Protocol-level Hidden Server Discovery

Zhen Ling*†, Junzhou Luo*, Kui Wu† and Xinwen Fu‡
*Southeast University, Email: {zhenling, jluo}@seu.edu.cn
†University of Victoria, Email: wkui@cs.uvic.ca
‡University of Massachusetts Lowell, Email: xinwenfu@cs.uml.edu

***Abstract*—Tor hidden services are commonly used to provide a TCP based service to users without exposing the hidden server's IP address in order to achieve anonymity and anti-censorship. However, hidden services are currently abused in various ways. Illegal content such as child pornography has been discovered on various Tor hidden servers. In this paper, we propose a protocol-level hidden server discovery approach to locate the Tor hidden server that hosts the illegal website. We investigate the Tor hidden server protocol and develop a hidden server discovery system, which consists of a Tor client, a Tor rendezvous point, and several Tor entry onion routers. We manipulate Tor cells, the basic transmission unit over Tor, at the Tor rendezvous point to generate a protocol-level feature at the entry onion routers. Once our controlled entry onion routers detect such a feature, we can confirm the IP address of the hidden server. We conduct extensive analysis and experiments to demonstrate the feasibility and effectiveness of our approach.**

***Keywords*-Anonymous Communication, Tor, Hidden Service**

I. Introduction

Tor is a broadly-used low-latency anonymous communication system which supports TCP applications over the Internet [1]. It provides users with anonymity service, helps fight against Internet censorship, and supports *hidden services* to preserve the anonymity of web services [2]. Tor was deployed in late 2003 and comprised hundreds of onion routers, while hidden services were released in early 2004. Due to increasingly high demand of privacy protection, the Tor network has seen steady growth, consisting of around 3000 volunteer based Tor onion routers as of July 2012.

Unfortunately, hidden services have been misused or abused for various illegal purposes. They can host botnet and illegal web contents such as drug trading information [3] and pornography. The consequence is severe: If botnets are deployed with the hidden service over Tor [4], they are hard to take down because of the anonymity protected with the hidden service; if a hidden service hosts child pornography website [5] [6][1], the hidden service actually blindly provides a protection to the illegal content in most countries.

Existing research work [7], [8] has been carried out to investigate the attacks which can locate the Tor hidden server. The approach in [7] is based on traffic analysis. The attacker controls both malicious client and entry routers, and uses a simple passive timing analysis, i.e., cell counting, to discover the same patten in the client's traffic and the entry's traffic to identify the hidden service at the entry side. Murdoch [8] presented a clock skew based approach to identifying whether or not a given Tor node is a hidden server. The attacker evaluates the load of a given Tor node. Since the server's temperature would increase while its workload rises, the attacker can identify whether the node is the attacked hidden service by estimating its temperature through measuring the clock skew. Nevertheless, the attacks based on traffic analysis may suffer a high rate of false positives due to various factors, such as Internet traffic dynamics, the load of Tor nodes, and the large number of cells for the purpose of statistical traffic analysis.

In this paper, we propose a protocol-level discovery approach to locating a hidden server by utilizing Tor protocol features. We (law enforcement) control a Tor client, a rendezvous point, several entry onion routers, and a central server. The discovery takes three phases. In Phase I, our Tor client continues to create circuits to the hidden server until one of our entry routers sees a special combination of cells of different types. Such a combination, denoted as a protocol-level feature, comes from the Tor protocol that creates the circuits between a client and a hidden server. However, even if our entry router observes such a feature, it may result from other clients that create circuits to a hidden server through our entry router. Phase II is to confirm that the hidden server chooses our entry router. We manipulate cells from our client to incur a special decryption error at the hidden server, which will destroy all circuits to the client. If our entry router sees the *destroy cell*, we know that our entry router is chosen by the hidden server. Phase III is used to correlate all the events above. In Phases I and II, related information of these cells observed at our clients, entry routers, and the rendezvous point has been sent to our central server. In Phase III, we exploit the timing information of these cells, and identify the correlation to confirm that the target hidden server is behind our entry router. In this way, we have located the hidden server.

Our approach has several unique advantages. First, it is easy to deploy our detection system. Second, compared to traffic analysis based methods, our approach is significantly faster, fully automatic and can quickly locate the hidden server using only several cells. Third, our approach is accurate with an observed detection rate of 100% and has an observed low false positive of 0%. Fourth, our approach works on the protocol level and is oblivious to traffic patterns; it is more general and can be used to identify malicious hidden services. A hidden server may also use its trusted entry routers or Tor bridges as the first hop into the Tor network. We discuss those complicated cases of tracking hidden servers in Section VI.

[1]The authors believe that illegal content is hosted at this hidden website although we did not dig it because of legal concerns.

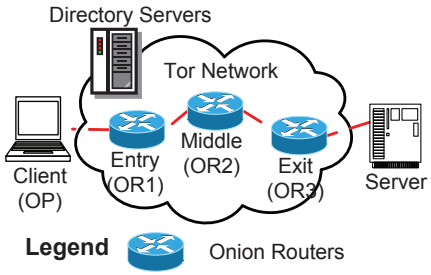

Fig. 1. Tor network

The rest of the paper is organized as follows. In Section II, we introduce the components of Tor, its basic operations and the protocol of hidden service. In Section III, we present the basic idea of our approach and then elaborate the algorithm. In Section IV, we analyze the effectiveness of the approach. In Section V, we show experimental results on Tor, and we discuss complicated cases of tracking hidden servers in Section VI. Related work is reviewed in Section VII. The paper is concluded in Section VIII.

II. Background

In this section, we first introduce the Tor network and then present its basic operations and the protocol of hidden service.

A. Components of the Tor Network

Figure 1 illustrates the basic architecture of Tor network. The following components are involved in the typical use of Tor network:

- *Tor clients*. A Tor client installs a local software referred to as *onion proxy* (OP), which packs application data into equal-sized *cells* (512 bytes) and delivers them into Tor network. A cell is the basic transmission unit of Tor.
- *Onion routers* (OR). The onion routers relay the cells on behalf of Tor client and server.
- *Directory servers*. Directory servers hold the information of onion routers and *hidden services*, such as the public keys of routers and hidden servers.
- *Application servers*. It supports TCP applications such as a web service and an IRC service.

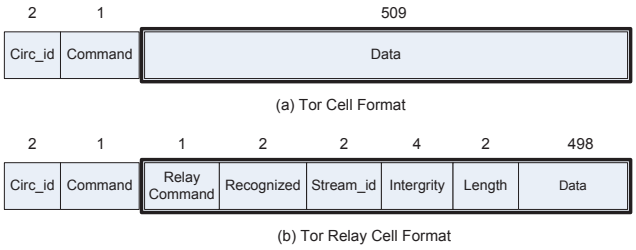

Fig. 2. Tor cell format [1]

Figure 2 illustrates the format of the Tor cell. The first three-bye header of the Tor cell is not encrypted so that the Tor onion router can read this header. The first two bytes is the circuit ID, while the third byte, is used to indicate the specific command of this cell. We categorize the Tor cell into two types: the *control* cell as illustrated in Figure 2 (a) and the *relay* cell as shown in Figure 2 (b). The filed *Command* of a control cell can be, for instance, *CELL_CREATE*/*CELL_CREATE_FAST* or *CELL_CREATED*/*CELL_CREATED_FAST*, employed for establishing a new circuit; and *CELL_DESTROY*, used for tearing down a circuit. The filed *Command* of *relay* cell is *CELL_RELAY* that is used to relay the application data. In addition, there are numerous types of relay commands (Relay Command), and its format is like *RELAY_COMMAND_X* where "X" is a word. In our paper, when we mention the *RELAY_COMMAND_X* cell, it indicates a *relay* cell and the content of this cell is onion-like encrypted. We will elaborate these commands further in later sections when we discuss the Tor operations from the perspective of protocol-level.

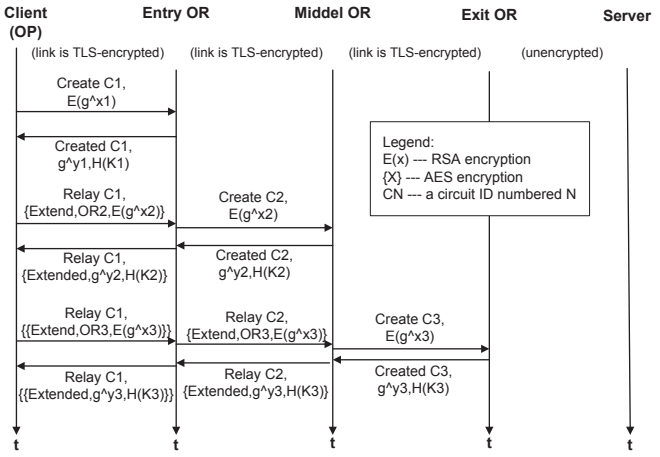

Fig. 3. Circuit creation

B. Circuit Selection and Creation

To communicate with an application server via Tor, a Tor client first downloads all of the onion router information from the directory server and uses source routing by choosing a series of onion routers as a route. We call the sequence of onion routers as the *path* through Tor. The number of onion routers is called the *path length*. In the Tor network, a path is also called a circuit, thus we use path/circuit interchangeably in this paper. We employ the default path length of 3 as an example in Figure 1 to show how a path is chosen. The client first selects an appropriate *exit* onion router (OR3), which should have an exit policy supporting the relay of the TCP stream from the client. Then, the client chooses a proper *entry* onion router (OR1) (also referred to as *entry guard*) and a middle onion router (OR2). After that, the client initiates the procedure of creating a circuit incrementally, one hop at a time. Eventually, the client can communicate with the remote server through this circuit, i.e., $OR1 \rightarrow OR2 \rightarrow OR3$.

Figure 3 illustrates the procedure that a client builds a circuit. As shown in Figure 3, the client first establishes a TLS connection with *entry* router using the TLS protocol. Then the client sends a *CELL_CREATE* cell through the TLS connection and uses the *Diffie-Hellman* (DH) handshake protocol to negotiate a base key $K_1 = g^{xy}$ with entry onion router, which responds with a *CELL_CREATED* cell. Note that the $H(K_1)$ is the hash value of K_1 in Figure 3. From this base key material, a forward symmetric key kf_1 and a backward symmetric key kb_1 are generated. In this way, the first hop of this circuit, denoted as $C1$, is created. Similarly, the client extends the circuit to include the second hop ($C2$) and the third hop ($C3$) of the circuit.

Figure 4 shows the procedure of the data transmission over the circuit. Once the circuit is established, the client sends a

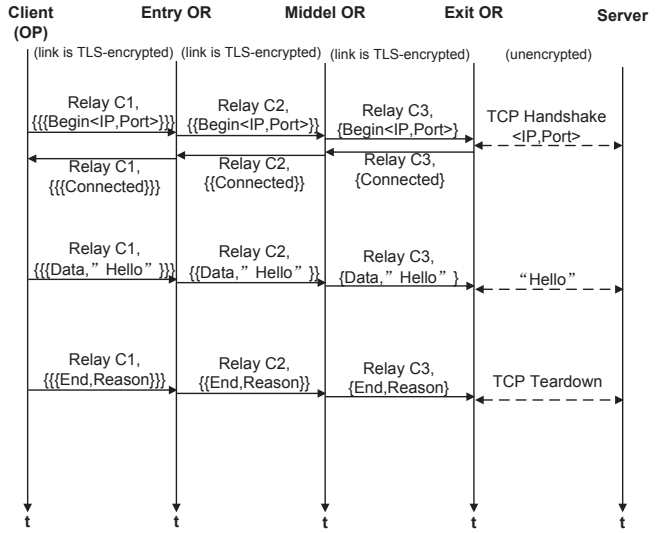

Fig. 4. Data transmission over the circuit

RELAY_COMMAND_BEGIN cell to the exit onion router, and the cell is encrypted as $\{\{\{Begin < IP, Port >\}_{kf_1}\}_{kf_2}\}_{kf_3}$, where the subscript refers to the key used for encryption of one specific onion skin. The three layers of onion skin are removed one by one each time the cell traverses an onion router through the circuit. When exit onion router removes the last onion skin by decryption, it recognizes that the request intends to open a TCP stream to a port at the destination IP pointing to the remote server. Therefore, the exit onion router acts as a proxy, builds a TCP connection with the server, and sends a *RELAY_COMMAND_CONNECTED* cell back to the client. Then the client can download the file.

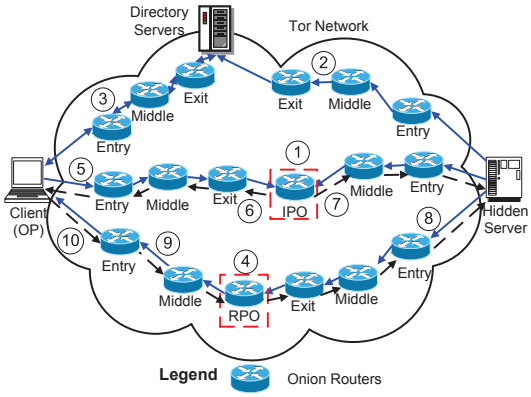

Fig. 5. Tor hidden service

C. How Does the Hidden Service Work

A hidden service involves six participants, including a Tor client, the directory server, onion routers, a rendezvous point, an introduction point, and a hidden server. Since the first three participants have been described above, we will introduce the functionality of the rendezvous point and the introduction point and how they work together to support a hidden service.

- *Introduction point* (IPO). An introduction point is selected and published in the directory server associated with the descriptor of the hidden service by the hidden server. Once the introduction point is decided, the hidden server establishes a circuit to the introduction point. Then the introduction point plays as the front interface to Tor client and waits until a Tor client creates a three-hop circuit to the introduction point and forwards the request data *from the Tor client side circuit to the hidden server side circuit.*
- *Rendezvous point* (RPO). A rendezvous point is chosen by the Tor client. Both the Tor client and the hidden server will establish a three-hop circuit to the RPO, which acts as a message relay to transmit the application data *between the hidden server side circuit and the Tor client side circuit.*
- *Hidden server*. A hidden server provides various TCP applications such as web server and IRC server. It could be deployed over OP or OR.

Figure 5 depicts the procedure of establishing a connection between the Tor client and the specific hidden server.

1) The hidden server first selects several onion routers as introduction points and builds the circuits to these introduction points. To build a circuit, the hidden server will send the *RELAY_COMMAND_ESTABLISH_INTRO* cell to an introduction point, and the introduction point will reply with the *RELAY_COMMAND_INTRO_ESTABLISHED* cell to inform the hidden server that the circuit is established.
2) Once the circuits to introduction points are established, the hidden server establishes a circuit to the directory server and advertises the service descriptor to the directory server, including the public key of the hidden server and the information regarding the introduction points. Then the owner of the hidden server can post the onion address[2] in a public place to attract users to access the hidden service via Tor.
3) When a Tor client obtains the onion address, the client creates a circuit to the directory server and fetches the relevant information advertised by the hidden service. Then the client learns the introduction points of the hidden service.
4) The client selects a rendezvous point and builds a circuit to the rendezvous point. The client will send a *RELAY_COMMAND_ESTABLISH_RENDEZVOUS* cell, which carries a rendezvous cookie, to the rendezvous point, which replies with a *RELAY_COMMAND_RENDEZVOUS_ESTABLISHED* cell to indicate the successful circuit establishment.
5) The client creates a three-hop circuit to one of the introduction points and transmits a *RELAY_COMMAND_INTRODUCE1* cell to the chosen introduction point. The cell carries the information such as the rendezvous point, rendezvous cookie and the *Diffie-Hellman* data g^x generated by the Tor client.
6) Once the introduction point receives the *RELAY_COMMAND_INTRODUCE1* cell, it replies with a *RELAY_COMMAND_INTRODUCE_ACK* cell to the client. After the client receives this ACK cell, it tears down this circuit to the introduction point.
7) The introduction point repacks the *RE-*

[2]Onion address is generated by the hidden server. It is a hostname of the form “x.onion”, where “x” consists of 16 *random* characters.

LAY_COMMAND_INTRODUCE1 cell into a *RELAY_COMMAND_INTRODUCE2* cell, and then sends the *RELAY_COMMAND_INTRODUCE2* cell to the hidden server.

8) After the hidden server receives the *RELAY_COMMAND_INTRODUCE2* cell, it knows the information of the rendezvous point, rendezvous cookie and *Diffie-Hellman* data g^x. The hidden server can generate the *Diffie-Hellman* data g^y and derive the key $K = g^{xy}$. This key is used for end-to-end encryption between client and server. Then the hidden server builds a circuit to the rendezvous point and sends a *RELAY_COMMAND_RENDEZVOUS1* cell to the rendezvous point via the circuit. The cell includes the key data g^y, the hash value of the key $H(K)$ and the rendezvous cookie.
9) The rendezvous point obtains the *RELAY_COMMAND_RENDEZVOUS1* cell and compares the rendezvous cookie from the cell and the one from the Tor client. Once the rendezvous cookies are matched, the rendezvous point removes the rendezvous cookie from the *RELAY_COMMAND_RENDEZVOUS1* cell and repacks the rest data into *RELAY_COMMAND_RENDEZVOUS2* cell, and then forwards the cell to the client.
10) When the Tor client receives the *RELAY_COMMAND_RENDEZVOUS2* cell, it can generate the key $K = g^{xy}$ using g^y and verify it based on $H(K)$. The key is used to encrypt the data between client and server and is the same as the one generated by the hidden server at Step 8. In this way, the client and hidden server complete the handshake. Then the client sends a *RELAY_COMMAND_BEGIN* cell to establish a stream to the hidden server via the six-hop circuit.

From the above procedure, the Tor client only knows the introduction point instead of the hidden server directly, and the hidden server only knows the rendezvous point instead of the Tor client directly. In addition, the introduction point acts as the front interface to the Tor client for service query and request only, and it does not get involved any more once the six-hop circuit passing through the rendezvous point is established for data communication between the Tor client and the hidden server. Since either introduction point or rendezvous point knows neither the location of Tor client nor the location of hidden server, anonymous web service is hence achieved. Note that in the Tor network, only the entry onion router knows IP addresses of hidden servers.

III. Protocol-Level Hidden Server Discovery

In this section, we first introduce the basic idea of discovering a Tor hidden server and then present detailed algorithms.

A. Basic Idea

Since only entry onion routers may know the real IP address of a hidden server, we assume that we are able to control several entry onion routers[3]. In addition, we need a client and rendezvous point to cooperate with entry onion routers. A central server is used to record information of related cells forwarded from the Tor client, entry onion routers, and rendezvous point.

The discovery is conducted as follows: (i) Our Tor client obtains the introduction point information from the directory server, and builds a circuit to the introduction point and also reports the circuit creation to our central server to indicate the start of discovery. (ii) The hidden server also establishes a circuit to the rendezvous point. If the hidden server chooses our entry router, our entry router will see a special combination of cells of different types, denoted as protocol-level features, during the creation of those circuits. However, such protocol-level features do not necessarily imply the hidden server chooses our entry router. We perform the following actions to confirm it is a true positive. (iii) Once the connection is established between the client and the hidden server, the client will send cells that contain application data to the hidden server as illustrated in Step 10 of Figure 5. **Our rendezvous point manipulates an appropriate cell** [9], and forwards the mangled cell to the hidden server. The rendezvous point also reports this cell to the central server. (iv) The mangled cell arrives at the hidden server. Since the hidden server cannot correctly decrypt the manipulated cell, it will destroy the circuit between the client and hidden server by sending a **destroy cell** to the client. This cell traverses along the circuit to the client. The rendezvous point will detect it and report it to the central server to indicate the end of the discovery process. In addition, our controlled entry onion routers will report to the central server immediately when a destroy cell is received. (v) To determine if the hidden server chooses one of our entry onion routers, the central server searches for correlation between the time when the rendezvous point sends the manipulated cell, the time when the rendezvous point receives the destroy cell, and the time when the entry onion router receives the destroy cell. Since the entry onion router knows the IP address of the circuit creator, once such time correlation is found, we can identify the hidden server. Figure 6 illustrates the work flow of the protocol-level hidden server discovery approach.

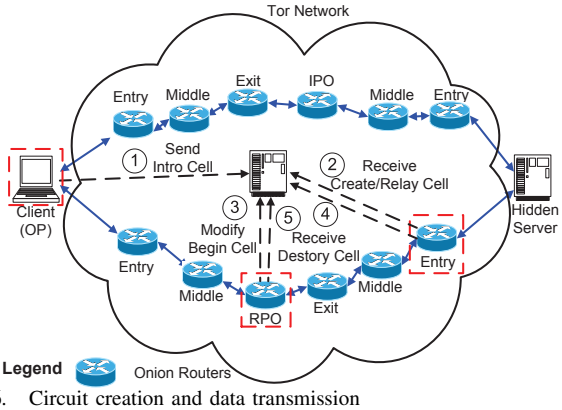

Fig. 6. Circuit creation and data transmission

[3]The same assumption was made in virtually all attacks towards the Tor network. This is reasonable because onion routers are set up by volunteers.

B. Details of Protocol-level Hidden Server Discovery

The hidden server discovery process can be divided into three phases. Phase I: Presumably identify the hidden server - the client continues to create circuits to the hidden server until one of our entry routers sees a special combination of cells of different types, i.e., protocol-level features. Phase II: Verify the hidden server - our rendezvous point manipulates a data cell and creates a decryption error at the hidden server, which has to send out a destroy cell to destroy the circuit. The destroy cell can be recognized by our entry router if the hidden server uses our entry router. Phase III: Conclude by time correlation - the central server uses timing information of collected cells to correlate the unique sequence of events of our discovery actions and draw a conclusion if the presumably identified entry router is chosen by the hidden server and locate the hidden server accordingly.

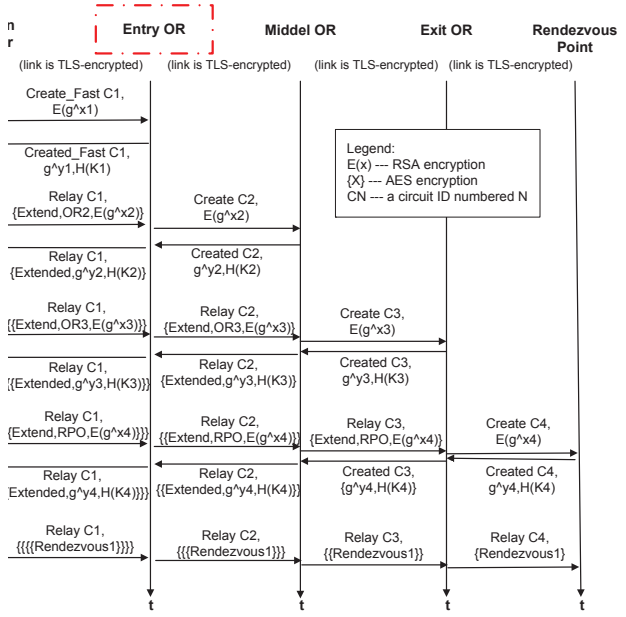

Fig. 7. Hidden server creating a circuit to the rendezvous point

Phase I: Presumably identify the hidden server. Recall that the Tor client can derive the introduction point information from directory server as illustrated in Step 3 of Figure 5. After the client establishes a circuit to the rendezvous point, the client will send a *RELAY_COMMAND_INTRODUCE1* cell to the introduction point in order to negotiate the *Diffie-Hellman* key with the hidden server. We select this cell as a begin-sign of our discovery approach and send this information to the central server.

When the hidden server receives the *RELAY_COMMAND_INTRODUCE2* cell forwarded from the introduction point, the hidden server will build a circuit to the rendezvous point as shown in Step 8 of Figure 5. Once the circuit is established, the hidden server will promptly send a *RELAY_COMMAND_RENDEZVOUS1* cell to the rendezvous point. The circuit creation process is illustrated in Figure 7. As we can see from Figure 7, the entry onion router will receive **one** *CELL_CREATE_FAST* cell and **four** *CELL_RELAY* cells, including a *RELAY_COMMAND_INTRODUCE2* cell, and relay **one** *CELL_CREATED_FAST* cell and **three** *CELL_RELAY* cells to the hidden server.[4] Let us denote the cells as **protocol-level feature**. Moreover, the entry onion router will report the related information of each cell, including the cell type, circuit ID, and the IP address of circuit creator, to the central server. Furthermore, after the rendezvous point receives the *RELAY_COMMAND_RENDEZVOUS1* cell, the rendezvous point needs to report the information to the central server immediately.

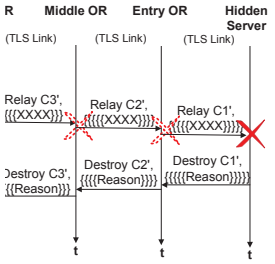

Fig. 8. Modify a cell at the RPO

Phase II: Verify the hidden server. Recall that after the rendezvous point receives a *RELAY_COMMAND_RENDEZVOUS1* cell from the hidden server, it repacks it into a *RELAY_COMMAND_RENDEZVOUS2* cell and forwards it to the client as illustrated in Steps 8 and 9 of Figure 5. The client will send the *RELAY_COMMAND_BEGIN* cell to the hidden server through the circuit in order to open a stream between the client and the hidden server. Then our controlled rendezvous point can detect this special cell based on the hidden service protocol, even if the rendezvous point cannot decrypt the cell and obtain the content. Once the rendezvous point catches this cell, we modify one bit of the cell and forward it to the hidden server. Due to the lack of integrity verification, other onion routers cannot detect the manipulated cell. For detection purpose, the rendezvous point needs to send the timestamp of the manipulated cell to the central server. Figure 8 illustrates the procedure of modifying the cell at the rendezvous point.

When the manipulated cell reaches the hidden server, the hidden server cannot correctly recognize this cell. According to the design of Tor, the hidden server will tear down the circuit between the client and hidden server by sending a *CELL_Destroy* cell promptly. The controlled entry onion router will be the first router that receives this cell, and it reports the cell type, the timestamp of the cell, circuit ID and the source IP address of the cell to the central server. Moreover, the rendezvous point will receive this *CELL_Destroy* cell as well. The rendezvous point also needs to report the timestamp of this cell to the central server.

Phase III: Conclude by time correlation. Since the central server may receive many cells from our entry onion routers, we carefully choose several appropriate feature cells and use them to filter out useless cell information. The central server records the source IP address of each cell, circuit ID,

[4]The *CELL_CREATE_FAST* cell and *CELL_CREATED_FAST* are used in the first hop creation of hidden server instead of *CELL_CREATE* and *CELL_CREATED*.

cell type, and timing information of the cell. Specifically, the *RELAY_COMMAND_INTRODUCE1* cell from the client is used as the **begin-sign cell of Phase I**, and the *RELAY_COMMAND_RENDEZVOUS1* cell from the rendezvous point is used as the **end-sign cell of Phase I**. Then, the manipulated *RELAY_COMMAND_BEGIN* cell sent from the rendezvous point is selected as the **begin-sign cell of Phase II**. After the central server receives the manipulated cell information, it can first filter out the circuits along which our entry onion routers are not chosen as the first router, by counting the number of *CELL_CREATE*, *CELL_CREATED* and *CELL_RELAY* cells based on the Tor circuit creation protocol. Eventually, we choose the *CELL_DESTROY* cell information received from the rendezvous point as the **end-sign cell of Phase II**. Finally, we search the remaining circuit information and find out whether we can detect a circuit that receives a *CELL_DESTROY* cell during the time period between the *begin-sign cell* and the *end-sign cell* of phase II. If such a circuit is detected, we find the IP address of the hidden server.

C. Make the Discovery Automatic

We want to emphasize that our discovery of hidden server is conducted fully automatically. The central server builds tcp connections to the Tor client, entry onion routers and rendezvous points, and receives the information from those nodes. It is a multiple thread program, maintains a list of various cell information and conducts the correlation to discover a hidden server. For a realistic network forensic practice, an automatic discovery is a necessity.

We discuss how the central server code works to discover a hidden server in detail as follow. The central server starts to record cell information once it receives the *begin-sign cell of Phase I* from the Tor client and starts the discovery process. Then, the central server receives cell information from entry onion routers, no matter whether or not the hidden server selects our entry onion routers as the first hop. The central server will record the cell type, circuit ID and the source IP address of a cell into a list. When the central server receives the *end-sign cell of Phase I* from our rendezvous point, it searches the list and finds the candidate circuits that match the **protocol feature** discussed in Phase I. Subsequently, the central server receives the *begin-sign cell of Phase II* from the rendezvous point and records the cell type, circuit ID, and the timing of the begin-sign cell, denoted as T_b. If our entry onion routers are selected by the hidden server as the first hop, the central server receives a *CELL_DESTROY* cell from the entry onion router and records the timing of the *CELL_DESTROY* cell, denoted as T_d. As time goes, the central server receives the report from the rendezvous point that the *CELL_DESTROY* cell arrives at the rendezvous point and records the timing of the *CELL_DESTROY* cell, denoted as T_e. Eventually, the central server searches the list of candidate circuits in order to confirm that the *CELL_DESTROY* cell received from the entry onion router is from one of the candidate circuits. Once the circuit is found, the central server compares the timing of the *CELL_DESTROY* cell from the entry onion router by using the condition $T_b < T_d < T_e$. This timing correlation is used to confirm that the appearance of the *CELL_DESTROY* cell at the entry onion router is caused by our manipulated cell at the rendezvous point. In this way, the central server accomplishes the discovery process. Eventually, we make the entire discovery process automatic.

IV. Analysis

For our approach to be effective, the key issue is that one of our controlled entry onion routers should be selected as the entry onion router by the hidden server. In this section, we analyze the chance that the hidden server selects one of our onion routers as the entry router, and propose a strategy that can greatly increases this chance without incurring large cost.

A. Catch Probability

To evaluate the effectiveness of our discovery approach, we analyze the *catch probability*, i.e., the probability that a circuit from a hidden server selects one of our controlled onion routers as the entry onion router. To start with, we need to introduce how Tor determines a circuit among the many possible paths. Tor adopts a bandwidth weighted circuit selection algorithm in order to enhance its performance. The detailed path selection algorithm is illustrated in our technique report [10]. To be specific, the onion routers are categorized into four classes: pure entry routers (entry guards), pure exit routers, both entry and exit routers (denoted as EE routers), and neither entry nor exit routers (denoted as N-EE routers). A router is marked as an entry guard by the authoritative directory server only if its mean uptime is above the median of all knowns routers and its bandwidth is greater than $max\{median, 250KB/s\}$. The exit routers support the users' traffic to get out to the public port, such as ports 80 and 443. EE routers are those marked as both entry guard and exit router by directory servers, and N-EE routers are marked as neither of them [11]. Therefore, the entry onion router set consists of the pure entry onion routers and the EE routers.

According to the Tor onion router selection algorithm, the bandwidth of each entry onion router is weighted. Assume the bandwidth of each the onion router is labeled as $\{B_1, B_2, \ldots, B_i, B_{i+1}, \ldots, B_N\}$ and the total bandwidth of the onion routers is B. Denote the bandwidth of the pure entry onion routers, the pure exit onion routers, and the EE onion routers as B_e, B_x and B_{ee}, respectively. Then the total bandwidth of entry routers is $B_E = B_e + B_{ee}$. According to Tor, the probability that the i^{th} entry onion router is selected from the entry set is equal to the ratio of the bandwidth of the i^{th} entry router over the total weighted entry router bandwidth, that is,

$$\mathcal{P}(i^{th}\ router\ is\ selected\ as\ entry\ router) = \frac{B_i}{B_e + B_{ee} \cdot \mathcal{W}_E} \quad (1)$$

where B_i is the bandwidth of the i-th entry router and the weight $\mathcal{W}_E$ is calculated by $max\{0, 1 - \frac{B}{3(B_x+B_{ee})}\}$.

Assume that we are able to inject k computers into the Tor network and these computers are established as entry onion routers. Now we denote the bandwidth of all onion routers as a set $\{B_1, B_2, \cdots, B_k, B_{k+1}, \cdots, B_{k+N}\}$, where $\{B_1, \cdots, B_k\}$ is the bandwidth of the k newly injected routers. Assume that the new routers advertise the same bandwidth[5],

[5]The Tor project released a new version that changes the upper-bound of high bandwidth to 10MB/s on August 30, 2007.

i.e., $B_1 = B_2 = \cdots = B_k = b$. Since B is the aggregated bandwidth of all original onion routers ($B = \sum_{i=k+1}^{k+N} B_i$), the total bandwidth now becomes $B + k \cdot b$.

Following Equation (1), after we inject the k onion routers into the set of entry onion routers, the catch probability can be easily calculated as:

$$\mathcal{P}(k, b) = \frac{k \cdot b}{B'_e + B'_{ee}}, \tag{2}$$

where k is the number of injected onion routers, b is the bandwidth claimed by each injected router, $B'_e = B_e + k \cdot b$, and $B'_{ee} = B_{ee} \cdot \mathcal{W}'_E$. Note that $\mathcal{W}'_E$ is the adjusted weight and can be calculated as $max\{0, 1 - \frac{B+k \cdot b}{3(B_x + B_{ee})}\}$.

We intentionally denote the catch probability as $\mathcal{P}(k, b)$ to emphasize that the probability depends on the number of injected onion routers and the claimed bandwidth of each injected onion router.

B. How to Improve the Catch Probability

Based on Equation (2), it is easy to prove the following claims and the proof can be found in our technical report [10]:

- The catch probability increases with the number of controlled Tor entry routers, i.e.,

$$\mathcal{P}(r, b) > \mathcal{P}(k, b), \text{ where } r > k. \tag{3}$$

- The catch probability increases with the bandwidth of controlled Tor entry routers, i.e.,

$$\mathcal{P}(k, l) > \mathcal{P}(k, b), \text{ where } l > b. \tag{4}$$

- The catch probability is determined by the aggregated bandwidth contributed by the controlled Tor entry routers. That is, if $M = k \cdot b$, $M' = k' \cdot b'$, and $M \geq M'$,

$$\mathcal{P}(k, b) \geq \mathcal{P}(k', b') \tag{5}$$

The equality holds when $M = M'$.

V. Evaluation

We have implemented the proposed Tor hidden service discovery approach in Section III. In this section, we elaborate the results of the empirical evaluation of the approach. Our experimental results match the theoretical analysis presented in Section IV well.

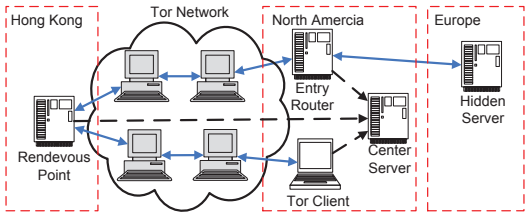

Fig. 9. Experiment setup

A. Experiment setup

Figure 9 illustrates the experiment setup for protocol-level hidden service discovery over the real-world Tor. We deployed a Tor client, entry onion router, and central server at two different campuses in the north American. We set up a rendezvous point on a PlanetLab node in Hong Kong and a hidden server on a PlanetLab node in Europe. To implement the discovery, we revised the source code of Tor at the Tor client side, rendezvous point, and entry router in order to establish the connections to our central server and send the related information. The version of Tor in our experiment is the latest stable version 0.2.2.37. The hidden server on the PlanetLab node is deployed on a Tor client and Apache server is installed as the HTTP server so as to offer hidden service. In addition, at the client side, we implemented a web client to automatically send a http request of the specific onion address, and then installed the HTTP proxy, i.e., Privoxy [12], in order to relay the http request into the OP.

B. Experiment Results

Table I gives the detection rate by our protocol-level hidden server discovery approach. To validate the effectiveness of our approach, we force our hidden server to select our entry onion routers. To this end, we use the Tor configuration file to manipulate the parameters *EntryNodes* and *StricEntryNodes* so that the hidden server will choose our 10 entry onion routers. The experiments were repeated for 1000 times for deriving the true positive rate, the probability that the hidden server is detected if it uses our entry server. In addition, to evaluate the false positive rate of our approach, we edit the configuration file to exclude our entry onion router by using the parameter *ExcludeNodes* so that the hidden server will not use our entry server and conduct the experiments for 1000 times again. One experiment lasts for around 15 seconds. Therefore, it took around 4 hours for running 1000 experiments. As we can see from Table I, the true positive rate of our approach is 100%, while the false positive rate is 0%.

TABLE I
Detection Rate of Protocol-level Hidden Server Discovery

True Positive	100%
False Positive	0%

Figure 10 illustrates the empirical cumulative distribution function (CDF) of bandwidth of all routers in the Tor network on July 23, 2012. There are 3101 onion routers in the Tor network, including 814 pure entry routers and 776 pure exit routers, 273 EE routers and 1238 N-EE routers. As we can see from this figure, the bandwidth of around 90% routers are lower than 1 MB/s.

Figure 11 illustrates the relationship between the catch probability and the number of controlled entry routers. Recall that catch probability is the probability that a hidden server selects entry servers controlled by us. We set the bandwidth of each controlled entry router as 10 MB/s. As shown in Figure 11, the catch probability increases with the number of controlled entry routers. When we control 30 entry routers, the catch probability $P(30, 10)$ is 24.42%.

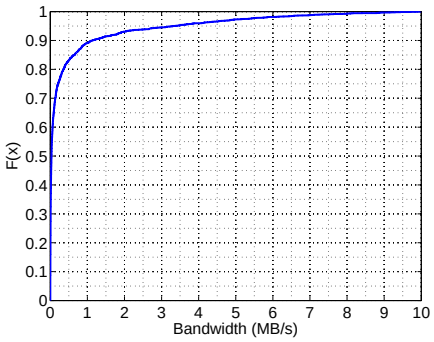

Fig. 10. Empirical CDF of bandwidth of all routers in the Tor network

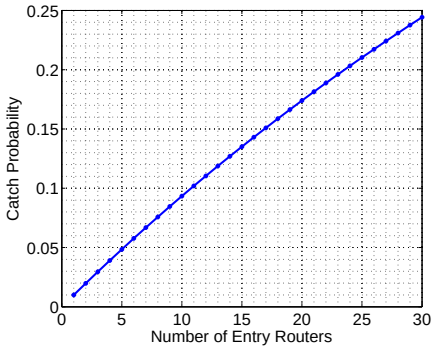

Fig. 11. Probability that a circuit selects the entry routers vs. number of controlled Tor entry routers

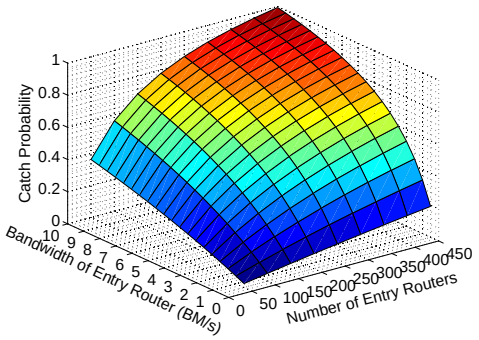

Fig. 12. Probability that at least a circuit traverses through the controlled entry routers

Figure 12 shows the relationship between the catch probability, the number of controlled entry routers and the bandwidth of the entry router. This figure demonstrates that the catch probability increases quickly with the number of controlled entry routers and the bandwidth of each entry router. When we control 450 entry routers of 10 MB/s bandwidth, the catch probability approaches 100%, i.e., $P(450 * 10) \approx 100\%$. The current Tor network has 776 pure entry routers and 273 EE routers. After inserting 450 entry routers into the Tor network, the percentage of our entry routers in the entire Tor entry router is around $450/(776 + 273 + 450) \approx 30\%$.

VI. Discussion

There are various complicated cases in discovering a hidden server via our protocol-level discovery approach. Complicated case I: in the current implementation of Tor, the hidden server maintains an entry guard list and assigns a random expiration time from 30 to 60 days to each entry guard [13]. If an entry guard is down or its life time expires, the hidden server will pick up a new entry guard to ensure that there are at least three online entry guards in the list. Therefore, the target hidden server will be safe until it selects our entry routers. This might take a long time given the lifetime of an entry guard for the hidden server. As a result, we should increase the proportion of our controlled entry guards in the entire entry routers so as to raise the probability that the target hidden server selects our entry guards. Once the hidden server chooses one of our entry routers, it will use our entry router in the following 30-60 days and will be exposed by our discovery approach.

Another complicated case is that the operator of a hidden server can choose three their trusted entry guards or Tor bridges[6] to avoid choosing our surveillance entry guards. To address this issue, we can employ two steps to discover the hidden server. In the first step, we can identify these entry routers and Tor bridges by simply revising our discovery approach. Recall in Phase I of our original approach, if our controlled middle router is selected by the hidden server side circuit, the middle router will receive **one** *CELL_CREATE* cell and **three** *CELL_RELAY* cells, including a *RELAY_COMMAND_INTRODUCE2* cell, and relay **one** *CELL_CREATED* cell and **two** *CELL_RELAY* cells to the hidden server as we can see from Figure 7. By using this new protocol feature, we can verify the location of our controlled middle onion router and identify the IP address of the suspect entry router or bridge. In the second step, after we find the first onion router of hidden server, we can take control of this node and apply our original approach again to trace the real IP address of the hidden server.

The two-steps discovery approach above can also be used to tackle Complicated case I. We can first identify the current entry router employed by a hidden server and then request this entry router to collaborate and locate the hidden server.

[6]Tor bridges are a type of hidden onion routers that are not public in the directory server. They are used as a first hop to connect to the Tor core network in the censorship region.

VII. Related Work

Existing work has explored the issues of locating the hidden server [7], [8], [14]. Øverlier and Syverson proposed the packet counting based traffic analysis to identify the hidden server at the entry onion router. Zhang *et al.* [14] investigated the HTTP features of hidden server that hosts a website. They used the HTTP features to identify the hidden server at the entry onion router. All those methods are based on traffic analysis, which may suffer a high rate of false positives due to various factors. They also need to analyze a large number of cells for the statistical analysis of traffic. Murdoch [8] employed a clock skew based approach to check if a given Tor node is a hidden server. Our approach is largely different from the existing approaches. It is based on protocol-level hidden server discovery, which does not depend on traffic analysis and thus is more general and effective.

There has been much research on degrading anonymous communication through various anonymous networks. Existing traffic analysis attacks against anonymous communication can be largely categorized into two groups: passive traffic analysis and active watermarking techniques. With passive traffic analysis techniques, the attackers record the traffic passively and identify the similarity between a server's outbound traffic and a client's inbound traffic. For example, Zhu *et al.* in [15] proposed the scheme of using mutual information for the similarity measurement. Liberatore and Levine [16] examined the packet sizes of HTTP traffic transmitted over persistent connection or tunneled via SSH port forwarding

to identify the web pages. Wright *et al.* [17] investigated the statistical distribution of packet sizes in encrypted Voice over IP (VoIP) connections and identified the language spoken based on the distribution in each conversation. Ling *et al.* [18] explored the RTT of the traffic over the single proxy based anonymous system in order to identify the websites visited by the clients. Dyer *et al.* [19] investigated the basic information of traffic over the single proxy based anonymous system and showed that the simple features of the traffic can degrade the anonymity of the system.

The active watermarking techniques intend to embed specific secret signal(or marks) into the target traffic [20], [21], [22]. Such techniques can reduce the false positive rate significantly if the signal is long enough and does not require massive training study of traffic cross correlation as required in passive traffic analysis. For instance, Murdoch et al. [23] investigated the timing based threats on Tor by using some compromised Tor nodes. Fu *et al*. [24] studied a flow marking scheme. Yu *et al*. [21] proposed a direct sequence spread spectrum (DSSS) based traceback technique, which could be used to trace users of an anonymous communication network. In this technique, attackers modulate a victim's traffic flow using a secret PN code. Ling *et al.* [25] proposed the cell counter based attack against Tor, in which an attacker embeds a signal into the variation of cell counter of the target traffic at the exit onion router. Houmansadr and Borisov [26] investigated a scalable watermark technique to encode the watermarks by changing the locations of packets within selected time slots.

VIII. Conclusion

The hidden service over Tor is a double-edged sword. While hidden servers preserve the anonymity of the services, they technically protect malicious users and organizations who host illegal contents such as drug trading information and child pornography. A system that tracks down a hidden service can effectively deter malicious users from abusing the Tor network for illegal usage. In this paper, we design, implement, and evaluate such a system. Our method augments the arsenal of existing detection tools, but it is unique in that we do not rely on conventional time consuming traffic analysis or watermarking techniques. Instead, we aim directly at the Tor protocols for hidden services and utilize the protocol-level features to identify a hidden server. This method has very accurate detection rate and is generally applicable for the detection of any hidden service. We also expect the debate of detrimental sides of anonymous communication to continue in the long run, and hope to give a choice to law enforcement for tracking notorious hidden services.

Acknowledgments

This work is supported by National Key Basic Research Program of China under Grants No. 2010CB328104, National Natural Science Foundation of China under Grants No. 61272054, 61070161, and 61003257, Natural Sciences and Engineering Research Council of Canada (NSERC), US NSF grants 1116644, 0942113, 0958477 and 0943479, China National Key Technology R&D Program under Grants No. 2010BAI88B03 and 2011BAK21B02, China National Science and Technology Major Project under grants No. 2010ZX01044-001-001, China Specialized Research Fund for the Doctoral Program of Higher Education under Grants No. 20110092130002, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK2008030, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201, and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No. 93K-9. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of those sponsors.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[2] The Tor Project, Inc., "Tor: Anonymity Online," https://www.torproject.org/, 2012.

[3] "Silk Road (marketplace)," http://en.wikipedia.org/wiki/Silk_Road_(marketplace), 2012.

[4] Dennis Brown, "Resilient Botnet Command and Control with Tor," https://www.defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-18-Brown-TorCnC.pdf, 2010.

[5] "Dutch police infiltrate hidden child porn websites in the U.S." http://lincolntribune.com/?p=19100, 2011.

[6] "TorDir - the link list /and pm system/ of tor," http://dppmfxaacucguzpc.onion/index.php?p=cat&cid=2&sid=o40h5p0sllf4nlatcd0uo31377, 2012.

[7] L. Øverlier and P. Syverson, "Locating Hidden Servers," in *Proceedings of the IEEE Security and Privacy Symposium (S&P)*, May 2006.

[8] S. J. Murdoch, "Hot or Not: Revealing Hidden Services by Their Clock Skew," in *Proceedings of ACM CCS*, November 2006.

[9] R. Pries, W. Yu, X. Fu, and W. Zhao, "A New Replay Attack Against Anonymous Communication Networks," in *Proceedings of the IEEE International Conference on Communications (ICC)*, May 19-23 2008.

[10] Z. Ling, J. Luo, K. Wu, and X. Fu, "Protocol-level Hidden Server Discovery," http://www.cs.uml.edu/~xinwenfu/paper/HiddenServer.pdf, UMass Lowell, Tech. Rep., 2012.

[11] "Tor Directory Protocol, Version 3," https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt, 2012.

[12] "Privoxy," http://www.privoxy.org/, 2011.

[13] 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 *Proceedings of the 11th ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012.

[14] L. Zhang, J. Luo, M. Yang, and G. He, "Application-level attack against Tor's hidden service," in *Proceedings of the 6th International Conference on Pervasive Computing and Applications*, 2011.

[15] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On Flow Correlation Attacks and Countermeasures in Mix Networks," in *Proceedings of Workshop on Privacy Enhancing Technologies (PET)*, May 2004.

[16] M. Liberatore and B. N. Levine, "Inferring the Source of Encrypted HTTP Connections," in *Proceedings of ACM CCS*, October 2006.

[17] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob?" in *Proceedings of the 16th Annual USENIX Security Symposium (Security)*, August 2007.

[18] Z. Ling, J. Luo, Y. Zhang, M. Yang, X. Fu, and W. Yu, "A novel network delay based side channel attack: Modeling and defense," in *Proceedings of the 31th IEEE INFOCOM*, 2012.

[19] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I Still See You: Why Traffic Analysis Countermeasures Fail," in *Proceedings of the 6th IEEE Symposium on Security and Privacy (S&P)*, 2011.

[20] X. Wang, S. Chen, and S. Jajodia, "Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet," in *Proceedings of ACM CCS*, 2005.

[21] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "DSSS-Based Flow Marking Technique for Invisible Traceback," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (S&P)*, 2007 May.

[22] Z. Ling, X. Fu, W. Jia, W. Yu, and D. Xuan, "A novel packet size based covert channel attack against anonymizer," in *Proceedings of The 30th IEEE INFOCOM*, April 2011.

[23] S. J. Murdoch and G. Danezis, "Low-Cost Traffic Analysis of Tor," in *Proceedings of the IEEE Security and Privacy Symposium (S&P)*, May 2006.

[24] X. Fu, Y. Zhu, B. Graham, R. Bettati, and W. Zhao, "On Flow Marking Attacks in Wireless Anonymous Communication Networks," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, April 2005.

[25] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell counter based attack against tor," in *Proceedings of 16th ACM Conference on Computer and Communications Security (CCS)*, November 2009.

[26] A. Houmansadr and N. Borisov, "SWIRL: A Scalable Watermark to Detect Correlated Network Flows," in *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, 2011.