An Anonymity Vulnerability in Tor

Qingfeng Tan, *Member, IEEE*, Xuebin Wang, Wei Shi, *Member, IEEE*, Jian Tang, *Fellow, IEEE*, and Zhihong Tian, *Senior Member, IEEE*

***Abstract*— Privacy is currently one of the most concerned issues in Cyberspace. Tor is the most widely used system in the world for anonymously accessing Internet. However, Tor is known to be vulnerable to end-to-end traffic correlation attacks when an adversary is able to monitor traffic at both communication endpoints. In this paper, we present a set of novel Trapper Attacks that can be used to deanonymize user activities by both AS-level adversaries and Node-level adversaries in a Tor network. First, AS-level adversaries can exploit the occasional failures of censored network to selectively control entry guards of the Tor users. Second, the adversaries can exploit poor reliability of the Tor communication (e.g., natural churn) to compromise the exiting nodes and the anonymous path. Once the adversaries gain control of the routes, they can identify and inspect any traffic entering and leaving the Tor network, consequently, deanonymize a Tor user's activity in the network. To demonstrate the effectiveness and feasibility of this attacks, we implemented a tool that can launch the proposed Trapper Attacks to automatic reveal communication relationships between a Tor user and its destinations running on a live Tor network. We also present a formal analysis framework to evaluate the integrity of the Tor network. With this framework, we successfully obtained quantitative estimates of Tor's security vulnerability. The proposed Trapper Attacks are also designed to scale up in real-world Tor networks. Namely, it allows an adversary to perform deanonymization in honey relays effectively, and compromise the anonymity of Tor clients in real time. Our experimental results show that the proposed attacks succeed in less than 40 seconds achieving a 100% accuracy rate and a false positive rate close to 0.**

***Index Terms*— Tor, traffic analysis, deanonymization, denial-of-service attacks.**

I. Introduction

NOWADAYS, with the fast development of Internet, the emergence of Internet-of-Things (IoT) and cyber-physical systems (CPS) have drastically changed our daily life and impacted individuals, societies and industries. Nevertheless, security and privacy issues are the main challenges in the pervasive Internet environment. For instance, most Apps collect personal information during Internet users' online activities from IoT devices, such as cell phones and other embedded devices. Users' personal information may be leaked and their privacy may be compromised.

In the past few decades, many anonymous communication systems have been designed and developed, aiming at protecting user identities from being revealed by untrusted destinations and third parties on Internet. Tor [1] is one of the most popular low-latency anonymous communication systems. As of early 2020, the Tor network comprises more than 7000 Tor routers operated by volunteers all around the world and carries terabytes traffic daily [2]. Millions of users around the world use Tor to protect their communication anonymity, including law-enforcement, intelligence agencies, whistle-blowers, journalists as well as ordinary citizens worried about the privacy of their online communications.

Anonymous communication techniques have received great attention, however are exploited by the adversaries in various ways. They face increasing growth of abuses for various illegal purposes such as having drug trading, hiding the executing of botnet command and control (C&C) servers, and sending spam over Tor. For example, WannaCry worm communicates with C&C servers hosted on Tor network to provide hidden service and avoid traceback.

A. Challenges in Tor Deanonymizing

Deanonymizing Tor network means recognizing communication relationship between a Tor user and his/her traffic destination (e.g. a web service or a website) in Tor network. Previous research works have presented various attacks to deanonymize Tor network through controlling guard relays so as to compromise routing path, or by correlating the underlying network communications. Unfortunately, all existing traffic correlation attacks assume that an adversary will be able to monitor Tor traffic that lies on the forward path or reverse path between source and destination. These studies make hypotheses about user settings, adversary capabilities, and the nature of the Tor network that are not able to meet the practical scenarios. More importantly, Tor's increasing scale (over 7000 Tor routers) and geographical distribution of Tor's nodes across many countries lead to the fact that the network flows from Tor clients to Internet destinations are often across wide area networks (e.g. the client to guards and exit to destination lie on different countries respectively).

Manuscript received September 26, 2021; revised March 11, 2022 and May 4, 2022; accepted May 6, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Caesar. This work was supported in part by the National Natural Science Foundation of China under Grant 61972105 and Grant U20B2046, in part by the Higher Education Innovation Group under Grant 2020KCXTD007, in part by the Guangzhou Higher Education Innovation Group under Grant 202032854, and in part by the Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme in 2019. *(Corresponding author: Zhihong Tian.)*

Qingfeng Tan and Zhihong Tian are with the Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China (e-mail: tianzhihong@gzhu.edu.cn).

Xuebin Wang is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China.

Wei Shi is with the School of Information Technology, Carleton University, Ottawa, ON K1S 5B6, Canada.

Jian Tang is with Midea Group, Foshan 528311, China.

Digital Object Identifier 10.1109/TNET.2022.3174003

To defend against deanonymization attacks, Tor introduced entry guard as its first-hop router to its core network. In this case, each Tor client maintains a fix list of guard relays and chooses one of them as the first hop whenever a new circuit is created. As presented in Tor guard specification [3], the guard list of a Tor client does not change within 120 days as long as all guards remain reachable. Therefore, even with the continuous growth of a Tor network, a relay node compromised by an adversary is less likely to be selected according to the guard selection algorithm. Further more, Elahi *et al.* [4] and Johnson *et al.* [5] suggest that the main impediment of deanonymization attacks for an adversary is to fully compromise the guard list of a given user.

Therefore, how many guard relays an adversary can control is the key challenge on Tor network for deanonymization attacks. These controlled guard relay nodes can either be an existing guard relay that is compromised by an adversary or a *Honey Relay* injected into a Tor network by an adversary. We say this list of adversary controlled guard relays are *Known* to the adversary.

B. Observation and Goals

In Tor's current defence model, it assumes that an adversary can manipulate a fraction of the onion routers and monitor traffics at the forward path or reverse path on the network, but Tor's routing algorithm ensures that it is difficult for adversaries to observe all the relays on a given routing path. A key observation on this model is that adversaries need to successfully compromise the two endpoints of a new Tor client's connections: guard and exit relay. In fact, the existing Tor network is a volunteer-operated overlay network, the Tor network evolves dynamically over time as new onion routers constantly join while others leave. Hence, an adversary can inject fake onion routers onto paths that are supposed to be on censored networks with Internet destinations. Or in other words, if the adversary can force the targeted Tor client traffic on to honey relays without their explicit collaboration, we can then selectively affect the reliability of Tor circuits.

On the other hand, such an attack can be easily misperceived as poor reliability of the Tor network communication. An adversary can stealthily perform it, which does not require the attacker's capable of controlling the multitude of Autonomous Systems (ASes), but to inject several honey relays. In addition, Trapper Attacks can also be combined with other attacks, such as Tor's path selection attacks on multiple ASes.

In this paper, we focus on the scenarios in which adversary is able to monitor only one endpoint of communication. The main goal of the adversaries could be either to compromise as many circuits as possible for a given class of users (e.g., lie on a censored network), or to deanonymize Tor clients accessing to a given destination (or a set of Internet destinations) during a time period.

To understand the severity of the proposed Trapper Attacks on a live Tor network, in this paper, we investigate the practical Trapper Attacks that are capable of improving tremendously the efficiency and accuracy over existing attacks on deanonymizing a Tor network. We also implement our attack, using honey relays, which takes input from a range of IP addresses of potential Tor users or a list of targeted websites. To demonstrate the effectiveness of the proposed Trapper Attack, we have built a live Tor network and performed tests on it.

C. Our Approach

In this study, we investigate on ways to alter Tor's guard selection algorithm. We modify the algorithm in order to alter Tor's path selection so that the nodes fall on the selected path are *known* to the adversary. Our approach includes three steps. First, we perform injections of several honey relays as guard relays and exit relays into the Tor network. Second, we propose techniques to force Tor path selection to select nodes that are known to the adversary. The idea of such techniques is to deny a Tor users' connections to trustworthy onion routers so that the user's traffics move toward the injected honey relays.

Finally, we can identify and inspect each traffic entering or leaving the honey relays, and then make correlation between a user (i.e. his/her IP) and a Internet destination (i.e. a web server IP), therefore, achieve deanonymization. Therefore, the key idea behind the proposed Trapper Attacks are to provide an adversary the ability to observe Tor users' traffic information on both ends of the communication with a high probability.

Our results show that Trapper Attacks can deanonymize a Tor user with a 100% accuracy rate and the false positive rate of almost 0 within 40 seconds on the live Tor network examined. The results also show that Trapper Attacks can increase the chance of compromising Tor users by up to 90% within 4 hours through deploying honey relays, in which 100 are guard relays and 20 are exit relays.

Furthermore, when the proposed Advanced Trapper Attacks are also combined with Tor's relay selection algorithm, it accelerates the speed of updating a Tor client's list of guard relays. The final result indicates that the time interval a client updates its guard relays is 3 minutes on average. This time interval is shortened from 3-6 months to 3 minutes, which greatly accelerates the process of compromising the guard-relays-list of a targeted client.

D. Our Contributions

In this paper, we investigate the severity and the security implications of the proposed Trapper Attacks on the Tor network. We comprehensively study the efficiency, accuracy and feasibility of the proposed attack as well as validate it with experiments carried out on a live Tor network. The main contributions of this paper are presented as follows:

i) We present a set of new attacks, called Trapper Attacks, that can provide an adversary the ability to selectively control Tor's routing path. Such an attack degrades the Tor path selection of a Tor user from probabilistic node selection to deterministic node selection.

ii) We then present a detection evasion study conducted on honey relays against Sybilhunter [6] and DannerDetector [7], which can significantly increase the survivability of the injected honey relays.

iii) We explain a proof-of-concept implementation of our proposed approach based on Trapper Attack that results in communication deanonymization.

iv) We have conducted a thorough evaluation on the practicality of the proposed attacks, ranging from compromising the list of guards of a given Tor client to carrying out attacks affecting the network as a whole. We present our evaluation results that are based on both large-scale simulations and real-world experiments performed on a live Tor network.

II. Related Work

In the past few decades, various methods of deanonymization attacks have been proposed against Tor. In this section, we focus on the most relevant works for such attacks in the literature which we will review below.

A. Path Selection Attacks

Tor path selection algorithm aims to help Tor clients avoid an adversary to monitor any direction of the traffic at both endpoints of the communications [8], [9].

Pappas *et al.* [10] proposed an attack which is called packet spinning, where an adversary built circular paths with target relay pass through the Tor network to keep the relay "busy". In this case, the adversary can force Tor users to select target relay with less probability. Jansen *et al.* [11] proposed the sniper attack based on a loophole of Tor, where an adversary could terminal an arbitrary relay in consensus list just in several minutes. Bauer *et al.* [12] proposed low-resource routing attacks against Tor network to replace all entry nodes with attackers' relays by falsifying the advertised bandwidth capacity. Borisov *et al.* [13] presented a selective DoS attack that malicious onion routers might choose only to facilitate connections with colluding relays by forcing the client to build new circuits. Tan *et al.* [14] proposed an Eclipse attacks on Tor hidden services that allow adversaries with an low cost to block arbitrary Tor hidden services.

We improve on previous research works in three significant ways: *i)* we explore in depth Tor's routing algorithm to evaluate how the proposed Trapper Attacks can decrease Tor user's anonymity. *ii)* we also propose an anti-detection policy to prevent the injected honey relays from being detected by current detection methods. Moreover, we set up a group of experiments to validate the effectivity of the proposed anti-detection policy. *iii)* our proposed Trapper Attacks can be executed by both AS-level adversaries [15], [16] and Node-level adversaries [17]. In addition, we implement the proposed attacks on a live Tor network in order to estimate their effect.

B. Traffic Analysis Attacks

Traffic analysis make use of traffic metadata, such as packet size or timings, to correlate the flows of the communicating end points [18]. In 2007, Murdoch *et al.* [19] showed that Internet-exchange-level adversaries can use sampled NetFlow data in IXes to launch traffic analysis attacks on the Tor network. Murdoch and Danezis presented a low-cost traffic analysis techniques that allow adversaries to infer which routers are being used to relay the anonymous streams [20]. Mittal *et al.* [21] showed that the use of throughput fingerprinting about the Tor relays in a circuit could be used to fingerprint Tor relays. Hopper *et al.* [22] studied network latency as a side channel to compromise the anonymity of Tor clients, and this information leaks can be used to associate two different flows to the same circuit by measuring the round trip time when a client connected to a pair of colluding Web sites. Johnson *et al.* [5] proposed a new metric for analyzing the security of the Tor network against AS-level adversaries. Zhen Ling *et al.* [23] presented a new cell counting attack against Tor to disclose anonymous communication relationship among users. Recent work by Sun *et al.* [24] showed, via an AS-level attack combining BGP hijacks and interceptions, that asymmetric traffic correlation can exactly deanonymize Tor users with up to 90% accuracy in 300 seconds.

III. Threat Model

We present in this section a threat model in Tor network under Trapper attacks. A Tor anonymous communication network is an overlay network over the transport layer. Thus Tor is known to be vulnerable against adversaries that are able to monitor networking traffic entering and exiting the Tor communication channel. Simply by correlating traffics observed, the adversary can identify the user and his/her message destination, completely subverting the protocols security goals. We consider the adversaries as organizations (e.g., ISP) capable of controlling Tor relays as honey relays and also allow adversaries to monitor all the targeted Tor user's incoming and outgoing connections. We suppose that the underlying network protocols are secure, however, the types and amounts of honey routers that an adversary controls to launch an deanonymizing attack is only limited.

We consider that adversaries can observe, alter, or drop a communication connection. For example, China simply blocked access to the IP addresses of each of those known entry nodes in Tor [25], [26]. Such an adversary has the ability to selectively block entry nodes and can therefore force targeted user traffic going through its entry nodes. Thus, the routing-capable adversaries are able to control the routing path between two endpoints of the commnications. Once on the path, the adversary can launch various end-to-end traffic correlation attacks. We discuss types and amounts of adversary resources under the following set of assumptions.

For an end-to-end traffic correlating attacks, an adversary that controls two endpoints of Tor users' connections are of primary importance since they have full visibility to Tor circuits. Thus, we assume that the adversary has some bandwidth and computing resources and the node-level adversary functions can run their own Tor routers or collude with some Tor routers. The network-level adversary function may control one or several ASes and is assumed in that case to monitor, alter, or drop any traffic entering or exiting the Tor network. A network-level adversary function may be interested in investigating who is accessing a specific destinations. Thus a user's entry nodes are an attractive target for the Trapper Attack.

Under some conditions, such an attacker can selectively block Tor connections from by observing all the TCP connections between the Tor client and her entry guards. By blocking the user's entry nodes, the adversary can force the user choose an adversarial relay. This process is repeated until an adversarial relay finally chosen. This selectively Denial-of-Service attack provides the adversary with a better circuit visibility which will dramatically improve the effectiveness of our proposed attack.

IV. The Proposed Trapper Attacks

In this section, we develop a set of Trapper Attacks against the Tor network that can be used to compromise users' circuit by forcing users to choose injected honey relays as guards and exit relays. To facilitate an understanding of the attack methodology, we firstly describe the basic Trapper Attacks which can mount by both Node-level adversaries and AS-level adversaries. We then describe a much safer variant that further protects honey relays from being detected by existing detection algorithms. Finally, we evaluate extensively the effectiveness of the proposed Trapper Attacks in detail.

A. Basic Attack

Tor network is known to be blocked in China, however, Ensafi's research works reveal that failures in the Great Firewall of China (GFC) occur throughout the entire country without any conspicuous geographical patterns [25], [26]. The approximate amount of directly connecting Tor users rarely exceeds 3000. Consequently, the adversary can leverage the occasional failures to selectively control Tor's routing path.

In this section, we consider two representative experimental scenarios for our Trapper Attack, we assume the attacker has several honey relays with high-bandwidth and high-uptimes deployed in the Tor network. In order to deanonymize the Tor network, adversaries are capable of launching honey relays, that selectively affect reliability of Tor nodes so as to dramatically increase the visibility of the adversary. We also assume that Trapper Attacks can be combined with AS-level adversaries, which are ASes, or an organization with the cooperation of ASes. The details of Trapper Attacks on Tor network are presented as follows:

1) Guard Relay Capture (GRC) Attack: Tor's relay selection algorithms attack allows an ISP to manipulate Tor's routing path by deviating from the real guard relays. That is after Hijacking Tor's relay selection, the traffic is routed to the honey relays automatically. Such an attack allows the adversary to monopolize all responsible entry guards of a particular Tor client during a given time period. Our observation is that the entry guards are likely to be updated in Tor's virtual circuit construction as soon as one of the guard relay in its guard list is not available. In this case, AS-level adversaries can attack Tor's relay selection algorithms to force a Tor client to choose a honey relay as a guard relay for all their circuits. The attack causes all Tor client traffics re-routed to the honey guards, consequently become visiable by the adversary. The GRC attack steps are described as follows:

i) Identify the entry guards of a Tor user;

ii) Test if the Tor user selected an adversarial relay as an entry guard. If not, selectively block the user's connections;

iii) Repeat *i)ii)* until an adversarial relay is selected as an entry guard.

AS-level adversaries can deploy a percentage of exit and entry routers, and then whitelist a set of such onion routers (IP addresses) at ISPs or in the locations that are under the adversaries' control. Since Tor chooses the entry guards roughly at random weighted on bandwidth, for every entry guard of Tor users, the adversary can use both deep packet inspection (DPI) and active probing in order to observe and recognize the traffic of Tor protocol, e.g., the fingerprints of SSL/TLS handshakes, including a list of supported cipher suites, packet-size distributions, etc. Recent works have shown that obfuscation protocols of Tor are vulnerable to machine learning attacks and entropy-based tests [27], [28].

Consequently, the ISP can observe the traffics, and retrieve the entry guards of a new Tor connection. if the Tor user don't select an adversarial relay as a responsible guard, then the ISP will block Tor's corresponding connections. Upon noticing this chosen guard not being available (as a result of the previous block of connection) in Tor, the Tor client will consequently be forced to choose a new entry guard.

This attacking process continues until a new entry guard node is added to the guard list. By blocking all the previous guard relays while keeping the malicious guard relay available, all entry guards of the Tor client' circuits used for communication will be under the adversary's control. The AS-level adversaries can implement such routing path attacks to control Tor's traffic, past work has shown that such attacks occasionally fails. Consequently, the GRC attack appears to be fairly innocuous to Tor users, therefore has a high survival rate without being detected.

2) Controlling User's Exit Node: GRC attack discussed above allows the adversary to take control the entry guard of a Tor client, however, in order to succeed a deanonymization attack, an adversary needs to observe a Tor users' traffic at both ends of the communication channel. Therefore, we use Tor Circuit Selective Destruction (TCSD) attack to force both the entry and exit relays of a Tor client onto the honey relays.

Upon choosing an entry guard, a Tor client process will automatically build a three-hop circuit and rebuild such a three-hop circuit every 10 minutes, by default. In the case that a given exit node is not a honey relay, we need to destruct this circuit immediately and repeat such process until the selected exit node is one of the honey relays. Consequently, an adversary controls both the entry and exit guards of the communication which is sufficient to achieve an end-to-end traffic correlation attack. More specifically, the adversaries can collect the throughput fingerprint and circuit construction sequence through *fast traffic analysis*, which starts with computing the correlation coefficient on each pair of controlled entry guard and exit node of the circuits.

To achieve this, we exploit the ability of *CMD_INTERRUPT* command mechanism in Tor that can precisely destruct a Tor circuit using a given circuit ID. Upon receiving a *CMD_INTERRUPT* command given by the entry guard of

a client, Tor client instance is forced to destruct the current circuit and reconstruct a new three-hop circuit with a new exit node. More specifically, a *Circuit Controller* is created to help decide whether a circuit should be destructed. In order to destruct a circuit, the *circuit controller* sends a command to the honey relays via *Tor control protocol* [29]. The honey relay that runs a modified Tor instance (based on Tor-0.4.0.5) has a *CMD_INTERRUPT* controller command added that can be used to destruct a circuit with a specific circuit ID given. The two specific ways of destructing/interrupting a circuit using *CMD_INTERRUPT* command is explained in detail in IV-C.2.

This attack eventually allows both the entry and exit nodes of a Tor client's three-hop circuit to be on honey relays. Such honey routers can then observe a large amount of targeted client traffic, e.g, the guard node can observe the Tor client IP address, and the exit can observe information about the Internet destination.

By forcefully destruct a circuit, an adversary accelerates the Tor circuit reconstruction process that greatly increases the probability of gaining control of an exit node in a timely fashion.

B. The Advanced Attacks With Anti-Attack-Detection Mechanisms

We now turn to discussing anti-detection policies that can improve the survivability of the proposed basic Trapper Attacks by preventing the injected honey relays from being detected. In order to prevent selective DoS attacks, some studies have succeeded to detect malicious relays in Tor network. Beyond what are represented in *Sybilhunter* [6], a deterministic method is proposed by Danner [7] (we use *DannerDetector* to refer to Danner's method in the following sections).

Sybilhunter is the system for detecting Sybil relays based on their appearance and behavior, such as configuration and uptime sequences. The more honey relays we inject into Tor network, the easier of *Sybilhunter* to find out our honey relays. *DannerDetector* aims to detecting selective DoS relays in Tor network. It could find out all honey relays within several rounds of execution. On each round, the detector established multiple testing circuit according to its policy through every relays for downloading a large file. If the circuit is interrupted when the file is downloading, the detector would suspect the existence of honey relays in the circuit.

However, we discovered an anti-detection policy that can bypassed the above-mentioned detecting policy. The anti-detection policy is comprised of three tactics: *i)* obfuscating deployment; *ii)* deferral decision and *iii)*collaborative filtering. Obfuscating deployment aims to bypass the *Sybilhunter* through varying the configuration of Tor instance. Deferral decision and collaborative filtering are proposed to conceal our honey relays from *DannerDetector*. In the following subsection, we describe these three tactics in detail.

1) Obfuscating Deployment: To simplify the management of dozens of honey relays, the reported detection operators tend to administrate their relays simultaneously, such as reboot all of them at the same time or use the same configuration file for each honey relays. Consequently, detectors like *Sybilhunter* would analyze the appearance and behavior of all relays, such as uptime matrix, fingerprint and the nearest neighbour ranking. And it is easy to find out the honey relays owned by a same administrator in Tor network.

In order to bypass the detection of *Sybilhunter*, we propose the obfuscating deployment method. Obfuscating deployment can significantly obfuscate the characteristic of Tor instance's appearance and uptime matrix of the honey relays. The key insight of obfuscating deployment is to randomly inject honey relays into Tor network and, in the meantime, keeping the attribute distribution of the Tor network. Obfuscating deployment contains two part: obfuscating appearance, and obfuscating behavior.

To obfuscate the appearance, we randomize the Tor instance in four dimensions: *IP address*, *nickame*, *port* and *software version*. For the IP address randomization, we randomly choose the geographical location of VPS from operators such as Amazon, Azure, Linode, etc. For the nickname, port and software version, we calculate the distribution of according to the average of the past 5 days, and then generate the configure file according to the calculated distribution.

To obfuscate the behavior, we randomize the uptime of Tor instance using Poisson distribution. Before we run the Tor instance each hour, we calculate the λ_{float} and λ_{stable} each hour. λ_{float} represents the average number of relays joining into the Tor network each hour of the past 3 hours. λ_{stable} represents the average number of relays joining into the Tor network each hour of the past 5 days. Then we set $\lambda \leftarrow \lambda_{stable} - \lambda_{float}$, and the num of Tor instance run in one hour n_{run} complying with Poisson distribution:

$$P(n_{run} = k) = \frac{e^{-\lambda}\lambda^k}{k!}$$

a) Deferral decision: DannerDetector can download a self-owned file through testing circuit which contains the honey relays. If the circuit is not compromised, the honey relay of basic Trapper Attacks would interrupt the circuit almost in a certain time. As a result, *DannerDetector* could easily discover the honey relays.

To solve this problem, we propose a method named *deferral decision*. Honey relays would delay deciding whether to interrupt the circuit in a random time wt. where

$$wt_{min} < wt < wt_{max}$$

Consequently, the lifetime of each interrupted testing circuit would not be similar. Additionally, honey relays can be automatically/semi-automatically deployed by an automation tool or a Python script. The tool will automatically enable and start a tor daemon by getting the configuration, such as *uptime*, *nickame*, *port* and *software version*, etc. as its current settings and variables.

2) Collaborative Filtering: In order to find out each honey relays in Tor network, *DannerDetector* test many rounds to increase the accuracy of its algorithm. In each testing round, *DannerDetector* firstly picks two relays from consensus to play the part as guard and exit node. Then it rotates the rest of nodes as middle node to construct the testing circuit.

For each testing circuit, *DannerDetector* downloads a large file through the circuit. Honey relays tend to interrupt the uncompromised testing circuit and that would result to the failing of file downloading. However, normal relays perform well while the testing circuit is downloading the file. Therefore, *DannerDetector* can recognize the honey relays via analyzing the result of downloading at last.

Fortunately, we have developed a Collaborative Detection (CD) algorithm to protect honey relays from being detected by *DannerDetector* (see algorithm 1). All instances of the honey relays work together and dynamically change the value of its parameter to bypass the testing circuit which has regular patterns created by *DannerDetector*. The process is shown as the following: *i)* We describe the behavior of the honey relays as $(p_{dropping}, p_{accepting})$, where $p_{dropping}$ and $p_{accepting}$ are the probabilities of killing and permitting about un-compromised and compromised circuits. *ii)* All of the honey relays build a history pattern list $list_{hist}$ which contains the historical circuit pattern pt discovered by all honey relays in a sliding time window. *iii)* For each circuit established on honey relays, the *Circuit Controller* (See IV-C.2) extract its circuit fingerprint f, guard node $node_g$ and exit node $node_e$ as circuit pattern pt, and then storage the pt into $list_{hist}$. *iv)* The *Circuit Controller* find out the pattern list contains patterns which is *similar* with current circuit in $list_{hist}$ (we define pt_1 is similar with pt_2 means that the guard and exit node of pt_1 is same with pt_2's and the Pearson correlation coefficient between the fingerprint f of pt_1 and pt_2 is larger than 0.7), and halve the parameter $(p_{dropping}, p_{accepting})$ of honey relays on current circuit according to the number of patterns in pattern list. *v)* The *Circuit Controller* decide whether interrupting the current circuit according to the modified parameter $(p_{dropping}, p_{accepting})$. As a result, honey relays could mount the Trapper Attacks on normal users and conceal themselves when *DannerDetector* create testing circuit on them.

In practical application, we consider the following three kinds of detecting scenario based on the algorithm of *DannerDetector*.

a) Scenario 1 - both guard node and exit node are honey nodes: In this scenario, honey relays lie as the guard and exit nodes of each testing circuit. And *Circuit Controller* would judge each testing circuit as compromised circuit. According to the anti-detection policy, honey relays would extract the circuit fingerprint f, guard node $node_g$ and exit node $node_e$ as circuit $pattern$,

$$pattern \leftarrow (node_g, node_e, f)$$

and dynamically change the value of $p_{accepting}$ referring to the number of patterns which are similar with $pattern$ in $list_{hist}$.

$$p_{accepting} = \frac{p_{accepting}}{2^{\psi(pattern, list_{hist})}}$$

Where $\psi(pattern, list_{hist})$ returns the number of patterns which are similar with $pattern$ in $list_{hist}$. So not all the testing circuit would success in a testing round, As a result, the honey relays on guard and exit would not exposed according to the algorithm of *DannerDetector*.

Algorithm 1 Collaborative Detection Algorithm

Input:
$\boldsymbol{S_{honey}}$ is the set of honey relays;
$\boldsymbol{C_{guard}}, \boldsymbol{C_{middle}}, \boldsymbol{C_{exit}}$ is the guard, middle and exit nodes of the circuit established on honey relays;
$\boldsymbol{t_{run}}$ is the running time since discovering the circuit.
Output:
$\boldsymbol{cmd_{dropping}}$ **:** interrupt the $circuit$;
$\boldsymbol{cmd_{accepting}}$ **:** do not kill the $circuit$

```
list_hist = {}
T_delay ← ramdom(wt_min, wt_max)
while T_run < T_delay do
    recording throughput
/* generate a random number range from 0 to 1 */
random_num ← random(0, 1)
S_ge ← C_guard ∪ C_exit
if |S_ge ∩ S_honey| = 1 then
    /* one of guard and exit node is honey node */
    if random_num < p_dropping then
        /* interrupt circuit according to the probability */
        return cmd_dropping
    else
        return cmd_accepting
else if |S_ge ∩ S_honey| = 2 then
    /* both guard node and exit node are honey nodes */
    pattern ← (node_guard, node_exit, f)
    count ← ψ(pattern, list_hist)
    list_hist += pattern
    if random_num < p_accepting / 2^count then
        /* permit circuit according to the probability */
        return cmd_accepting
    else
        return cmd_dropping
else
    /* both guard node and exit node are not honey nodes */
    /* the middel relay of circuit is honey relay */
    pattern ← (node_prev, node_next, f)
    count ← ψ(pattern, list_hist)
    list_hist += pattern
    if random_num < p_dropping / 2^count then
        /* interrupt circuit according to the probability */
        return cmd_dropping
    else
        return cmd_accepting
```

b) Scenario 2 - both guard node and exit node are normal nodes: In this scenario, honey relays lie as the middle node of each testing circuit. *Circuit Controller* would judge each

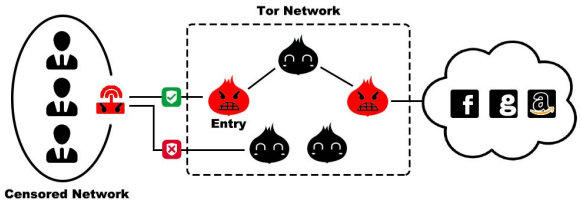


Fig. 1. Attack scenarios for the trapper attacks.

testing circuit as non-compromised circuit when the middle node is honey relay. According to the anti-detection policy, honey relays would extract the circuit fingerprint f, guard node (previous hop) $node_p$ and exit node (next hop) $node_n$ as circuit $pattern$,

$$pattern \leftarrow (node_p, node_n, f)$$

and dynamically change the value of $p_{dropping}$ referring to the number of patterns which are similar with $pattern$ in $list_{hist}$.

$$p_{dropping} = \frac{p_{dropping}}{2^{\psi(pattern, list_{hist})}}$$

Where $\psi(pattern, list_{hist})$ returns the number of patterns which are similar with $pattern$ in $list_{hist}$. So the honey relays at the rear of each testing round would interrupt the testing circuit with a small probability. And this could dramatically protect the part of honey relays in the rear according to the algorithm of *DannerDetector*.

c) Scenario 3 - one of the guard node and exit node is a honey node: In this scenario, testing circuits are interrupted according to the fixing parameter $p_{dropping}$, and the detectors could not exactly distinguish whether middle node or edge node (guard node or exit node) should take responsibility for the circuit failing. For the detecting policy mentioned in the paper [7], detectors will misdeem the middle node as honey relay when the circuit is interrupted in this scenario.

C. Deanonymization in Tor

1) The Trapper Attacks Prototype: In this section, we discuss the affect of the proposed Trapper Attacks on a real Tor network. We consider a representative experimental scenarios for our Trapper Attacks illustrated in Fig. 1. In our attack, we assume the adversary has several honey routers with high-bandwidth and high-uptimes deployed in Tor network. In order to deanonymize the Tor Network, adversaries are capable of launching Trapper Attacks that selectively affect reliability of Tor nodes so as to dramatically increase the visibility of the adversary. We also assume that the attacks can be combined with AS-level adversaries, which are ASes, or an organization with the cooperation of ASes. The proposed Trapper Attacks offer a novel deanonymization that allows the adversaries to compromise the guard and exit relays both at AS-level and Node-level.

Here we use an example to present the basic idea. Let us suppose that a Tor user is communicating to a Web server. Existing traffic correlation analysis only considers the scenario that adversary can monitor any direction of the traffic at both

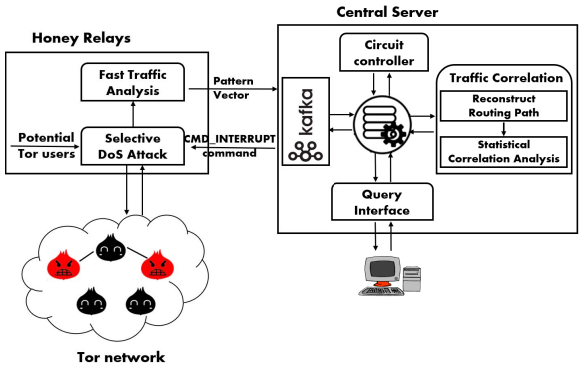


Fig. 2. System architecture.

communicating endpoints. However, network flows from Tor clients to Internet destinations often across wide area networks, the paths between the client-to-entry may lie on censored network, but the paths between the exit relay and the Web server may lie on another country. In addition, Tor clients use a fixed entry guards during a period of time, which will decrease the visibility of the adversaries, thus, the adversary may not be able to monitor the network flows on the both paths between Tor clients and Internet destinations.

To deanonymize Tor network in the wild, we present a system architecture illustrated in Figure 2. It includes three key components: a *circuit controller module*, a *fast traffic analysis module* and a *traffic correlation module*. The *circuit controller module* can selectively affect reliability of Tor circuits so as to dramatically increase the visibility of the honey relays. The *fast traffic analysis module* automatically analyzes each circuit, which extracts the features of its high-level protocol features, e.g., total size of incoming and outgoing cells per second, and then summarizes them into the Tor users' feature vectors respectively. Once feature vectors have been extracted, the *traffic correlation module* then works on these features to correlate (i.e. finding a user-destination pair) any traffic entering or leaving the honey relays.

2) Circuit Controller: In a Tor network, the onion router uses specific cells to communicate with each other and with Tor users. There are two different types of cells: control cells and relay cells. Control cells are always interpreted at the receiving nodes, which can issue commands such as create, created, destroy or padding. Relay cells are used to carry end-to-end TCP stream data. And a relay cell has an additional header. The additional header contains streamID, end-to-end checksum, length of relay payload and relay commands. There are numerous types of relay commands, including *RELAY_BEGIN*, *RELAY_DATA*, *RELAY_END*, *RELAY_SENDME*, *RELAY_DROP*, *RELAY_RESOLVE*, etc. Tor control protocol is used for other programs to communicate with a locally running Tor instance (onion routers), the controller and Tor instance can send typed messages to each other over the underlying stream via Tor control protocol.

For every possible entry and exit traffic flow combination, the *Circuit Controller* first tests whether a Tor circuit needs to

TABLE I
CIRCUIT-BASED FINGERPRINTING FEATURES

Feature	Value	Description
Cell Count	Integer	The amount of cells that a circuit sends or receives per unit time
Throughput	Float	The amount of data that a circuit transfers per unit time
Cell Type	Boolean	The *Command* field of a fixed-length cell
Direction	Boolean	Incoming or outgoing
Circuit ID	Integer	The *CircID* field determines which circuit

be destroyed according to the Pearson correlation coefficient ρ. If the $\rho < t$ (suggests t = 0.7), then the *Circuit Controller* can send *CMD_INTERRUPT* command (which is a control cell) to the honey relay for destroying a given Tor circuit. Once received a *CMD_INTERRUPT* command from *Circuit Controller*, an entry guard (on a honey relay) could interrupt the circuit in two ways: *i)* Modify one random bit of data cell which is carried by the target circuit. The modified data cell will cause the failure of decryption at the edge of a circuit. As a result, the circuit would be interrupted by the guard relay or exit relay of that circuit. *ii)* Directly send a *CMD_DESTROY* cell to the adjacent relay in circuit. Consequently, the circuit would be destroyed recursively. The circuit controller would randomly choose one method to interrupt the circuit in practice.

Additionally, a circuit controller could mount the deferral decision and collaborative filtering to protect the honey relays not to be detected as part of the anti-detection policy proposed in this advanced Trapper Attacks.

3) Fast Traffic Analysis: In this section, we discuss the fast traffic analysis techniques that can be used by an adversary to determine whether two circuits share a common path. We do so by monitoring high-level protocol features and using a statistical correlation algorithm to determine if their flow pattern is correlated. To this end, fast traffic analysis translates each raw cell into pattern vectors that can facilitate further traffic correlation. First of all, given a collection $A_i = (a_{i,1}, \ldots, a_{i,n})$ of n cells at circuit i, where $a_{i,k}$ represents the kth cell on circuit i. Next, given a cell $a_{i,k}$, feature extraction module translates each raw cell into pattern vector $X = \{x_1, \ldots, x_m\}^T$, where m is the number of features. For instance, a cell can be represented with the following pattern vector:

$$< direction, type, sip, sport, dip, dport, circid, time >,$$

where $direction$ is the incoming or outgoing direction of a Tor circuit, $type$ is the cell type, $< sip, sport, dip, dport >$ is the origin-destination pair of Tor circuits, $circid$ is Tor circuit identifier (unique in both directions). Table I illustrates the above-defined features.

4) Traffic Correlation: Traffic correlation aims to determine, from the recorded circuits' fingerprints, if two sets of packets belong to the same user. However, Tor multiplexes more than one TCP streams along a single circuit that can involve traffic exit to a server from hundreds of other client entering the traffic flows constantly. Thus, it is difficult to correctly correlate the flows of Tor users.

Algorithm 2 Traffic Correlation Algorithm

Input:
$\boldsymbol{F_g}$ is the set of circuit-based fingerprints at Guard nodes;
$\boldsymbol{F_e}$ is the set of circuit-based fingerprints at Exit nodes;
$\boldsymbol{t}$ is the threshold of Pearson correlation coefficient.
Output:
$\boldsymbol{P}$ is the set of an origin-destination pair of Tor users.

```
/*  flag potential origin-destination
pairs for every possible entry and exit
traffic flow combination using Tor
circuit construction sequences.        */
```
foreach $f_g \in F_g$ **do**
  foreach $f_e \in F_e$ **do**
    $P_{candidate} \leftarrow$ predicPath(f_g, f_e)
foreach $p \in P_{candidate}$ **do**
  /* for each potential origin-destination pairs, extract its throughput fingerprinting *pattern* at the Guard and Exit node, respectively. */
  $pattern_g^p \leftarrow (node_g, f_g^p)$
  $pattern_e^p \leftarrow (node_e, f_e^p)$
  /* the time T is divided into adjacent time windows W, the time windows W = 1s, by default. */
  foreach $W \in T$ **do**
    /* for each window, calculate Pearson correlation coefficient. */
    $\rho \leftarrow$ Pearson($pattern_g^p$, $pattern_e^p$)
    if $\rho > t$ **then**
      $P \leftarrow (p, \rho)$
return P

In this section, we present our correlation analysis approach to perform exact deanonymization of Tor users (see algorithm 2). The input to traffic correlation algorithm is a sequence of the circuit-based fingerprinting generated by the previous stage. The output of this algorithm is an origin-destination pair of Tor circuits identified by the Trapper Attack, where each origin-destination pair symbolizes a communication relationship of a Tor user.

Our approach uses a two-stage correlation to improve accuracy. The first stage uses the way of Tor circuit construction to flag origin-destination pairs as potentially resulting from a targeted Tor user. The circuit fingerprints of all potential origin-destination pairs is then pushed to a backend system that asynchronously performs a statistical correlation analysis. In order to exploit the circuit construction algorithm, we correlate the timing of each Tor's circuit building stage and recognize the patterns of the cells, e.g. the number, type and direction of the cells over time. In this stage, the adversary can easily reconstruct the potential routing path of a Tor user's connection. In the second stage, we use throughput

fingerprinting of a Tor circuit by computing the amount of cells that a circuit transfers per unit time. Once receiving cell sequence information of a Tor circuit, we build a vector of all unique circuit identifiers. We next divide time into windows of W seconds, and then compute the number of cells x_i and y_i during the ith window for every possible entry-exit pair. Finally, we rely on *Pearson Correlation Coefficient* to identify the closest matching flows for our correlation analysis. Below we briefly discuss the Pearson correlation coefficient we use in this work.

Pearson Correlation Coefficient is a nonparametric measure of the linear correlation between two variables X and Y, It is widely used to assess the degree of linear dependence between two random variables. The Pearson correlation coefficient can be computed from following formula:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}. \quad (1)$$

where, ρ is the Pearson correlation coefficient, $\bar{x}$, $\bar{y}$ are the means of the two observation X and Y, n is the number of windows in each data set.

For each pair of observed Tor clients, we can calculate the Pearson's correlation coefficient ρ between the pattern vector of circuit-based fingerprinting features over time. If the server flows match a Tor client with the highest correlation coefficient, then the IP address pair is added to the final origin-destination pair, that is, if the correlation coefficient ρ for each entry-exit pair exceeds some threshold t (suggests t = 0.7 as producing the best results), we recognize the communication relationship of Tor users.

V. Evaluation

To evaluate the effectiveness of our proposed attacks and understand their security implications, we present a group of experiments on the proposed Trapper Attacks, both on the cost of the attacks and the survivability of the honey relays.

A. Effectiveness of Trapper Attacks

In this section, we first explain a group of experiments that are used to perform the effectiveness evaluation on the proposed Trapper Attacks. To achieve this goal, we modified the source code of Tor-0.4.0.5, which allows us to start logging information on Tor's circuits. Our network clients run at 10 vantage points. In each vantage point, 20 Tor clients can run concurrently. At the beginning of the attack, we sample 10, 20, 50, 100, 150, 200 guards and exit nodes from Tor consensus as our controlled honey relays to validate the catch probability of the guard and exit nodes. Then, the clients running at these vantage points are forced to choose honey relays as entry nodes for all their circuits under the proposed attack. We use Linux's *iptables* rules to simulate Tor's relay selection hijacking between Tor clients and guard nodes, then record the guard and exit nodes selected by clients on each circuit.

For theoretical analysis of catch probability, we assume m honey relays are injected into the Tor network. According to the different tags that Tor authoritative directory servers assign to each onion router, Tor relays can be divided into four types: guard, exit, both guard and exit routers, and neither guard nor exit routers, whose total bandwidth is denoted as B_G, B_E, B_{GE} and B_{NGE} respectively. Let b be the bandwidth advertised by adversarial guards. Then the probability that a Tor client selects adversarial Tor routers as the guards can be calculated as:

$$P_G(b) = \frac{b}{B_G + B_{GE} * W_E}, \quad (2)$$

where the weight W_E can be expressed as

$$max\{0, 1 - \frac{B_G + B_E + B_{GE} + B_{NGE}}{3 * (B_E + B_{GE})}\}.$$

In addition, let $\hat{b}$ be the bandwidth of Tor exit routers of an adversary. Then, the probability that the Tor exit router are chosen for a circuit can be derived from:

$$P_E(\hat{b}) = \frac{\hat{b}}{B_E + B_{GE} * W_G}, \quad (3)$$

where the weight W_G can be expressed as

$$max\{0, 1 - \frac{B_G + B_E + B_{GE} + B_{NGE}}{3 * (B_G + B_{GE})}\}.$$

Now we discuss the time it takes to deny services to trustworthy onion routers so that the honey relays are selected by the Tor client. Assuming adversaries have a budget for bandwidth, that is, the adversary can afford to run a percentage of decoy routers. Denote B as total bandwidth of our Tor guard routers. Assuming a Tor client tries to create a three-hop circuit to communicate with a server through Tor network. After n updates for choosing the entry guard, the probability that at least one circuit select the honey router as guard can be calculated by

$$P_G(B, n) = 1 - (1 - P_G(B))^n. \quad (4)$$

We can also derive the exit compromising probability with the total bandwidth of the Tor exits $\hat{B}$ and the number of Tor exit rotation m based on Equation (2).

$$P_E(\hat{B}, m) = 1 - (1 - P_E(\hat{B}))^m. \quad (5)$$

Recall that if the first and last router in a three-hop circuit will be colluded, the Tor's routing path can be compromised. Consequently, the **compromising probability** that the honey routers are chosen as both entry guard and exit routers for a three-hop circuit can be approximately derived by

$$P_{n,m}(B, \hat{B}) = P_G(B, n) * P_E(\hat{B}, m). \quad (6)$$

Based on the theoretical analysis, we can obtain the probability that a Tor client chooses the guards and exits.

Figure 3 illustrates the impact of the number of guard nodes controlled by the attacker on the probability of getting the first compromised node chosen as a guard node. Figure 4 illustrates the impact of the attacker-controlled guard bandwidth ratio on the number of updates required to get a first compromised node chosen (i.e. the first time a honey node is chosen as a guard), Figure 4 also shows that an adversary can compromise the first guard of the Tor client after blocking 47 Tor circuit

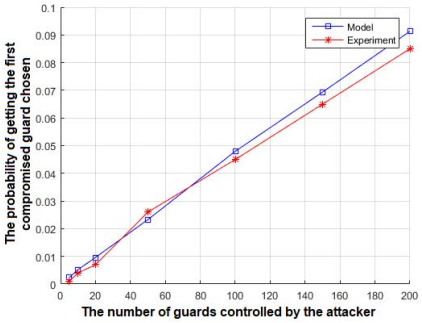


Fig. 3. The impact of the number of guards controlled by the attacker on the probability of getting the first compromised node chosen as a guard node.

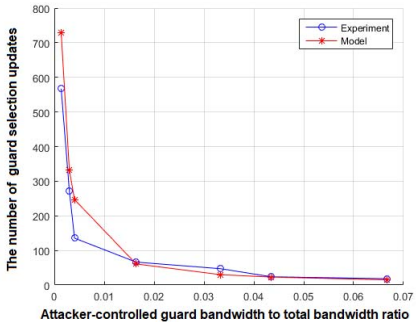


Fig. 4. The impact of the attacker-controlled guard bandwidth ratio on the number of updates required to get a first compromised guard chosen.

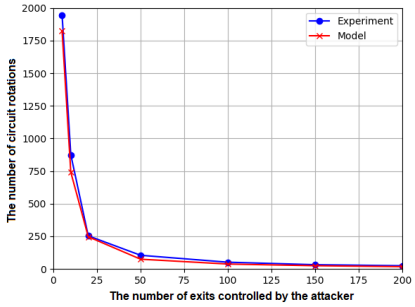


Fig. 5. The impact of the number of attacker-controlled exits on the number of circuit rotations (destroy and rebuilt) required to get a first compromised node chosen as an exit.

creation attempts, when the attacker controls 100 Guard nodes. The time interval used for a client to update its guard relays decreases from 3-6 months to less than 3 min on average. Figure 5 illustrates the impact of the number of attacker-controlled exit nodes presents in the Tor network on the number of circuit rotations (destroy and rebuilt) required to get a first compromised node chosen as an exit. Figure 6 illustrates that the probabilities of an attacker successfully compromising both a Tor user's guard and exit (i.e. a Tor user's guard and exit both are on honey nodes), and the number of circuit recreations, Figure 6 also shows that an adversary can compromise the routing path of a Tor client with 90% probability after blocking 400 Tor circuit creation attempts when 100 guards and 20 exits nodes are controlled. The result indicates that the adversary has a more than 90% chance of

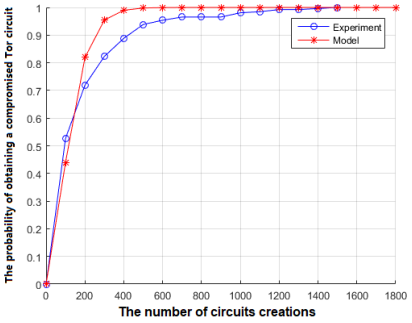


Fig. 6. The success rate of obtaining a compromised Tor circuit (guard and exit) when trying different number of circuits creations with 100 Guards and 20 Exits controlled by attackers.

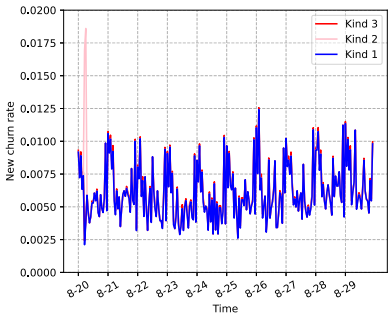


Fig. 7. New churn rate of Tor network.

deanonymization within 4 hours against roughly 100% of users in locations under the adversary' control by 100 guard relays and 20 exit relays.

B. Advanced Trapper Attacks Survivability Evaluation

In order to explore the effectiveness of survivability provided by our anti-detection policy under the circumstance of the current detection policy, we experiment with our simulation implementation of the Advanced Trapper Attack. In this section, we compare the performance of our Advanced Trapper Attacks under *Sybilhunter* and *DannerDetector* detections respectively.

To evaluate the effectiveness of our anti-detection policy under *Sybilhunter*, we generate relay-descriptor files from 2020-01-01 to 2020-08-30 using different policy. For comparison, we use three data sets: *origin set (kind 1)*, *honey set without Trapper Attacks (kind 2)* and *honey set with Trapper Attacks (kind 3)*. The *origin set* is the relay-descriptor file from 2020-01-01 to 2020-08-30 downloaded from Tor Collector Project. For the second kind of *honey set*, we simulate an attacker to randomly launch several honey relays in Tor network without applying our obfuscating method. The third kind of *honey set* is the relay-descriptor which generated by our anti-detection policy.

First, we calculate the network churn rate of the joining or leaving relays using the formula mentioned in [6] on the three data sets. Figure 7 and Figure 8 illustrate the network churn rates during ten days in August 2020. We found an unexpectedly high churn rate without applying the obfuscating method in 2020-08-20. The high new or gone churn rate means

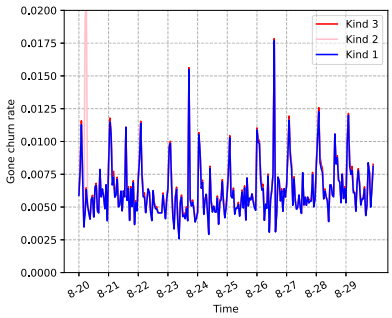


Fig. 8. Gone churn rate of Tor network.

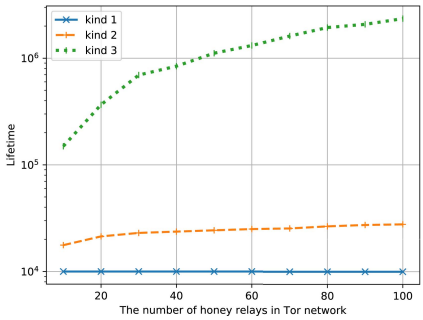


Fig. 9. Lifetime of different policy under DannnerDetector.

TABLE II
RESULT OF UPTIME DETECTION BY SYBILHUNTER

Month	Injected relays	Survival rate with Trapper Attacks	Survival rate without Trapper Attacks
2020-01	209	95.20%	5.74%
2020-02	195	92.80%	5.13%
2020-03	219	93.60%	6.85%
2020-04	220	95.00%	4.09%
2020-05	211	96.20%	6.16%
2020-06	197	94.90%	7.10%
2020-07	205	92.19%	5.37%
2020-08	213	94.36%	4.22%

that many relays joined or left the Tor network as also revealed in Sybilhunter. Figure 7 and Figure 8 also show that the difference of network churn rate between *kind 1* and *kind 3* is small. The injected honey relays do not cause an obvious change on the network churn rate.

Through running *Sybilhunter* on the origin datasets and honey datasets, we can evaluate the effectiveness of the Trapper Attacks before and after applying the obfuscating method. The result (see Table II) of detection shows that we can inject 209 honey relays per month on average with our obfuscating method and about 94.28% of them could not be detected by *Sybilhunter*. However, the survival rate is only at 5.58% without applying our proposed obfuscating method.

To demonstrate the effectiveness of our anti-detection policy under *DannerDetector*, we simulate our anti-detection policy under the detection algorithm to calculate the average testing times until the last honey relay is detected. We define the *lifetime* of policy as average testing times when the last honey relay is detected by *DannerDetector*. We set up three kinds of honey relays which has different behavior parameters. The first kind of honey relays *(kind 1)* interrupt every circuits that are not compromised, which could be represented as $(p_{dropping} = 1.0, p_{accepting} = 1.0)$. The second kind of honey relays *(kind 2)* interrupt the circuits according to a predefined probability value, which could be represent as $(p_{dropping} = 0.9, p_{accepting} = 0.9)$. The third kind of honey relays *(kind 3)* interrupt the circuits complying with deferral decision and collaborative filtering, and the initial parameter of honey relays is $(p_{dropping} = 0.9, p_{accepting} = 0.9)$.

We simulate the behavior of these three kinds of honey relays under the detection algorithm with $N = 10000$, $i = 1, 2, 3$ and $k = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$. Figure 9 shows the lifetime, in logarithmic form, of these three kind of honey relays when different policy under *DannerDetector* are applied. It demonstrates that the *lifetime* of honey relays with deferral decision and collaborative filtering is *i)* about 85 times more than the *lifetime* of honey relays with random interrupt policy when we have 100 honey relays; and *ii)* about 235 times more than the *lifetime* of honey relays which interrupt each uncompromised circuit.

In conclusion, the experiments show that the anti-detection policy drastically increase the anti-detection ability of the injected honey relays under the detection method of both *Sybilhunter* and *DannerDetector*.

C. Accuracy of Deanonymization Attacks

To evaluate the accuracy of deanonymization attacks, we set up three typical usages of Tor network on the live Tor network at 7 vantage points to simulate the realistic Tor users: a web browsing client, and an IRC client, and a bulk transfer client. The three client types work as follows:

i) Web browsing client: We use the iMacros Firefox plugin [30] to automate the recording of web browsing by controlling the functionality of Tor. We first randomly picks a group websites from the list of the top 500 websites reported by Alexa [31]. Then we use the client from each vantage point starts browsing these web pages one by one during a period of time. With this method, we can automatically collect realistic traffic of web browsing clients from a list of the open-world URLs.

ii) IRC client: In order to generate realistic IRC traffics, we implement a simple IRC bot with our python script. We randomly pick a few IRC channels from popular IRC Wiki directories and then configure the IRC bot to login IRC channel and do things automatically based on our scripts.

iii) Bulk transfer client: In order to simulate bulk transfer client, we use the Vuze BitTorrent client [32] to generate P2P file sharing traffic by configuring Vuze over the Tor proxy. We randomly choose a few torrent files from popular torrent websites, which include music, movies,and open source applications, and then download these torrent files one by one.

Additionally, we launched 3 guard and 5 exit nodes as honey relays, and force Tor to explicitly use specific guard and exit nodes when visiting certain destinations. To collect

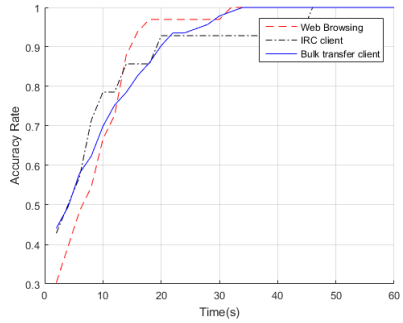


Fig. 10. The accuracy of the attack over time.

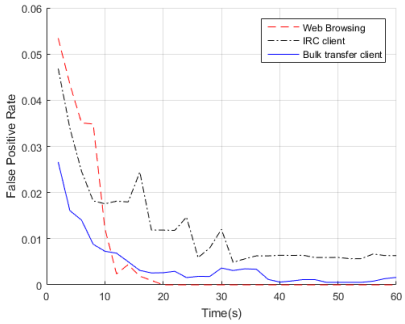


Fig. 11. The FPR of the attack over time.

experimental results, we record the cell sequence information of Tor circuits on multiple honey relays. To achieve this goal, we modified the source code of Tor-0.4.0.5, which allows us to log out the information about Tor's circuits establishing command sequences and data transfer sequences. Once logging the information about each observed circuit, we build a vector of all unique circuit identifiers found from the log file. Further, for each circuit, we compute total lifetime of a circuit that is the time between the circuit creation and circuit destruction. We also compute the total active time of a circuit that is the time from the first to the last relay cells. We next remove outliers by the interquartile range that satisfy the following inequality.

$$\lambda < 300 \text{ seconds}, \quad \text{or } \hat{\lambda} \leq 3 \text{ seconds}$$

where λ is the circuit total lifetime, and $\hat{\lambda}$ is the circuit active time. Then, we periodically push all pattern vectors of Tor circuit fingerprints to central server using Kafka that is an open source message broker software.

Finally, we analyze the accuracy and false positive rate as reported in Figure 10 and Figure 11. These figures show that we can detect a user's source IP and destination IP within 30s with 100% accuracy and less than 1% false positive rate.

VI. Discussion

After demonstrating the threat of Trapper Attacks against Tor network, we now discuss how the proposed attacks would affect the anonymity in other systems and possible countermeasures.

A. Understanding Trapper Attacks and Anonymity Implications

Low-latency anonymous communication systems, such as Tor, are inherently weak to end-to-end traffic analysis attacks, because an attacker can observe traffic pattern on both ends of the communications. However, with the continuous growth of the Tor network, the probability of a particular user chosen both attacker-controlled entry and exit nodes in a Tor circuit remains extremely low. Additionally, the guard mechanism was also designed to mitigate several deanonymizing attacks, e.g, predecessor attack, end-to-end traffic correlation attacks, by decreasing the chance for an attacker to be the entry node of a given user.

The proposed Trapper Attacks can provide an adversary the ability to selectively control a Tor's routing path. Such an attack degrades the Tor path selection algorithm from probabilistic node selection to deterministic node selection. If a client has chosen a compromised guard, the client's entry node will be the compromised guard in every circuit for up to 3-6 months, it poses an imminent privacy and security risk to Tor users.

Other low-latency anonymous communication systems, such as I2P, freenet, also have the volunteer-operated relays. If AS-level adversaries control a fraction of the relay nodes, the proposed Trapper Attacks can then affect path selection algorithm, compromise the client's entry and exit nodes, and launch end-to-end traffic analysis attacks. However, the severity and the security implications on these low-latency anonymous networks should be carefully assessed in relation to the potential impacts of Trapper Attacks in the future.

B. Mitigation of Trapper Attacks

In the following, we propose two possible countermeasures to make Tor path selection more robust: *(i)* **Limit the number of exposed guard nodes.** The key challenges for the successful deployment of a Trapper Attack is that adversaries should selectively control Tor's routing path, such that an adversary has high chance of directing traffic to honey relays. Trapper Attacks exploit the vulnerability of Tor path selection algorithm, that is, entry guards are likely to be updated in Tor circuit construction when the guard in its guard list is not available. There are two possible ways to minimize the chance that the attacker-controlled guards become a Tor user's entry guard. The first is to limit the number of updated guard nodes: if a client can only choose a limited number of entry guards in a given period of time, the chance of choosing one of the attacker-controlled guards will be greatly reduced. The second is to use circumvention tools for getting around compromised guards. For example, use Tor over a VPN: the circumvention tools encryption will prevent the attackers from seeing Tor traffic. *(ii)* **Monitoring behavior of Tor relays.** Recall that the Trapper Attacks need to manipulate Tor circuit. If the manipulated Tor circuits are detected, the effectiveness of such attack could significantly decrease. A key challenge in this case is to find out which router is responsible for tearing down a particular circuit on an anonymous path. In particular, when a Tor circuit is destroyed, it is difficult to identify whether the

circuit is destroyed intentionally or it is due to a node/network failures. One naive way to avoid this is to design a reputation system that monitors the behavior of each Tor router in order to detect the presents of an unreliable router.

VII. Conclusion

In this paper, we present novel deanonymization attacks: the Trapper Attacks that allow adversaries to effectively and accurately identify the anonymous communication over Tor. We demonstrate through experiment that, with the proposed attacks, an adversary gains the capability of controlling routing path either through selectively affecting the reliability of Tor circuits, or hijacking Tor's relay selection, consequently, deanonymizing user communications.

In particular, our approach is to deny service on the trustworthy onion routers so that users' data move toward our injected honey relays. Such attacks can drastically increase the knowledge of an adversary. The feasibility, survivability and effectiveness of the proposed Trapper Attacks are demonstrated through the experiment conducted on a live Tor network in the presence of two different honey relay detection softwares. Results show that our proposed attacks can deanonymize a real world Tor network in near real time with a very high survival rate in the presence of honey relay detection softwares. Finally, we also present a formal analysis framework to quantitative assess the severity of the proposed attacks on the live Tor network and its security implications.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. 13th Conf. USENIX Secur. Symp.*, Berkeley, CA, USA, vol. 13, 2004, p. 21.

[2] T. T. Project. (Jun. 2020). *Tor Metrics Portal*. [Online]. Available: https://metrics.torproject.org/

[3] I. Lovecruft, G. Kadianakis, O. Bini, and N. Mathewson, "Tor guard specification,". [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/guard-spec.txt, Jun. 2020.

[4] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg, "Changing of the guards: A framework for understanding and improving entry guard selection in Tor," in *Proc. ACM Workshop Privacy Electron. Soc.*, 2012, pp. 43–54.

[5] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on Tor by realistic adversaries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 337–348.

[6] P. Winter, R. Ensafi, K. Loesing, and N. Feamster, "Identifying and characterizing Sybils in the Tor network," in *Proc. USENIX Secur. Symp.*, 2016, pp. 1169–1185.

[7] N. Danner, S. Defabbia-Kane, D. Krizanc, and M. Liberatore, "Effectiveness and detection of denial-of-service attacks in Tor," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 3, pp. 1–25, Nov. 2012.

[8] K. Kohls, K. Jansen, D. Rupprecht, T. Holz, and C. Pöpper, "On the challenges of geographical avoidance for Tor," in *Proc. 26th Symp. Netw. Distrib. Syst. Secur. (NDSS)*, Feb. 2019.

[9] R. Jansen, T. Vaidya, and M. Sherr, "Point break: A study of bandwidth denial-of-service attacks against Tor," in *Proc. 28th USENIX Conf. Secur. Symp.*, 2019, pp. 1823–1840.

[10] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos, "Compromising anonymity using packet spinning," in *Proc. Int. Conf. Inf. Secur.*, 2008, pp. 161–174.

[11] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the Tor network," in *Proc. NDSS*, 2014.

[12] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against Tor," in *Proc. ACM Workshop Privacy Electron. Soc. (WPES)*, 2007, pp. 11–20.

[13] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 92–102.

[14] Q. Tan, G. Yue, J. Shi, X. Wang, B. Fang, and Z. Tian, "Toward a comprehensive insight into the eclipse attacks of Tor hidden services," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1584–1593, Apr. 2019.

[15] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and mitigating AS-level adversaries against Tor," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–12.

[16] L. Vanbever, O. Li, J. Rexford, and P. Mittal, "Anonymity on quicksand: Using BGP to compromise TOR," in *Proc. 13th ACM Workshop Hot Topics Netw.*, 2014, p. 14.

[17] G. Wan, A. Johnson, R. Wails, S. Wagh, and P. Mittal, "Guard placement attacks on path selection algorithms for Tor," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 4, pp. 272–291, Oct. 2019.

[18] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on Tor using deep learning," in *Proc. 2018 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1962–1976.

[19] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *Proc. 7th Int. Conf. Privacy Enhancing Technol.*, Ottawa, ON, Canada. Berlin, Germany: Springer-Verlag, 2007, pp. 167–183.

[20] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proc. IEEE Symp. Secur. Privacy*, May 2005, pp. 183–195.

[21] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, 2011, pp. 215–226.

[22] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, p. 13, 2010.

[23] Z. Ling *et al.*, "A new cell-counting-based attack against Tor," *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 1245–1261, Sep. 2012.

[24] Y. Sun *et al.*, "RAPTOR: Routing attacks on privacy in Tor," in *Proc. 24th USENIX Secur. Symp. (USENIX Secur.)*, 2015, pp. 271–286.

[25] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining how the great firewall discovers hidden circumvention servers," in *Proc. Internet Meas. Conf.*, Oct. 2015, pp. 445–458.

[26] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the great firewall of China over space and time," *Proc. Privacy Enhancing Technol.*, vol. 2015, no. 1, pp. 61–76, Apr. 2015.

[27] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 57–69.

[28] S. Frolov, J. Wampler, and E. Wustrow, "Detecting probe-resistant proxies," in *Proc. 27th Symp. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–17.

[29] T. T. Project, *Tor Control Protocol*, Tor Project, Jun. 2020. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/control-spec.txt

[30] Ipswitch. (Jun. 2020). *Imacros for Firefox*. [Online]. Available: https://addons.mozilla.org/en-U.S./firefox/addon/imacros-for-firefox/

[31] Alexa. (Jun. 2020). *The Top 500 Sites on the Web*. [Online]. Available: http://www.alexa.com/topsites

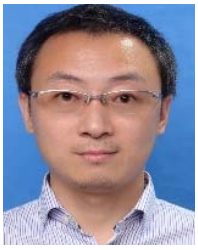
[32] I. Azureus Software. (Jun. 2020). *Vuze Bittorrent Client*. [Online]. Available: http://www.vuze.com/

Qingfeng Tan (Member, IEEE) received the Ph.D. degree in information security from the University of Chinese Academy of Sciences, Beijing, China, in 2017. He is currently an Associate Professor with the Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, China. He has published over 30 articles in reputable conferences and journals. His current research interests include computer networks and network security. He is a member of the China Computer Federation.

Xuebin Wang was born in 1986. He is currently pursuing the Ph.D. degree with the University of Chinese Academy of Sciences, Beijing, China. He is also with the Institute of Information Engineering, Chinese Academy of Sciences, and a member of the China Computer Federation. His current research interests include information security and cryptocurrencies privacy.

Wei Shi (Member, IEEE) received the bachelor's degree in computer engineering from the Harbin Institute of Technology, China, and the master's and Ph.D. degrees in computer science from Carleton University, Ottawa, ON, Canada. She is currently a Professor with the School of Information Technology, Faculty of Engineering and Design, Carleton University. She is specialized in algorithm design and analysis in distributed systems, such as distributed data center and clouds, edge networks, mobile agents and actuator systems, and wireless sensor networks. She has also been conducting research in data privacy and big data analytics. She has published over 80 articles in reputable conferences and journals. She is a professional engineer licensed in Ontario, Canada.

Jian Tang (Fellow, IEEE) received the Ph.D. degree in computer science from Arizona State University in 2006. He is currently with Midea Group. He has published over 170 papers in premier journals and conferences. His research interests include AI, the IoT, wireless networking, mobile computing, and big data systems. He is an ACM Distinguished Member. He received an NSF CAREER Award in 2009. He also received several best paper awards, including the 2019 William R. Bennett Prize and the 2019 Technical Committee on Big Data (TCBD) Best Journal Paper Award from the IEEE Communications Society (ComSoc), the 2016 Best Vehicular Electronics Paper Award from the IEEE Vehicular Technology Society (VTS), and the Best Paper Awards from the 2014 IEEE International Conference on Communications (ICC) and the 2015 IEEE Global Communications Conference (Globecom). He has served as an Editor for several IEEE journals, including IEEE TRANSACTIONS ON BIG DATA and IEEE TRANSACTIONS ON MOBILE COMPUTING. He has also served as a TPC Co-Chair for a few international conferences, including the IEEE/ACM IWQoS 2019, MobiQuitous 2018, and IEEE iThings 2015; the TPC Vice Chair for the INFOCOM'2019; and an Area TPC Chair for INFOCOM 2017–2018. He is also an IEEE VTS Distinguished Lecturer and the Chair of the Communications Switching and Routing Committee of IEEE ComSoc.



Zhihong Tian (Senior Member, IEEE) is currently a Professor and the Dean of the Cyberspace Institute of Advanced Technology, Guangzhou University, Guangdong, China. He is also a Distinguished Professor at Guangdong Province Universities and Colleges Pearl River Scholar. He is a part-time Professor at Carlton University, Ottawa, Canada. Previously, he served in different academic and administrative positions at the Harbin Institute of Technology. His research has been supported in part by the National Natural Science Foundation of China, the National Key Research and Development Plan of China, the National High-Tech Research and Development Program of China (863 Program), and the National Basic Research Program of China (973 Program). He has authored over 200 journals and conference papers in these areas. His research interests include computer networks and cyberspace security. He is a Senior Member of the China Computer Federation. He has served as a member, the chair, and the general chair of a number of international conferences.