One Cell is Enough to Break Tor's Anonymity

Xinwen Fu and Zhen Ling
White Paper for Black Hat DC 2009

***Abstract*— Tor is a real-world, circuit-based low-latency anonymous communication network, supporting TCP applications over the Internet. In this paper, we present a new class of attacks, *protocol-level attacks*, against Tor. Different from existing attacks, these attacks can confirm anonymous communication relationships quickly and accurately by manipulating one single cell and pose a serious threat against Tor. In protocol-level attacks, a malicious entry onion router may duplicate, modify, insert, or delete cells of a TCP stream from a sender. The manipulated cells traverse middle onion routers and arrive at an exit onion router along a circuit. Because Tor uses the counter mode AES (AES-CTR) for encrypting cells, the manipulated cells disrupt the normal counter at exit onion routers and decryption at the exit onion router incurs cell recognition errors, which are unique to the investigated protocol-level attacks. If an accomplice of the attacker at the entry onion router also controls the exit onion router and recognizes such cell recognition errors, the communication relationship between the sender and receiver will be confirmed. Protocol-level attacks can also be used for launching the denial-of-service (DoS) attack to disrupt the operation of Tor. We have implemented these attacks on Tor and our experiments validate their feasibility and effectiveness. We also present guidelines for defending against such attacks.**

***Index Terms*— Protocol-level Attacks, Anonymity, Mix Networks, Tor**

I. Introduction

Concerns about privacy and security have received greater attention with the rapid growth and public acceptance of the Internet and the pervasive deployment of various wireless technologies. Anonymity has become a necessary and legitimate aim in many applications, including anonymous web browsing, location-based services (LBS), and E-voting. In these applications, encryption alone cannot maintain the anonymity required by participants [1], [2], [3].

Since Chaum pioneered the basic idea of anonymous communication systems, referred to as mixes [4], in 1981, researchers have developed various anonymity systems for different applications. Mix techniques can be used for either message-based (high-latency) or flow-based (low-latency) anonymity applications. Email is a typical message-based anonymity application, which has been thoroughly investigated [5], [6]. Research on flow-based anonymity applications has recently received a lot of attention in order to preserve anonymity in low-latency applications, including web browsing and peer-to-peer file sharing [7], [8], [9].

Tor [8] is a popular low-latency anonymous communication network, supporting TCP applications on the Internet. On October 18, 2008, there were 1164 active Tor onion routers operating around the world, which form an overlay-based mix network[1]. To communicate with an application server, a Tor client selects an *entry* onion router, a middle onion router and an *exit* onion router in the case of default path length of 3. A circuit is first built through this chain of three onion routers and the client negotiates a session key with each onion router. Then, application data is packed into cells, encrypted and decrypted in an onion-like fashion and transmitted through the circuit to the server [8].

Extensive research work has been carried out to investigate attacks which can degrade the anonymous communication over Tor. Most existing approaches are based on traffic analysis [3], [10], [11], [12], [13], [14], [15]. Specifically, to determine whether Alice is communicating with Bob through Tor, such attacks measure the similarity between the sender's outbound traffic and the receiver's inbound traffic in order to confirm their communication relationship. However, attacks based on traffic analysis may suffer a high rate of false positives due to various factors, such as Internet traffic dynamics and the need for a number of packets for the statistical analysis of traffic.

In this paper, we present a new class of attacks against Tor, namely *protocol-level* attacks, which do not rely on traffic analysis. In these attacks, the attacker needs to manipulate only one cell to confirm the communication relationship between the sender and receiver and poses a serious threat against Tor. In order to do so, the attacker may control multiple onion routers, similar to assumptions in existing attacks [3], [12]. A malicious entry onion router may duplicate, modify, insert, or delete cells of a TCP stream from a sender. The manipulated cells traverse middle onion routers and arrive at exit onion routers along circuits. Tor uses the counter mode of Advanced Encryption Standard (AES-CTR) for encryption and decryption of cells at onion routers. The manipulated cells will disrupt the normal counter at the middle and exit onion routers and the decryption at the exit onion router will incur cell recognition errors. Our investigation shows that such cell recognition errors are unique to protocol-level attacks. If an accomplice of the attacker at the entry onion router controls the exit onion router and detects such cell recognition errors, the communication relationship between the sender and receiver will be confirmed.

We have implemented these protocol-level attacks on Tor and our experiments validate the feasibility and effectiveness of these attacks. These attacks may also threaten the availability of the anonymity service by Tor since a malicious onion entry may release the circuits passing through it. We also provide guidelines for defending against these attacks. The attacks presented in this paper are one of the first to exploit the

Xinwen Fu is with Department of Computer Science, University of Massachusetts Lowell (Email: xinwenfu@cs.uml.edu). Zhen Ling is with School of Computer Science and Engineering, Southeast University, Liwenzheng Building (North) #241, Nanjing 210096, P.R. China (Email: zhen_ling@seu.edu.cn).

[1]In this paper, we use *Tor router* and *onion router* interchangeably. For brevity, *router* is used for the same purpose and the meaning should be clear from the context.

implementation of known anonymous communication systems such as Tor.

The remainder of this paper is organized as follows: We introduce the basic operation of Tor in Section II. We present the details of the protocol-level attacks, including the basic principle, algorithms, and discussion, in Section III. In Section IV, we show experimental results on Tor and validate our findings. We give guidelines for defending against these protocol-level attacks in Section V. We review related work in Section VI and conclude this paper in Section VII.

II. Basic Components and Operation of Tor

In this section, we first introduce the basic components of the Tor network. We then present its operation, including the circuit setup and its usage for transmitting TCP streams anonymously.

A. Components of the Tor Network

Tor is a popular overlay network for anonymous communication over the Internet. It is an open source project and provides anonymity service for TCP applications [16]. Figure 1 illustrates the basic components of Tor [17]. As shown in Figure 1, there are four basic components:

1) *Alice* (i.e. *Client*). The client runs a local software called *onion proxy* (*OP*) to anonymize the client data into Tor.
2) *Bob* (i.e. *Server*). It runs TCP applications such as a web service and anonymously communicates with *Alice* as the client over Tor.
3) *Onion routers (OR)*. Onion routers are special proxies that relay the application data between Alice and Bob. In Tor, Transport Layer Security (TLS) connections are used for the overlay link encryption between two onion routers. The application data is packed into equal-sized cells (512 bytes as shown in Figure 2) carried through TLS connections.
4) *Directory servers*. They hold onion router information such as public keys for onion routers. There are *directory authorities* and *directory caches*. Directory authorities hold authoritative information on onion routers and directory caches download directory information of onion routers from authorities. The client downloads the onion router directory from directory caches.

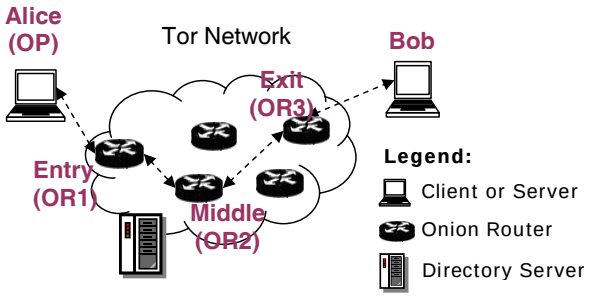

Fig. 1. Tor Network

Functions of onion proxy, onion router, and directory servers are integrated into the Tor released software package. A user can edit a configuration file and configure a server to have different combinations of those functions.

Figure 2 illustrates the cell format used by Tor. All cells have a three-bytes header, which is not encrypted in the onion-like fashion so that the intermediate Tor routers can see this header. The other 509 bytes are encrypted in the onion-like fashion. There are two types of cells: the *control* cell shown in Figure 2 (a) and *relay* cell shown in Figure 2 (b). The command field (*Command*) of a control cell can be: *CELL_PADDING*, used for keepalive and optionally usable for link padding, although not used currently; *CELL_CREATE* or *CELL_CREATED*, used for setting up a new circuit; and *CELL_DESTROY*, used for releasing a circuit. The command field (*Command*) of a relay cell is *CELL_RELAY*. Notice that relay cells are used to carry TCP stream data from Alice to Bob. The relay cell has an additional header, namely the relay header. There are numerous types of relay commands (*Relay Command*), including *RELAY_COMMAND_BEGIN*, *RELAY_COMMAND_DATA*, *RELAY_COMMAND_END*, *RELAY_COMMAND_SENDME*, *RELAY_COMMAND_EXTEND*, *RELAY_COMMAND_DROP*, and *RELAY_COMMAND_RESOLVE*[2]. We will explain these commands further in later sections of the paper when we discuss the Tor operations from protocol-level attacks perspective.

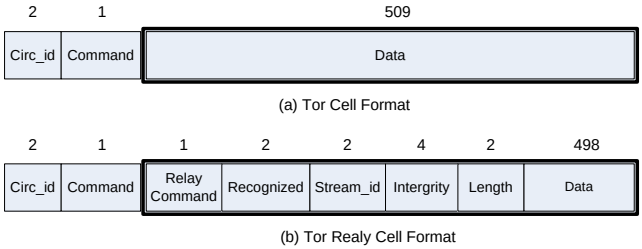

Fig. 2. Cell Format By Tor [8]

B. Selecting a Path and Creating a Circuit

In order to anonymously communicate with applications, i.e., browsing a web server, a client uses a way of source routing and chooses a series of onion routers from the locally cached directory, downloaded from the directory caches [18]. We denote the series of onion routers as the *path* through Tor [19]. The number of onion routers is referred to as the *path length*. We use the default path length of 3 as an example in Figure 1 to illustrate how the path is chosen. The client first chooses an appropriate exit onion router $OR3$, which should have an exit policy supporting the relay of the TCP stream from the sender. Then, the client chooses an appropriate entry onion router $OR1$ (referred to as *entry guard* and used to prevent certain profiling attacks [20]) and a middle onion router $OR2$.

Once the path is chosen, the client initiates the procedure of creating a circuit over the path incrementally, one hop at a time. Figure 3 illustrates the procedure of creating a circuit when the path has a default length of 3. Tor uses TLS/SSLv3 for link authentication and encryption. In Figure

[2]All these can be found in *or.h* in released source code package by Tor.

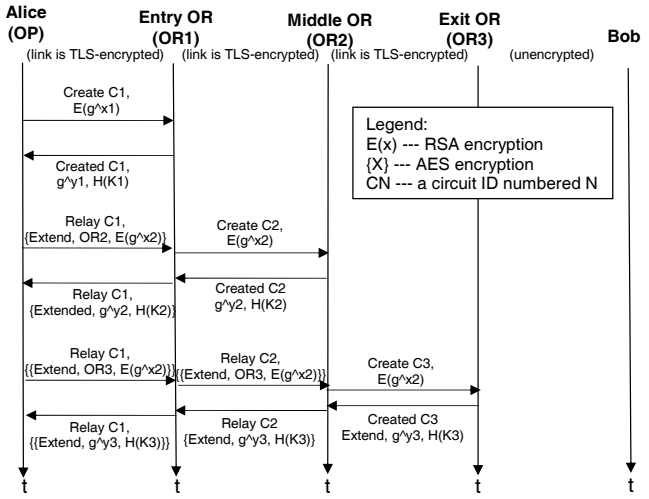

Fig. 3. Tor Circuit Creation [8]

3, *OP* first sets up a TLS connection with *OR1* using the TLS protocol. Then, tunneling through this connection, OP sends a *CELL_CREATE* cell and uses the *Diffie-Hellman* (DH) handshake protocol to negotiate a base key $K_1 = g^{xy}$ with *OR1*, which responds with a *CELL_CREATED* cell. From this base key material, a forward symmetric key kf_1 and a backward symmetric key kb_1 are produced [17]. In this way, a one-hop circuit *C1* is created.

To extend the circuit one hop further, the *OP* sends to *OR1* a *RELAY_COMMAND_EXTEND* cell, specifying the address of the next onion router, i.e., *OR2* in Figure 3. This *RELAY_COMMAND_EXTEND* cell is encrypted by AES in the counter mode (AES-CTR) with kf_1. Once *OR1* receives this cell, it decrypts the cell and negotiates secret keys with *OR2* using the DH handshake protocol. Therefore, a second segment *C2* of the 2-hop circuit is created. *OR1* sends *OP* a *RELAY_COMMAND_EXTENDED* cell, which holds information for *OP* generating the shared secret keys: forward key kf_2 and backward key kb_2, with *OR2*. This *RELAY_COMMAND_EXTENDED* cell is encrypted by AES-CTR with key kb_1. *OP* will decrypt the *RELAY_COMMAND_EXTENDED* cell and use the information to create the corresponding keys. Encryption of later cells by these secret keys uses AES-CTR as well.

Consequently, to extend the circuit to a 3-hop circuit, *OP* sends *OR2* a *RELAY_COMMAND_EXTEND* cell, specifying the address of the third onion router, e.g., the *OR3* shown in Figure 3, through the 2-hop circuit. As we can see, the cell is encrypted in an onion-like fashion [17]. The payload is first encrypted by kf_2 and then by kf_1. The encrypted cell, like an *onion*, becomes thinner when it traverses an onion router, which removes one layer of onion skin by decrypting the encrypted cell. Therefore, when *OR2* decrypts the cell, it finds that the cell is meant to create another segment of the circuit to *OR3*. *OR2* negotiates with *OR3* and sends a *RELAY_COMMAND_EXTENDED* cell back to *OP*. This cell is first encrypted by kb_2 at *OR2* and then by kb_1 at *OR1*. *OP* decrypts the encrypted backward onion-like cell and derives the shared secret keys with *OR3*, including the forward key kf_3 and backward key kb_3.

In summary, *OP* negotiates secret keys with the three onion routers one by one and consequently creates a circuit along the path[3]. With the exception that the connection from the exit onion router to the server is not link encrypted, other connections along the path are all protected by TLS within Tor. That is, cells encrypted in the onion-like fashion are protected by link encryption. In the description above, we simply use a circuit of path length 3 as an example and a circuit of path length greater than 3 can be set up in a similar manner.

C. Transmitting TCP Streams

Without loss of generality, we will use a short TCP stream, transferring 5 bytes of data "Hello" from Alice (OP) to Bob, as an example to illustrate how a TCP stream is tunneled through the circuit that has already been created by the procedures described in Section II.B. Figure 4 illustrates this simple example. Recall that at this stage, a client's *OP* has established secret keys with other onion routers and can encrypt the application payload.

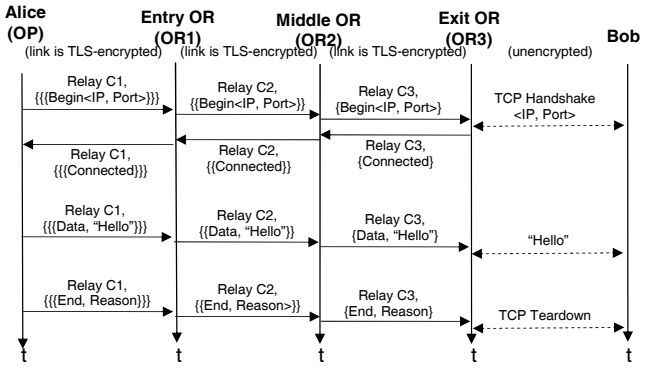

Fig. 4. TCP Connection Creation and Data Transmission on Tor

To transmit data to Bob, Alice's application (such as web browser) first contacts the *OP*, which is implemented as a SOCKS proxy locally. The *OP* learns the destination IP address and port. *OP* sends a *RELAY_COMMAND_BEGIN* cell to the exit onion router *OR3*, and the cell is encrypted as $\{\{\{Begin < IP, Port >\}_{kf_3}\}_{kf_2}\}_{kf_1}$, where the subscript refers to the key used for encryption of one onion skin. The three layers of onion skin are removed one by one each time the cell traverses an onion router through the circuit as we described in Section II.B. When *OR3* 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*, which belongs to Bob. Therefore, *OR3* acts as a proxy, sets up a TCP connection with Bob, and sends a *RELAY_COMMAND_CONNECTED* cell back to Alice's *OP*. The *OP* then accepts data from Alice's application, packs it into relay cells with the *Relay Command* of *RELAY_COMMAND_DATA* and transmits it to Bob through the circuit. The whole process is transparent to Alice, who only needs to configure the application to use the *OP*. When Alice's application finishes the data transmission,

[3]Each onion router checks the flag, "Recognized" field within the relay cell shown in Figure 2 (b) to determine whether the cell reaches its end. In this way, the encrypted cell has a fixed size and its length does not swell as in the public key encryption case [4].

the connection from Alice's application to the *OP* will be released. As shown in Figure 4, after 5 bytes of data "Hello" in a *RELAY_COMMAND_DATA* cell is transmitted, Alice's application releases the connection to *OP*. *OP* then sends a *RELAY_COMMAND_END* cell to *OR3* and *OR3* finally releases the connection to Bob. In this way, the circuit of path over Tor will be released completely.

III. Protocol-Level Attacks

In this section, we first introduce the basic principle of these protocol-level attacks. We then present the detailed algorithms followed by discussion.

A. Basic Principle

Recall that the purpose of these attacks is to confirm that Alice is communicating with Bob over Tor. We assume that an attacker can control the entry and exit onion routers (also called the malicious onion routers) used by a given circuit for a TCP stream and launch protocol-level attacks by manipulating the cells associated with the given circuit. The malicious entry onion router logs the information, including the source IP address and port used for a given circuit, the circuit ID, and the time of the cell being manipulated. The attacker may launch protocol-level attacks in the following ways: (i) *duplicating* a target cell along the given circuit and then sending the duplicated cell at an appropriate time; (ii) *modifying* some bits of 509-bytes data of a target cell and forwarding such a modified cell to the next hop along the circuit over Tor; (iii) *inserting* an artificial cell into the victim circuit at an appropriate time; and (iv) *deleting* a target cell without forwarding it to the next hop. The duplicated cell, modified cell, artificially inserted cell, or the cell after the deleted cell traverses the circuit and arrives at the exit onion router. The attacker at the malicious exit onion router can detect cell recognition errors raised by those manipulated cells. The attacker records the time of the cell recognition error, the destination IP address and port associated with the circuit, and the corresponding circuit ID. In this way, the attackers can confirm that the target cell enters Tor through the malicious entry onion router and the target cell exits Tor from the malicious exit onion router. Since the entry onion router knows the source IP address of the TCP stream and the exit onion router knows the destination IP address of the TCP stream, the communication relationship between the sender and receiver will be confirmed. In the following, we will explain the detailed algorithms of these protocol-level attacks.

B. Algorithms of Protocol-Level Attacks

We studied and implemented these four protocol-level attacks based on the Tor release version of 0.2.0.28[4]. To validate such attacks, we need to modify the source code of the malicious entry onion router and exit onion router. From the description in Section III.A, we know that for a successful protocol-level attack, there are two important issues. One is how to choose the time to launch the attack and how to select the cell to manipulate at the entry onion router. The other is how to recognize the error at the exit onion router.

At an entry onion router, the attacker needs to carefully choose the time to launch the attack and identify the cell to be manipulated. For example, if a cell is selected during the circuit setup process, the duplicated cell traversing through the victim circuit will cause numerous protocol errors and immediately cause the circuit to fail upon its creation. Therefore, the protocol-level attacks need to manipulate cells carrying TCP stream data instead of cells carrying control commands for circuit setup. Although cells are encrypted, the attacker at the entry onion router can determine the relay cells based on the relay command in the cell header. We now present the detailed steps of launching these protocol-level attacks.

Step 1: The attacker at entry onion routers receives many requests from an *OP* or other onion routers. The attacker needs to verify whether these requests originate from an *OP*, not from other onion routers that use the malicious entry onion router as a middle onion router or an exit onion router.

The rule of the verification is that, if the source IP address of the request is not in the list of directory servers, this request is from an *OP*. From the procedure of creating a circuit shown in Figure 3, we know that the attacker can determine the moment when the circuit is created. In terms of a circuit with default path length of 3, the circuit is created if one *CELL_CREATE* and two *CELL_RELAY* cells are transmitted on the forward path, and one *CELL_CREATED* cell and two *CELL_RELAY* cells on the backward path. Therefore, at a malicious entry onion router, after one *CELL_CREATE* and two *CELL_RELAY* cells are transmitted on the forward path, the attacker knows that this circuit is completely created.

Step 2: Now the attacker needs to determine the time to launch the attack and select appropriate target cells.

After the circuit is created, according to the procedure of transmitting a TCP stream shown in Figure 4, *OP* will send a *relay* cell with the relay command *RELAY_COMMAND_BEGIN* in the relay header of the cell. This specific cell is used to request the exit onion router to setup a TCP connection to the server. After receiving the cell, the exit onion router creates a TCP connection to the server directly. Then the next *relay* cell sent by an *OP* shall contain TCP stream data and relay command of this cell is *CELL_RELAY_DATA*. After an *OP* successfully sends all data to the server, it will receive and forward the final *relay* cell with relay command *CELL_RELAY_END*. When the exit router receives this cell, it releases the TCP connection to the server.

Therefore, according to the procedures of creating a circuit and transmitting TCP streams over the circuit, the attacker at the entry onion router can determine the *CELL_RELAY_BEGIN* cell and the first *CELL_RELAY_DATA* cell. To summarize, after the attacker at the entry onion router records one *CELL_CREATE* cell and three *CELL_RELAY* cells on the forward path with the same circuit ID, the attacker decides that the third *CELL_RELAY* cell on the forward path will be a *CELL_RELAY_BEGIN* cell. Then the relay cell after that will be *CELL_RELAY_DATA* cell, i.e., the first cell with TCP stream data from an *OP*.

Step 3: Since the cells from an *OP* are identified in the second step, the attacker can now launch the protocol-level

[4]Newer release versions of Tor have not changed the algorithms investigated in this paper.

attacks in the following different ways:

1. Replay A Cell: Figure 5 illustrates the basic principle of this attack. At an entry onion router, the attacker identifies the first *CELL_RELAY_DATA* cell on a victim circuit and duplicates it. Then, the duplicated cell will traverse the circuit and arrive at the exit onion router. The attacker at the malicious exit onion router will detect a cell recognition error caused by this duplicated cell.

We now go through cases and explain details that cause the cell recognition error. When a data cell is duplicated at *OR1*, the decryption at *OR2* and *OR3* will fail. The reason is that the cell's onion layers are encrypted using AES in the counter mode and the counter is disturbed by the duplicated cell. Specifically, in the counter mode, encryption and decryption operations need to keep a synchronized value, *a counter*. The encryption of a cell at an *OP* increases the AES counter by one. The three routers along the path increase the counter for each cell they receive and decrypt the original cell successfully. When *OR1* duplicates a cell, the duplicate cell causes *OR2* and *OR3* to increase the counter and this makes the decryption of this cell on *OR2* and *OR3* unsynchronized and incurs a decryption error. In the current Tor implementation, default actions to this error are: *OR3* releases the circuit and an *OP* creates another circuit for continuous communication. Notice that although the decryption at *OR2* is wrong, it does not raise any action on the circuit. This is because the cell is onion-like encrypted, the two fields, *Recognized* and *Integrity* (in Figure 2 (b)) used for integrity checking can only be recognized after all layers of encryption are removed, and *OR2* cannot recognize the decryption error associated with the duplicated cell. *OR3* can use the fields of "Recognized" or "Integrity" of the relay header in Figure 2 (b) to recognize the error since all the onion layers should have been removed at *OR3*.

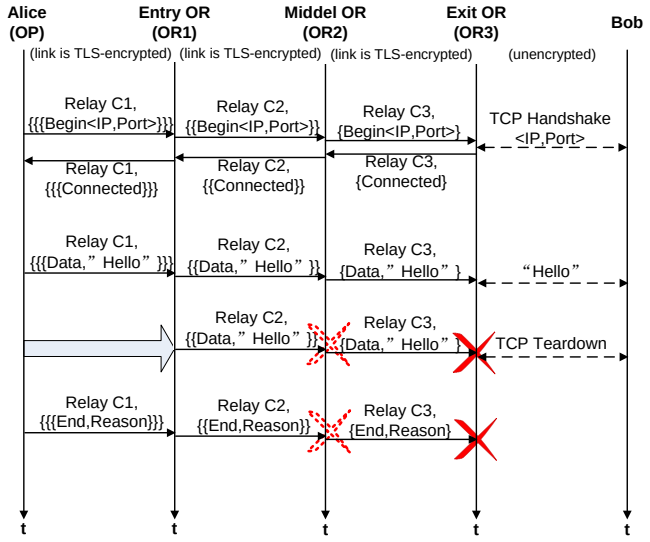

Fig. 5. Replay a Cell on Tor

2. Modify A Cell: Figure 6 illustrates the basic principle of this attack. At an entry router, the attacker captures the first *CELL_RELAY_DATA* cell on a circuit and modifies certain data in the encrypted payload. For example, the attacker can set the first byte of the encrypted payload to zero. When this modified cell passes through the circuit and arrives at the exit onion router, the attacker at the malicious exit onion router will also detect the cell recognition error caused by this modified cell since the modified cell destroys the integrity of the cell and the exit onion router will not be able to decrypt it correctly.

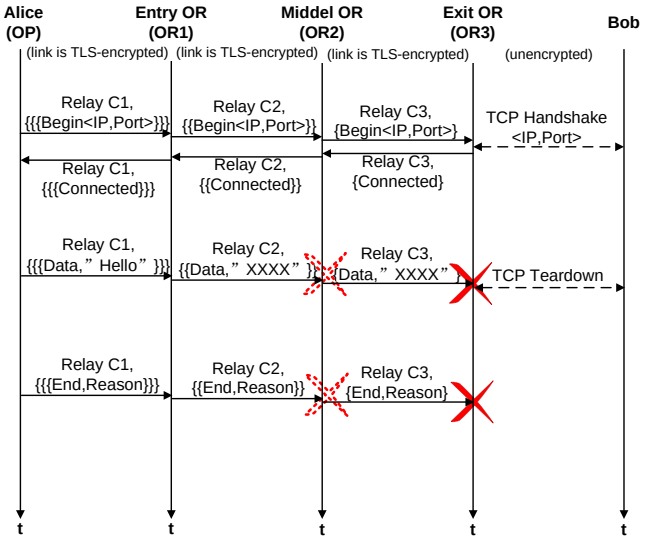

Fig. 6. Modify a Cell on Tor

The attack of modifying a cell shares some similarity with "tagging" attack described in [8]. The work in [8] claimed that Tor can prevent tagging attacks by applying integrity checks. However, the attacks we investigated in this paper utilize the error information created by the integrity check at malicious routers. The attack of modifying a cell can still confirm the communication relationship and pose a serious threat against Tor.

3. Insert a Faked Cell: Figure 7 illustrates the basic principle of this attack. When the attacker relays the first *CELL_RELAY_DATA* cell on a circuit, the attacker at an entry onion router inserts a new faked relay cell constructed by himself on the forward path. The circuit ID of the faked cell will be the same as other cells on the target circuit. However, the payload of this faked cell will be randomly generated. Then the faked cell will traverse the circuit and arrive at the exit onion router. The attacker at the malicious exit onion router will detect a cell recognition error caused by this faked cell. The principle of the cell recognition error caused at the exit onion router is similar to the one which replays a cell on the circuit. When *OR1* inserts a new faked cell, the inserted cell causes *OR2* and *OR3* to increase the counter. This will make the encryption and decryption of the faked cell at *OR2* and *OR3* unsynchronized.

4. Delete A Cell: Figure 8 illustrates the basic principle of this attack. An attacker at the entry onion router identifies the first *CELL_RELAY_DATA* cell on a circuit and deletes it. Then, the attacker relays the second relay cell, as usual. The second relay cell will traverse the circuit and arrive at the exit onion router. The attacker at the malicious exit onion router will detect a cell recognition error caused by the deleted cell. The principle of the recognition error caused at the exit onion router is also similar to replaying a cell on the circuit.

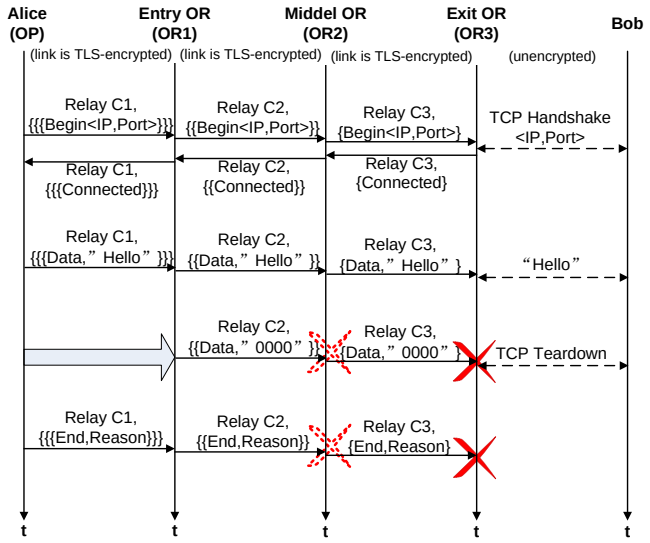

Fig. 7. Insert a Faked Cell on Tor

When *OR1* deletes a cell, the deleted cell causes *OR2* and *OR3* to not be able to increase the counter. This makes the encryption and decryption of succeeding cells at *OR2* and *OR3* unsynchronized.

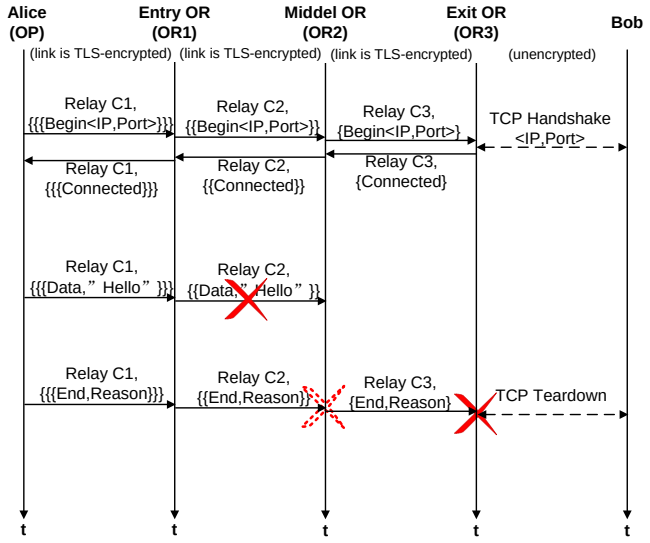

Fig. 8. Delete a Cell on Tor

Step 4: At this step, the attackers will confirm the communication relationship between Alice and Bob.

Recall that when cells of a given circuit are manipulated at the malicious entry onion router, cell recognition errors will show up at the exit onion router if the TCP stream is transmitted through that circuit. The exit onion router records the circuit ID, the destination IP address, the port number, and timestamp. The entry router records the timestamp of manipulation, the circuit ID, and the source IP address. We use Network Time Protocol (NTP) to synchronize the malicious entry and onion routers. By correlating the time of sending a manipulated cell with the time of detecting a cell recognition error, we can confirm that the recognition error is actually caused by the manipulated cell. To the best of our knowledge, based on extensive experiments on Tor over months, these cell recognition errors are unique to protocol-level attacks and the probability of other facts causing such errors is very low. Once there is a cell manipulation at the entry onion router and a cell recognition error shows up at the exit onion router simultaneously, the attackers know that the circuit segment IDs recorded at the entry and exit routers belong to the same circuit, which carries the target TCP stream data. Since the entry onion router knows the source IP address of the TCP stream and the exit onion router knows the destination IP address of the TCP stream, the attackers can link the communication relationship between Alice and Bob. In Section IV, we will use the time correlation as a measure to demonstrate the correlation between the cell manipulation and recognition error.

We can see that these protocol-level attacks are a very powerful threat against Tor, since the attackers only need to manipulate one cell and detect recognition errors. Therefore, these attacks are simple, fast, and accurate. All these make these attacks quite different from other existing attacks based on traffic analysis, which require lengthy parameter tuning for the trade-off between the false positive rate and detection rate [21], [11], [3], [12], [14], [15]. Additionally, these protocol-level attacks are robust to the network size, traffic dynamics, and other anti-traffic analysis strategies, including batching, reordering, and dummy traffic schemes [2], [22].

C. Discussion

1) Making Attacks Stealthy: In order to make the attack stealthy, the attacker can choose an appropriate time to manipulate cells. Notice that once there is a cell recognition error, the corresponding circuit will be released by default because the AES counter is disturbed along the circuit. If the attacker manipulates the cells when a TCP connection is still running, the circuit will be released and other circuits will have to be created to relay the rest of the TCP stream data from Alice to Bob. This may raise Alice and Bob's attention. Therefore, the attacker shall replay the cells at the moment when the circuit is not occupied with the stream data from Alice and before the circuit is released by Alice. Such an attack will not degrade the TCP performance and can be stealthy.

The attacker may even use a loop-control way to detect the status of the TCP stream data and send the duplicated cell in a proper time. One possible way is that the attacker at the exit onion router with the full information of the target TCP stream notifies the attacker at the entry onion router. The attacker at the entry onion router identities the first *CELL_RELAY_DATA* cell on a circuit and holds the duplicated cell until he or she receives the indication from the attacker at the exit onion router. When a *CELL_RELAY_END* cell is received at the exit onion router, the attacker at the exit onion router will notify the attacker at the entry onion router to send the duplicated cell. After the duplicated cell arrives at the exit onion router, the attacker at the exit onion router will detect an error caused by the cell duplication. In this case, the TCP connection will be disconnected by the *OP* as usual, and such an attack will not be as detectable detected by Alice and Bob.

Figure 9 illustrates one example of this type of stealthy attack. In a stealthy attack of replaying a cell, the attacker can duplicate a cell, hold it, and replay the cell when the current TCP session from OP is complete.

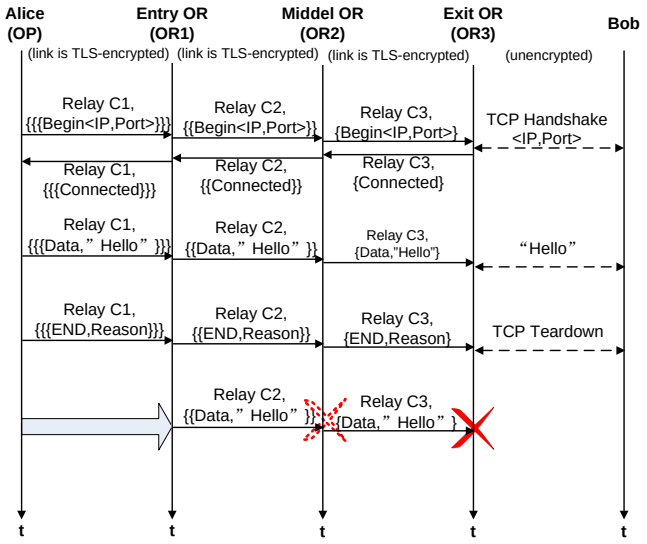

Fig. 9. Duplicate and Hold a Cell on Tor

2) Controlling Onion Routers: In the discussion of these protocol-level attacks, we assume that the attacker controls some entry and exit onion routers. This is a reasonable assumption due to the design principle of Tor, i.e., volunteer-based operation [8]. Anyone can set up entry onion routers and exit onion routers and join Tor. As long as a router has an exit policy enabling access to external services, this onion router becomes an exit onion router. To become an entry onion router, a Tor router must meet some criteria. If an onion router has a mean time between failure (MTBF) not less than the median for active onion routers or at least 10 days, it becomes a *stable* onion router. A stable onion router can be promoted to an entry onion router if its bandwidth is either at least the median among known active onion routers or at least 250KB/s [18]. This set of criteria are not difficult to meet by attackers in real-world practice. Experiments in Section IV-B confirm this claim.

These protocol-level attacks can be more flexible. The requirement of a malicious exit onion router is not necessary in these protocol-level attacks if an attacker can monitor outbound streams from an exit onion router. This kind of traffic monitoring capability has been widely used by other existing attacks [21], [11], [3], [12], [14], [15]. To this end, using network traffic monitoring tools, the attacker can record the destination IP address and port number of outbound TCP streams from an exit onion router. When the manipulated cell arrives at the exit onion router and the monitored TCP stream from this exit onion router aborts abruptly, this gives a positive sign that the TCP stream from the sender travels along that exit onion router, addressed to the corresponding receiver.

3) Reducing Noise: We now discuss the noise reduction related to these protocol-level attacks. The false positive of these attacks comes from unexpected cell recognition errors caused by attacks. Based on our month-long experiments on exit onion routers in Tor, we have not recorded such unexpected errors. This confirms that the false positive rate of protocol-level attacks against Tor is very low.

In order to further decrease the false positive rate, the attacker may process multiple buffered cells from a single TCP stream simultaneously. For each processed cell, we assume that the detection rate and false positive rate of the protocol-level attacks is p_d and p_f, respectively. We now derive the detection rate P_D and false positive rate P_F for processing n cells. When n cell recognition errors are detected at the exit onion router, the probability that all errors are not caused by the cell manipulation becomes $(1-p_d)^n$ and the detection rate becomes $P_D = 1-(1-p_d)^n$. The corresponding false positive rate is $P_F = p_f^n$. Therefore, by choosing an appropriate n, the attacker can achieve a high detection rate and a small false positive rate.

4) Launching DoS Attack: These protocol-level attacks can also be used to launch other attacks, including DoS attack. In order to launch DoS attack, the attacker only needs to control entry onion routers. If the malicious entry onion router manipulates cells, it will cause corresponding exit onion routers to disconnect the circuit and release the TCP connection. This will slow the operation of Tor network if the attacker control multiple malicious entry onion routers. In addition, Tor's directory authorities monitor the activities of onion routers and may blacklist those innocent exit onion routers which unexpectedly drop circuits and TCP connections. Although those malicious entry routers are the root-cause for this, those innocent exit onion routers become scapegoats. Due to the anonymity naturally maintained by Tor, it will be non-trivial to trace back to those malicious entry onion routers.

IV. Evaluation

We have implemented the four protocol-level attacks with schemes presented in Section III on Tor [23]. In this section, we use real-world experiments to demonstrate the effectiveness and feasibility of these protocol-level attacks on Tor. All the experiments were conducted in a controlled manner and we experimented on TCP flows generated by ourselves in order to avoid legal issues.

A. Experiment Setup

Figure 10 shows the experiment setup. We use two malicious onion routers as the Tor entry onion router and exit onion router. The entry onion router, client (Alice) and server (Bob) are located in an office on campus. The exit onion router is located in an off-campus location. Computers on campus and off-campus are on different public IP segments connecting to different Internet service providers (ISPs).

To minimize the side effects of the protocol-level attacks on Tor's normal operation, we conduct experiments in a partially controlled environment. We modify the Tor client code for attack verification purposes. The Tor client would only build circuits through the designated malicious exit onion router and entry onion router in Figure 10. The middle onion router is selected using the default routing selection algorithm released by Tor. Recall that the goal of the protocol-level attacks is to confirm whether the client communicates with the

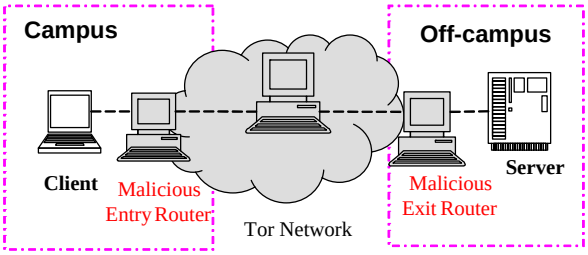

Fig. 10. Experiment Setup

server. For verification purposes, we created a simple client/server application which transmits data through TCP. The server in our experiments binds to port 41, receives packets, and outputs relevant connection information to the server's screen for debugging and measurement purpose. The Tor client utilizes *tsocks* [24] to automatically transport its outbound TCP stream through the *OP* using SOCKS. By using the Tor configuration file and manipulatable parameters, such as *EntryNodes*, *ExitNodes*, *StrictEntryNodes*, and *StrictExitNodes* [19], we setup the client to select the malicious onion routers along the circuit. The exit onion router uses the default exit policy from Tor and the entry onion router's exit policy only allows it to be used as either an entry or middle router.

B. Experimental Results of Protocol-Level Attacks

The publicly available bandwidth information of onion routers from the Tor directory servers confirms that the set of criteria for becoming an entry onion router is not difficult to meet. According to the bandwidth information collected from the directory server on October 18, 2008, there were 1164 active onion routers on Tor, including 239 pure entry onion routers, 411 pure exit onion routers, and 117 EE routers. Figure 11 shows the bandwidth distribution of onion routers on Tor, based on the directory information collected on August 18, 2008. The mean value of the bandwidth is only around 57KB/s. After running for only about 5 days, our onion router with a bandwidth of 200KB/s was promoted to be an entry guard.

To validate the accuracy of these protocol-level attacks, in our experiments we let the client send a message packed in one cell to the server approximately every 10 seconds. The revised code at the entry onion router records the time of manipulating cells. The revised code at the exit onion router records the time of recognition errors and carries out the correlation test to confirm the communication relationship between the sender and receiver. We use the *correlation coefficient* r to measure the strength of correlation between the time of manipulating cells and the time of detecting the cell recognition errors. Correlation coefficient is defined as

$$r = \frac{\sum_{x,y}(x-\bar{x})(y-\bar{y})}{\sqrt{\sum_{x}(x-\bar{x})^2}\sqrt{\sum_{y}(y-\bar{y})^2}}, \quad (1)$$

where x is the time of manipulating cells at the entry onion router, y is the time of cell recognition errors incurring at the exit onion router, and $\bar{x}$ and $\bar{y}$ are the mean values of x and y, respectively.

Figures 12, 13, 14, 15 and 16 show the relationship between the time of duplicating, modifying, inserting, and deleting cells and the time of incurring cell recognition errors. Notice that the Figure 13 shows the enhanced strategy to replay cells in a stealthy manner as we discussed in Section III-C.1. As we can see from these figures, there is a perfect linear correlation in all the cases, since the actual correlation coefficient between them is *one*. This strongly confirms that these protocol-level attacks can accurately confirm the communication relationship if the sender and receiver use Tor to anonymize their communication. In addition to the high accuracy, these protocol-level attacks can be very quick and efficient, since the attacker only needs to manipulate one cell and recognize the error caused by the manipulated cell. Notice that the time correlation is not necessary for these protocol-level attacks against Tor. The perfect time correlation just validates the accuracy of these attacks.

V. Guideline of Countermeasures

We have demonstrated the threat of several protocol-level attacks against Tor. We now discuss possible countermeasures to these attacks.

1) Minimizing Number of Compromised Entry Routers: Recall that the protocol-level attacks require an attacker to fully control at least one entry router. To achieve this, the attacker may advertise the false bandwidth resource and promote compromised servers to be entry routers of Tor. There are two possible ways to minimize the chance that compromised servers become entry routers. First, the path selection algorithm may be evolved and select only fully trusted and dedicated ones through strict authentication and authorization processes. Second, countermeasures may be developed to detect false bandwidth advertisements from a compromised router that intends to become Tor entry router via the attack similar to Sybil attack [25]. For example, the path selection protocols used by Tor can be augmented to allow onion routers to proactively monitor each other and validate other onion routers' bandwidth [26]. A reputation-oriented defensive scheme can be develop to further facilitate the countermeasure to the attacks. In this way, the attacker will have less chance to control the entry onion router and the effectiveness of these protocol-level attacks will be reduced. However, this approach cannot completely defend against these protocol-level attacks, since the attackers may still contribute servers with high bandwidth if enough bandwidth resources are available.

2) Monitoring Manipulated Cells: Recall that these protocol-level attacks need to send the manipulated cells. If manipulated cells can be detected and dropped at a middle router before they reach to the exit onion router, the effectiveness of such attacks will be significantly reduced. To this end, one naive way is to allow the middle onion router along the circuit to detect manipulated cells by buffering historical cells. However, this will raise more overhead to onion routers, since Tor requires using a pair of memory buffers for reading and writing data from each TCP stream [27].

Another way to detect these protocol-level attacks is to have the Tor's clients and exit routers monitor the connections

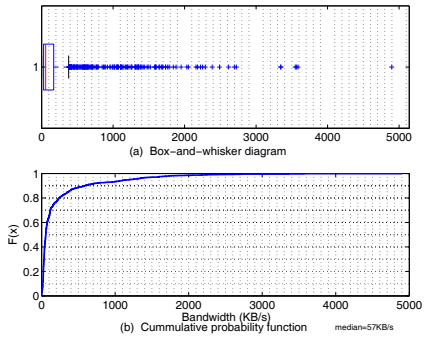

Fig. 11. Onion Routers' Bandwidth Distribution on Tor: Bandwidth Median=57KB/s; (a) Box and Whisker Plot of Bandwidth; (b) Cumulative Distribution Function of Bandwidth

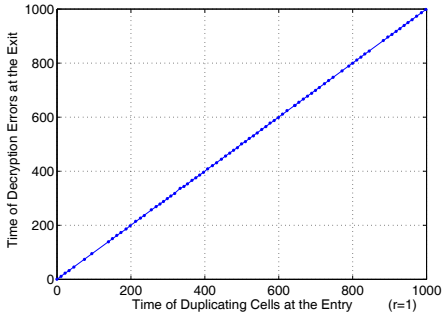

Fig. 12. Correlation Between Time of Duplicated Cells and Time of Cell Recognition Errors

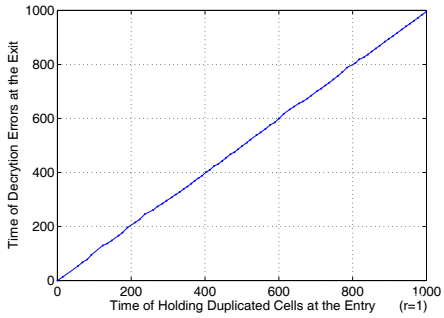

Fig. 13. Correlation Between Time of Holding Duplicated Cells and Time of Cell Recognition Errors

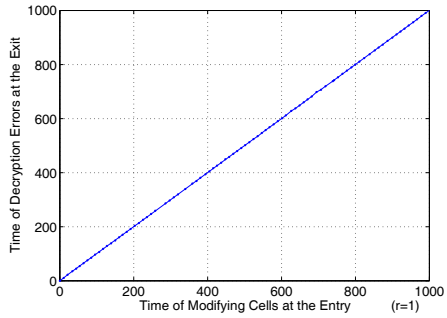

Fig. 14. Correlation Between Time of Modified Cells and Time of Cell Recognition Errors

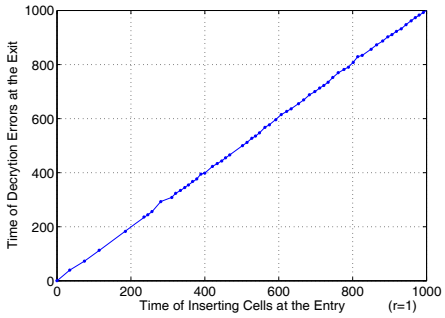

Fig. 15. Correlation Between Time of Inserted Cells and Time of Cell Recognition Errors

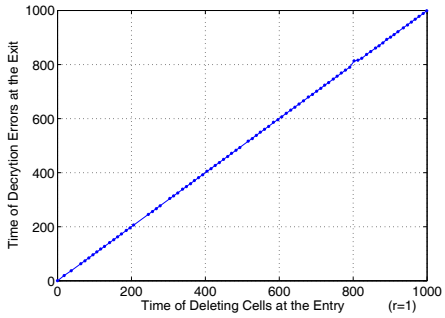

Fig. 16. Correlation Between Time of Deleted Cells and Time of Cell Recognition Errors

which have anomaly behavior. Since these attacks will cause connections to be released and force the client switch to a new circuit, a frequent connection release and circuit switch may indicate the possibility of these protocol-level attacks. The client cannot solely rely on the reported reason codes for circuit release for detection purpose since the malicious exit router may manipulate the reason code on purpose. When a protocol-level attack is launched to confirm the communication relationship which does not exist, exit routers other than the malicious ones will receive manipulated cells and detect decryption errors. Such decryption errors may indicate a high possibility of such attacks.

VI. Related Work

Chaum pioneered the idea of anonymous communication systems in [4]. A good review of various mix systems can be found in [8], [5]. There has been much research on how to degrade anonymous communication through mix networks. To determine whether Alice is communicating with Bob, through a mix network, similarity between Alice's outbound traffic and Bob's inbound traffic may be measured. For example, Zhu *et al.* [11] proposed the scheme of using mutual information for the similarity measurement. Levine *et al.* [10] investigated a cross correlation technique. Murdoch *et al.* [12] also investigated the timing-based attacks on Tor by using some compromised Tor routers. Fu *et al.* [2] studied a flow marking scheme to actively embed a specific pattern in the target flow and confirm the communication relationship between the sender and receiver. Overlier *et al.* [3] studied a scheme using one compromised mix node to identify the "hidden server" anonymized by Tor. Yu *et al.* [15] proposed an invisible traceback approach based on the direct sequence spread spectrum (DSSS) technique. This approach could be used by attackers to secretly trace the communication relationship via the anonymous communication networks.

Interval-based watermarks are proposed to trace attackers through the stepping stones. For example, Wang *et al.* [28] proposed a scheme that injected nondisplayable content into packets. Wang *et al.* [29] proposed an active watermarking scheme that was robust to random timing perturbation. They analyzed the tradeoffs between the true positive rate, the maximum timing perturbation added by attackers, and the number of packets needed to successfully decode the watermark. Wang *et al.* [30] also investigated the feasibility of a timing-based watermarking scheme in identifying the encrypted peer-to-peer VoIP calls. By slightly changing the timing of packets, their approach can correlate encrypted network connections. Nevertheless, these timing-based schemes are not effective at tracing communication through a mix network with batching strategies that manipulate inter-packet delivery timing, as indicated in [15]. Peng *et al.* [31] analyzed the secrecy of timing-based watermarking traceback proposed in [29], based on the distribution of traffic timing. Kiyavash *et al.* [32] proposed a multi-flow approach detecting the interval-based watermarks [33], [34] and DSSS-based watermarks [15]. This multi-flow based approach intends to average the rate

of multiple synchronized watermarked flows and expects to observe a unusual long silence period without packets or a unusual long period of low-rate traffic.

There is little research that has been conducted on the attacks based on non-traffic analysis. To the best of our knowledge, Murdoch *et al.* [35] investigated an attack to reveal hidden servers of Tor by exploiting the fact that the clock deviations of a target server should be consistent with the server's load. Differently, the protocol-level attacks studied in this paper exploit the fundamental protocol design in Tor. Our investigated attacks are simple, accurate, quick, and easy to deploy.

VII. Conclusion

In this paper, we introduced a new class of protocol-level attacks on Tor, which allows the attacker to quickly and accurately confirm the anonymous communication over Tor. In these attacks, the attacker at the malicious entry onion router manipulates cells from the sender's outbound TCP stream. The manipulated cell will be carried along a circuit of Tor and causes the cell recognition errors at the exit onion router. Since such cell recognition errors are unique to these new attacks, the attacker can confirm the communication relationship between the sender and receiver accurately and quickly. Via extensive theoretical analysis and real-world experiments, the effectiveness and feasibility of the protocol-level attacks are validated. Our other preliminary data showed that these protocol-level attacks may drastically degrade the anonymity service that Tor provides, if the attacker is able to control a small number of Tor routers. Such attacks may also be used to threaten the availability of the anonymity service by Tor.

Due to Tor's fundamental design, defending against these protocol-level attacks remains a challenging task that we will investigate in our future research.

Acknowledgment

We acknowledge the fruitful discussion of these protocol-level attacks with Tor developers and researchers including Roger Dingledine and Nick Mathewson.

References

[1] Q. X. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, and L. L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, May 2002.

[2] 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.

[3] L. Overlier and P. Syverson, "Locating hidden servers," in *Proceedings of the IEEE Security and Privacy Symposium (S&P)*, May 2006.

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

[5] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: design of a type iii anonymous remailer protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy (S&P)*, May 2003.

[6] C. Gülcü and G. Tsudik, "Mixing email with babel," in *Proceedings of the Network and Distributed Security Symposium (NDSS)*, February 1996.

[7] M. Reiter and A. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, 1998.

[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[9] Anonymizer, Inc., http://www.anonymizer.com/, 2008.

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

[11] 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.

[12] 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.

[13] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against anonymous systems," in *Proceedings of ACM Workshop on Privacy in the Electronic Society (WPES)*, October 2007.

[14] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proceedings of the IEEE Symposium on Security & Privacy (S&P)*, May 2008.

[15] 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.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: anonymity online," http://tor.eff.org/index.html.en, 2008.

[17] R. Dingledine and N. Mathewson, "Tor protocol specification," http://tor.eff.org/svn/trunk/doc/spec/tor-spec.txt, 2008.

[18] N. Mathewson, "Tor directory protocol, version 3," http://tor.eff.org/svn/trunk/doc/spec/dir-spec.txt, 2008.

[19] R. Dingledine and N. Mathewson, "Tor path specification," http://tor.eff.org/svn/trunk/doc/spec/path-spec.txt, 2008.

[20] G. Danezis and R. Clayton, "Route fingerprinting in anonymous communications," in *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, September 2006.

[21] M. Wright, M. Adler, B. N. Levine, and C. Shields, "Defending anonymous communication against passive logging attacks," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[22] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood: active attacks on several mix types," in *Proceedings of Information Hiding Workshop (IH)*, February 2002.

[23] "Tor: anonymity online," http://tor.eff.org/, 2008.

[24] "A transparent socks proxying library," http://tsocks.sourceforge.net, 2008.

[25] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against anonymous systems," University of Colorado at Boulder, Tech. Rep., August 2007.

[26] K. Harfoush, A. Bestavros, and J. W. Byers, "Measuring bottleneck bandwidth of targeted path segments," in *Proceedings of the IEEE INFOCOM*, April 2003.

[27] "Theonionrouter/torfaq," http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ, 2008.

[28] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, "Sleepy watermark tracing: An active network-based intrusion response framework," in *Proceedings of 16th International Conference on Information Security (IFIP/Sec)*, June 2001.

[29] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of inter-packet delays," in *Proceedings of the 2003 ACM Conference on Computer and Communications Security (CCS)*, November 2003.

[30] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer voip calls on the internet," in *Proceedings of the 12th ACM Conference on Computer Communications Security (CCS)*, November 2005.

[31] P. Peng, P. Ning, and D. S. Reeves, "On the secrecy of timing-based active watermarking trace-back techniques," in *Proceedings of the IEEE Security and Privacy Symposium (S&P)*, May 2006.

[32] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *Proceedings of USENIX Security*, 2008.

[33] Y. J. Pyun, Y. H. Park, X. Wang, D. S. Reeves, and P. Ning, "Tracing traffic through intermediate hosts that repacketize flows," in *Proceedings of IEEE INFOCOM*, May 2007.

[34] S. C. X. Wang and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (S&P)*, May 2007.

[35] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, November 2006.