implementations, e.g Privoxy. In [31] is shown that many Tor users transmit sensitive information, like account names, user names and passwords through the Tor network without using end-to-end encryption. More similar to our approach is the timing attack with JavaScript injection described in [1]. They assume that both, entry and exit node are controlled by an attacker and that JavaScript can inject unique temporal signals for each browser. The aim and approaches of our work are different from [1] because we aim to reconstruct the browsing sessions by controlling only an exit node and mine observed traffic, while in [1] they aimed at identifying clients by time-based attacks.

VII. CONCLUSIONS

We have described a new inference attack against anonymous communication systems that combines an active tag injection scheme with a Data Mining algorithm to cluster observed traffic flows. Each cluster corresponds to the browsing history of one user. We have assessed the performance of this attack on several realistic data sets and proposed a formal modeling framework based on game-theoretical concepts. More advanced attack and defensive strategies have been evaluated in context of this framework and an optimal set of strategies has been identified (in the sense of the Nash equilibrium). We did not intend to reveal the user location or identity, but it could be done by going beyond a simple HTML injection by injecting malicious JavaScript like Beef [11] or XSSProxy, because simple JavaScript injection would build permanent connections (for the lifetime of a browsing session) and allow advanced recognition actions against users' privacy. Our approach has some limits. If all users use end-to-end encryption as SSL, our method is not able to correlate pairs of requests/replies to reconstruct browsing histories. However, based on our experiments, less than 3% of all traffic is encrypted. Another limitation of our method is due to the maximum life-time of an established Tor tunnel that is currently set to 10 minutes. The described attack is particularly suited for Web browsing reconstructions. User agent-specific headers and HTTP parameters drive the clustering phase. A further challenge is to extend our approach to the more general case of any application using anonymous overlay networks. Several browser plugins already block image tags. PithHelmet[9], an optional ad-blocker for Safari[10] includes this option and also blocks images matching a (fixed) list of common banner ad sizes, e.g. 1x1pixel. It blocks them by displaying a transparent graphic of the same size after downloading them first, in order to define the image size, so our attack still works. ICab[11] mobile, a little-known third-party browser for the iPhone has a user-editable list of sizes to block. But again, even if 1x1 images are blocked, they have to be downloaded first. The more general ad-block functionality, by Adblock Plus[12] for FireFox, simply does not download content from URLs matching any of a list of patterns. This offers no protection against our attack unless the image-hosting site gets blacklisted. An interesting future work consists in counting Tor users. This is particularly difficult, since many exit nodes have major differences in terms of bandwidth, service policies and location. For this purpose, we are currently researching extrapolations based on local (one or several exit nodes) observed network traffic.

Acknowledgment

This project is partially supported by the EFIPSANS EU Project.

References

[1] T. Abbott, K. Lai, M. Lieberman and E. Price, *Browser-based attacks on Tor*. Privacy Enhancing Technologies, 7th International Symposium PET 2007, LNCS 4776, 185-199, Springer Verlag, 2007.

[2] P. Baldi, S. Brunak, Y. Chauvin, C.A.F. Andersen and H. Nielsen, *Assessing the accuracy of prediction algorithms for classification: an overview*. Bioinformatics review, vol.16 no.5 2000, 412-424, 2000.

[3] K. Bauer, D. Maccoy, D. Grunwald, T. Kohno and D. Sicker, *Low-resource routing attacks against anonymous systems*. Proceedings of ACM Workshop on Privacy in the Electronic Society, 2007.

[4] C.M. Bishop, *Neural networks for pattern recognition*. Oxford University Press, 0198538642, 1995.

[5] R. Cannings, H. Dwivedi and Z. Lackey, *Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions*. McGraw-Hill Osborne Media, 0071494618, 978007149618, 2007.

[6] D. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*. Communications of the ACM, vol.4, no.2, 1981.

[7] R. and N. Dingledine, *Tor: The second-Generation Onion Router*. Proceedingss of the 13th USENIX Security Symposium, 303–320, San Diego, CA, USA, 2004.

[8] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem.*. In Proceedings of the American Mathematical Society, volume 7, 48-50, 1956.

[9] M. Gilleland, *Levenshtein distance, in three flavours*. http://www.merriampark.com/ld.htm.

[10] A. Greenwald, *Matrix Games and Nash Equilibrium*. Lecture in Game-theoretic Artificial Intelligence, Brown University CS Dep. Providence, US, 2007.

[11] J. Grossmann, R. Hansen and P. Petkov, *Cross Site Scripting Attacks: Xss Exploits and Defense*. Syngress Media, 13-978-1597491549, 2007.

[12] A. Hintz, *Fingerprinting Websites Using Traffic Analysis*. Privacy Enhancing Technologies, 171–178, 2002.

[13] *HTTP*. RFC2616, http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[14] B.N. Levine, M.K. Reiter, C. Wang and M. Wright, *Timing attacks in low-latency mix-based systems*. Proceedings of Financial Cryptography, 2004.

[15] D. Maccoy, K. Bauer, D. Grunwald, T. Kohno and D. Sicker, *Shining Light in Dark Places: Understanding the Tor Network*. PETS'09: Proceedings of the 8th International Symposium on Privacy Enhancing Technologies, Leuven–Belgium, Springer-Verlag, 63–76, 978-3-540-70629-8, 2008.

[16] D. L. Mills, *Network Time Protocol (Version 3) Specification, Implementation and Analysis*. RFC, 1992.

[17] *MIME - Media Types*. RFC2045, http://www.isi.edu/in-notes/rfc2045.txt.

[18] S.J. Murdoch and G. Danezis, *Low-cost traffic analysis of Tor*. Proceedings of The IEEE Security and Privacy Symposium, 2006.

[19] L. Overlier and P. Syverson, *Locating Hidden Servers*. Proceedings of the IEEE Symposium on Security and Privacy, IEEE CS, 2006.

[20] M. Owens, *Embedding an SQL database with SQLite*. Linux Journal n.110, vol. 2003, Specialized Systems Consultants In., Seattle, WA, USA, 1075-3583, 2006.

[21] Ø. Lasse, *Locating Hidden Servers*. SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, USA, 100-114,2006.

[22] *Securing the Tor Network*. http://www.freehaven.net/˜arma/SecuringTheTorNetwork.pdf.

[23] F. Stablum, *Tcpick*. http://tcpick.sourceforge.net/.

[24] Timothy G. Abbott et al, *Browser-based attacks on Tor* Lecture Notes in Computer Science.

[25] *Tor: Hidden Service Protocol*. http://www.torproject.org/hidden-services.html.en.

[26] *Torscanner*. http://code.google.com/p/torscanner/.

[27] T. Turocy, *Gambit*. http://gambit.sourceforge.net/, 2007.

[28] X. Wang, S. Chen and S. Jajodia, *Network flow watermarking attack on low-latency anonymous communication systems*. Proceedings of the IEEE Security and Privacy Symposium, 2008.

[29] X. Wang, J. Luo, M. Yang and Z. Ling, *A novel flow multiplication attack against Tor*. Proceedings of the 13th IEEE International Conference on Computer Supported Cooperative Work on Design, 2009.

[30] W. Yu, X. Fu, B. Graham, D. Xuan and W. Zhao, *Dsss-based flow marking technique for invisible traceback*. Proceedings of the IEEE Security and Privacy Symposium, 2007.

[31] K. Zetter, *Tor Researcher Who Exposed Embassy E-mail Passwords Gets Raided by Swedish FBI and CIA*. 2007.

[32] Y. Zhu, Y. Fu, B. Graham, R. Bettati and W. Zhao, *On flow correlation attacks and countermeasures in mix networks*. Proceedings of Workshop on Privacy Enhancing Technologies (PET), 2004.

[33] X. Zhu and Z. Ghahramani, *Learning from labeled and unlabeled data with Label Propagation*. Technical Report CMU-CALD-02-107, Carnegie Mellon Univ., 2002.

[9] http://www.culater.net/software/PithHelmet/PithHelmet SampleAdBlocking.php

[10] http://www.apple.com/safari/

[11] http://www.icab.de/

[12] http://adblockplus.org/en/