Computer Networks 57 (2013) 869–886

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Protocol-level attacks against Tor

Zhen Ling [a,1], Junzhou Luo [a,2], Wei Yu [b,*,3], Xinwen Fu [c,4], Weijia Jia [d,5], Wei Zhao [e,6]

[a] School of Computer Science and Engineering, Southeast University, Nanjing 211189, PR China
[b] Department of Computer and Information Sciences, Towson University, Towson, MD 21252, United States
[c] Department of Computer Science, University of Massachusetts Lowell, Lowell, MA 01854, United States
[d] Department of Computer Science, City University of Hong Kong, Hong Kong Special Administrative Region, PR China
[e] Rector of University of Macau, Macau SAR, China

ARTICLE INFO

Article history:
Received 31 January 2012
Received in revised form 20 July 2012
Accepted 6 November 2012
Available online 16 November 2012

Keywords:
Protocol-level attacks
Anonymity
Mix networks
Tor

ABSTRACT

Tor is a real-world, circuit-based low-latency anonymous communication network, supporting TCP applications over the Internet. In this paper, we present an extensive study of *protocol-level attacks* against Tor. Different from existing attacks, the attacks investigated in this paper can confirm anonymous communication relationships quickly and accurately by manipulating one single cell and pose a serious threat against Tor. In these attacks, a malicious entry onion router may duplicate, modify, insert, or delete cells of a TCP stream from a sender, which can cause cell recognition errors at the exit onion router. 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. These attacks can also be used for launching the denial-of-service (DoS) attack to disrupt the operation of Tor. We systematically analyze the impact of these attacks and our data indicate that these attacks may drastically degrade the anonymity service that Tor provides, if the attacker is able to control a small number of Tor routers. We have implemented these attacks on Tor and our experiments validate their feasibility and effectiveness. We also present guidelines for defending against protocol-level attacks.

1. Introduction

1.1. Motivation

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 (LBSs), and E-voting. In these applications, encryption alone cannot maintain the anonymity required by participants [1–3].

Since Chaum pioneered in 1981 the basic idea of anonymous communication systems, referred to as mixes [4], 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 escalated in response to the need to preserve anonymity in low-latency applications, including web browsing and peer-to-peer file sharing [7–9].

* Corresponding author. Tel.: +1 214 208 5951.
E-mail addresses: zhenling@seu.edu.cn (Z. Ling), jluo@seu.edu.cn (J. Luo), wyu@towson.edu (W. Yu), xinwenfu@cs.uml.edu (X. Fu), wei.jia@cityu.edu.hk (W. Jia), WeiZhao@umac.mo (W. Zhao).
[1] Tel.: +1 250 4725837; fax: +1 250 8132361.
[2] Tel.: +025 52091010; fax: +025 52091010.
[3] Tel.: +410 704 5528; fax: +410 704 3868.
[4] Tel.: +978 934 3623; fax: +978 934 3551.
[5] Tel.: +852 3442 9701; fax: +852 3442 0147.
[6] Tel.: +853 8397 4301; fax: +853 2883 1694.

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.11.005

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. In this paper, we use *Tor router*, *onion router* and *router* interchangeably. To communicate with an application server, a Tor client downloads all the router information from dedicated directory servers, and selects three routers as an *entry* onion router, a middle onion router and an *exit* onion router in the case of default path length of 3. A circuit (a special tunnel) 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. Notice that cells are transmission units of Tor, encrypted and decrypted in an onion-like fashion and transmitted through the circuit to the server [8]. Please refer to the basic components and operation of Tor in Section 2.

Extensive research work has been conducted to investigate attacks degrading the anonymous communication over Tor. Most existing approaches are based on traffic analysis [3,10–15]. 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 (e.g., Internet traffic dynamics) and also the need for a number of packets for the statistical analysis of traffic.

1.2. Our contribution

In this paper, we present an extensive study of *protocol-level attacks* against the live Tor system for the first time. In these attacks, an attacker needs to manipulate only *one* cell (the transmission unit of Tor) 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 cell traverses the middle onion routers and arrives at the exit onion router along circuits. Tor uses the counter mode of Advanced Encryption Standard (AES-CTR) for encryption and decryption of cells at onion routers. The manipulated cell 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 these 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 disrupt the circuits passing through it anonymously. We also provide guidelines for defending against these attacks. The attacks presented in this paper are one of the first to exploit the implementation of known anonymous communication systems such as Tor in practice.

Two salient features distinguish the protocol-level attacks investigated in this paper from other existing attacks. First, the protocol-level attacks are highly effective and require only one cell to be successful. No existing attack can achieve this effect. Second, the protocol-level attacks do not rely on the analysis of traffic timing, which is often hard to control and predict on the Internet, but is critical to the success of timing-based attacks. For example, Bauer et al. in [13] presented an attack utilizing circuit setup timing. When a client uses a malicious entry, the attacker at the entry node knows which middle node the client tries to use. When the client tries to connect to the exit node via this middle node, if the selected exit node is another malicious node, the attackers may determine that the client builds a circuit along this particular sequence of entry, middle and exit nodes based on timing correlation. However, there may be other clients using the same middle and exit Tor routers. Therefore, timing is critical for the success of this attack. Differently, traffic dynamics will not affect the success of a protocol-level attack since we exploit the implementation of Tor protocol, rather than traffic timing. Although Figs. 12–16 in Section 4 illustrate the correlation of the time of decryption errors at an exit with the time of cell manipulation at an entry, their purpose is to demonstrate that decryption errors indeed occur after the manipulation of cells. Hence, even if Tor would deploy the mechanisms of traffic padding [10,16,17] to defeat many traffic timing-based attacks, it cannot defeat the protocol-level attacks.

1.3. Related work

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 and Danezis [12] investigated the timing-based attacks on Tor by using 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 and Syverson [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.

The authors in [18,8] briefly discussed the possibility of tagging attacks, which share some similarity with the cell modification attack, *one* of the five attacks extensively investigated in this paper. The goal of such a so-called

tagging attack is to change the cell content at an entry router and recognize the changed cell leaving the Tor network by matching the changed content. The authors claimed that integrity checking used by Tor can prevent this type of malleability attack because it is hard for an attacker to guess the SHA1 MAC (Message Authentication Code) and tag the message. However, protocol-level attacks in this paper exploit circuit decryption errors, other than content "tags". To the best our knowledge, there is no discussion of attacks utilizing decryption errors in any related work, including [18,8]. We are the first exploiting this Tor protocol defect.

Interval-based watermarks are proposed to trace attackers through the stepping stones and anonymous communication networks. For example, Wang et al. [19] proposed a scheme that injected nondisplayable content into packets. Wang and Reeves [20] proposed an active watermarking scheme that was robust to random timing perturbation. They analyzed the tradeoff 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. [21] 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. [22] analyzed the secrecy of timing-based watermarking traceback proposed in [20], based on the distribution of traffic timing. Kiyavash et al. [23] proposed a multi-flow approach detecting the interval-based watermarks [24,25] 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 unusually long silence period without packets or a unusually long period of low-rate traffic.

There is little research conducted on the attacks based on non-traffic analysis. To the best of our knowledge, Murdoch [26] 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.

1.4. Paper organization

The remainder of this paper is organized as follows: We introduce the basic operation of Tor in Section 2. We present the details of the protocol-level attacks, including the basic principle and algorithms, in Section 3. We also discuss issues such as making attacks stealthy and controlling onion routers in this section. In Section 4, we show experimental results on Tor and validate our findings. We give guidelines such as using bridge relays for defending against protocol-level attacks in Section 5. We conclude the paper in Section 6.

2. 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 anonymously transmitting TCP streams.

2.1. 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 [27]. Fig. 1 illustrates the basic components of Tor [28]. As shown in Fig. 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* over the Tor network.
(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 Fig. 2) carried through TLS connections.
(4) *Directory servers*. Directory servers hold onion router information such as router public keys. 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.

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

Fig. 2 illustrates the cell format used by Tor. All cells have a three-byte 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 Fig. 2a and *relay* cell shown in Fig. 2b. The command field (*Command*) of a control cell

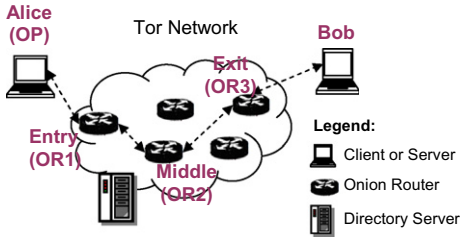


Fig. 1. Tor network.

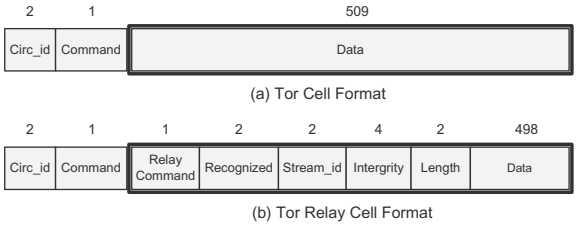


Fig. 2. Cell format by Tor [8].

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*.[1] The command field (*Recognized*) is used to identify whether the cell is correctly recognized by the client or exit router. Because multiple streams are multiplexed into a single circuit, the command field (*Stream_id*) is used to identify the specific stream for the corresponding applications at the client or exit router. The command field (*integrity*) is used to verify the integrity of the data. Because the data can be padded into equal size, the command field (*length*) is used to indicate the size of real data packed into a cell. We will explain these commands further in later sections when we discuss the Tor operations from the perspective of protocol-level attacks.

2.2. 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 [29]. We denote the series of onion routers as the *path* through Tor [30]. The number of onion routers is referred to as the *path length*. We use the default path length of 3 as an example in Fig. 1 to illustrate how the path is selected. The client first chooses an appropriate exit onion router *OR*3, 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 *OR*1 (referred to as *entry guard* used to prevent certain profiling attacks [31]) and a middle onion router *OR*2.

Once the path is chosen, the client initiates the procedure of creating a circuit over the path incrementally, one hop at a time. Fig. 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 Fig. 3, *OP* first sets up a TLS connection with *OR*1 using

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

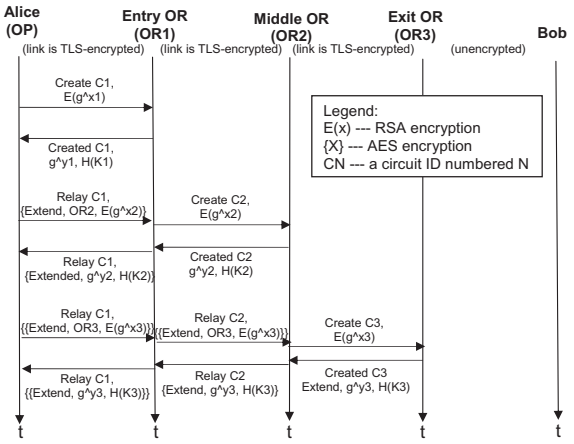


Fig. 3. Tor Circuit creation [8].

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 *OR*1 and derive the hash value of $H(K_1)$, which corresponds to a *CELL_CREATED* cell. From this base key material, a forward symmetric key kf_1 and a backward symmetric key kb_1 are generated [28]. In this way, a one-hop circuit *C1* is created.

To extend the circuit one hop further, the *OP* sends to *OR*1 a *RELAY_COMMAND_EXTEND* cell, specifying the address of the next onion router, i.e., *OR*2 in Fig. 3. Notice that *RELAY_COMMAND_EXTEND* is simplified as *Extended* in this figure because of the limited space. This *RELAY_COMMAND_EXTEND* cell is encrypted by AES in the counter mode (AES-CTR) with kf_1. Once *OR*1 receives this cell, it decrypts the cell and negotiates secret keys with *OR*2 using the DH handshake protocol. Therefore, a second segment *C2* of the 2-hop circuit is created. *OR*1 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 *OR*2. 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 *OR*2 a *RELAY_COMMAND_EXTEND* cell, specifying the address of the third onion router, e.g., the *OR*3 shown in Fig. 3, through the 2-hop circuit. As we can see, the cell is encrypted in an onion-like fashion [28]. 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 *OR*2 decrypts the cell, it finds that the cell tends to create another segment of the circuit to *OR*3. *OR*2 negotiates with *OR*3 and sends a *RELAY_COMMAND_EXTENDED* cell back to *OP*. This cell is first encrypted by kb_2 at *OR*2 and then by kb_1 at *OR*1. *OP* decrypts the encrypted backward onion-like cell and derives the shared secret keys with *OR*3, including the forward key kf_3 and backward key kb_3.

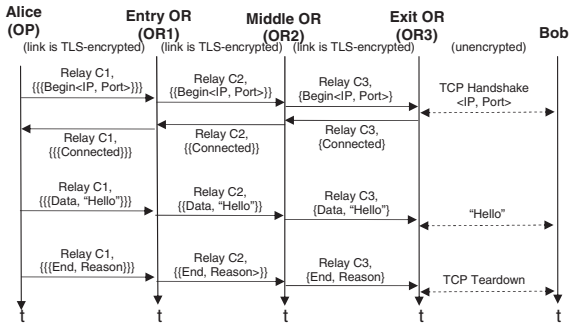


Fig. 4. TCP connection creation and data transmission on Tor.

In summary, *OP* negotiates secret keys with the three onion routers one by one and consequently creates a circuit along the path.[2] 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.

2.3. 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 2.2. Fig. 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.

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 *OR*3, 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 2.2. When *OR*3 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, *OR*3 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, the connection from Alice's application to the *OP* will be released. As shown in Fig. 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 *OR*3 and *OR*3 finally releases the connection to Bob. In this way, the circuit of path over Tor will be released completely.

[2] Each onion router checks the flag, "Recognized" field within the relay cell shown in Fig. 2b 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].

3. 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.

3.1. 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 via the malicious entry onion router and the target cell exits Tor via 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.

3.2. Algorithms of protocol-level attacks

We studied and implemented the aforementioned four protocol-level attacks based on the Tor release version of 0.2.0.28.[3] To validate those attacks, we need to modify the

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

source code of the malicious entry onion router and exit onion router. From the description in Section 3.1, 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. The formal algorithm can be found in Appendix A.

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 Fig. 3, we know that the attacker can determine the time 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 Fig. 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 attacks in the following ways:

1. *Replay A Cell*: Fig. 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 *OR*1, the decryption at *OR*2 and *OR*3 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 *OR*1 duplicates a cell, the duplicate cell causes *OR*2 and *OR*3 to increase the counter and this makes the decryption of this cell on *OR*2 and *OR*3 unsynchronized and incurs a decryption error. In the current Tor implementation, default actions to this error are: *OR*3 releases the circuit and an *OP* creates another circuit for continuous communication. Notice that although the decryption at *OR*2 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*

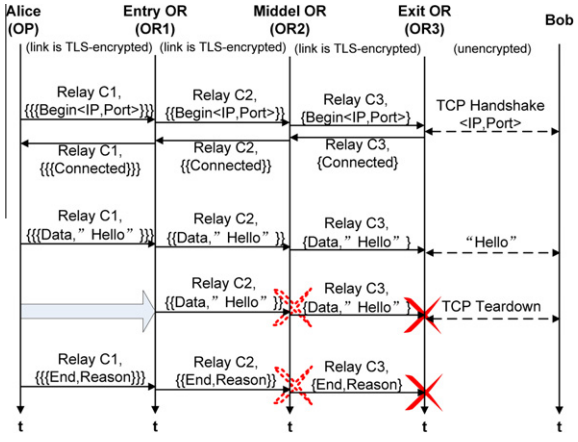


Fig. 5. Replay a cell on Tor.

(in Fig. 2b) used for integrity checking can only be recognized after all layers of encryption are removed, and *OR*2 cannot recognize the decryption error associated with the duplicated cell. *OR*3 can use the fields of "Recognized" or "Integrity" of the relay header in Fig. 2b to recognize the error since all the onion layers should have been removed at *OR*3.

2. *Modify A Cell*: Fig. 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 be unable to decrypt it correctly. The attack of modifying a cell shares some similarity with the "tagging" attack described in [18,8]. The work in [18,8] 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*: Fig. 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 *OR*1 inserts a new faked cell, the inserted cell causes *OR*2 and *OR*3 to increase the counter. This will make the encryption and decryption of the faked cell at *OR*2 and *OR*3 unsynchronized.

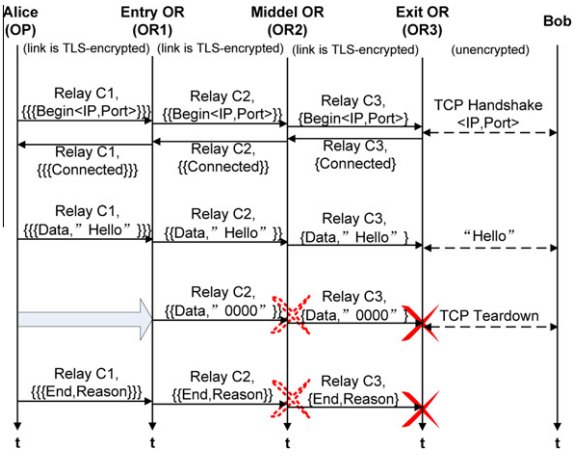


Fig. 7. Insert a faked cell on Tor.

4. *Delete A Cell:* Fig. 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. The attacker then 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. When *OR*1 deletes a cell, the deleted cell causes *OR*2 and *OR*3 fail to increase the counter. This makes the encryption and decryption of succeeding cells at *OR*2 and *OR*3 unsynchronized.

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 appear at the exit onion router if the TCP stream is

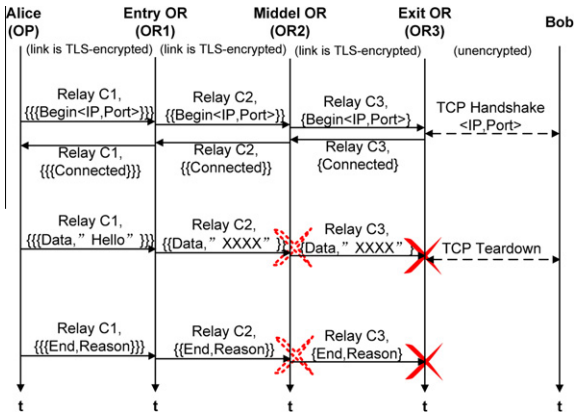


Fig. 6. Modify a cell on Tor.

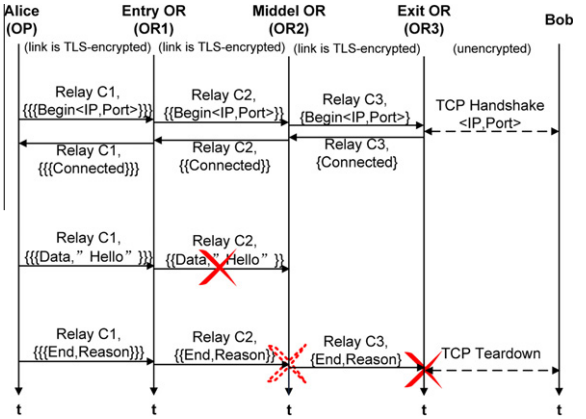


Fig. 8. Delete a cell on Tor.

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 appears at the exit onion router, 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 4, 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, making 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 [32,11,3,12,14,15]. In addition, 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 [2,33].

3.3. Discussion

3.3.1. Making attacks stealthy

In order to make the attack stealthy, the attacker can choose an appropriate time to manipulate cells. Note 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. In this way, the attack will not degrade the TCP performance and can be stealthy.

The attacker may even use a loop-control method 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 identifies the first *CELL_RELAY_DATA* cell on a circuit and holds the duplicated cell until the indication from the attacker at the exit onion router is received. In particular, 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 the attack will not be detectable by Alice and Bob.

Fig. 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.

3.3.2. 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 volunteer-based operation principle of Tor [8]. Anyone, including governments conducting censorship over Tor, 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 250 KB/s [29]. This set of criteria is not difficult to meet by attackers in real-world practice. Experiments in Section 4.4 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 [32,11,3,12,14,15]. 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

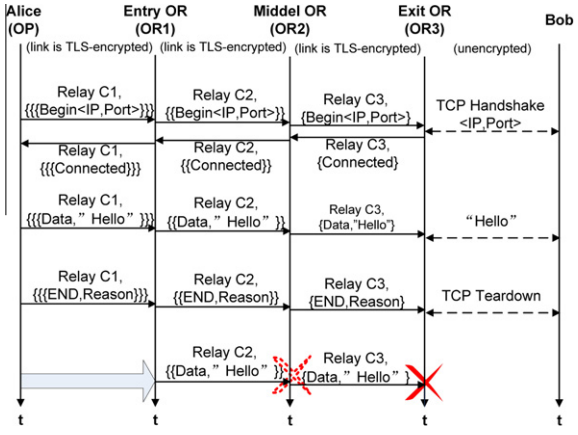


Fig. 9. Duplicate and hold a cell on Tor.

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.

3.3.3. 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 can be 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.

3.3.4. Launching protocol-level attacks in parallel

In an extreme case, many independent attackers may try to launch the protocol-level attacks, sometimes simultaneously. False-positives would rise accordingly although a single protocol-level attack does not introduce the false positive. To address this problem, the attackers can use attack timing information to stand them out. Note that a single session of protocol-level attack does not require timing information. Synchronized adversaries can conduct protocol-level attacks in a time division multiplexing fashion to achieve a low false positive rate.

3.3.5. Launching DoS attack

These protocol-level attacks can also be used to launch other attacks, including DoS attack. In order to do so, 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 controls 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, the innocent exit onion routers become scapegoats. Due to the anonymity naturally maintained by Tor, it will be non-trivial to identify those malicious entry onion routers.

4. Evaluation

We have implemented the five protocol-level attacks illustrated in Figs. 5–9 in Section 3 on Tor [34]. Note: the attack in Fig. 9 is the enhanced strategy to replay cells in a stealthy manner as we discussed in Section 3.3.1. In this section, we use real-world experiments to demonstrate the effectiveness and feasibility of these attacks on Tor. All experiments were conducted in a controlled manner and we experimented on TCP flows generated by ourselves in order to avoid legal issues.

4.1. Experiment setup

Fig. 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 Fig. 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 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* [35] 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* [30], 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.

4.2. Experimental results of protocol-level attacks

The publicly available onion router bandwidth information from the Tor directory servers confirms that becoming an entry onion router is not difficult. According to the bandwidth information collected from the directory server

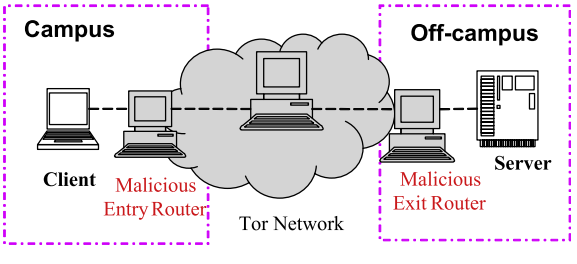


Fig. 10. Experiment setup.

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. Fig. 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 57 KB/s. After running for only about 5 days, our onion router with a bandwidth of 200 KB/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 s. 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.

Figs. 12–16 show the relationship between the time of duplicating, modifying, inserting, and deleting cells and the time of incurring cell recognition errors. Note that Fig. 13 shows the enhanced strategy to replay cells in a stealthy manner as we discussed in Section 3.3.1. 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 high

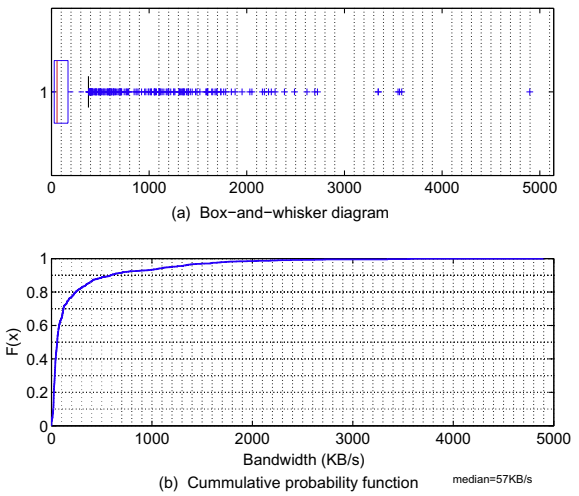


Fig. 11. Onion routers' bandwidth distribution on Tor: bandwidth median = 57 KB/s; (a) box and whisker plot of bandwidth; (b) cumulative distribution function of bandwidth.

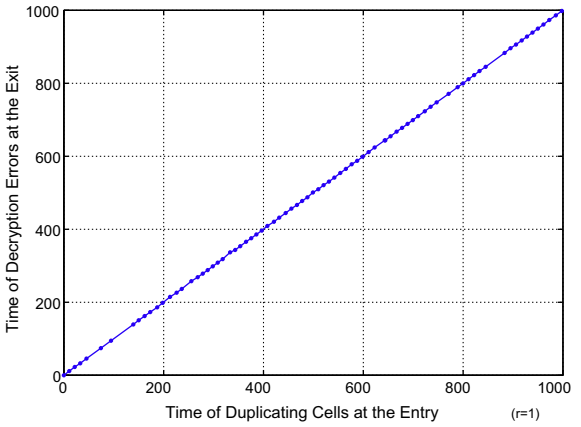


Fig. 12. Correlation between time of duplicated cells and time of cell recognition errors.

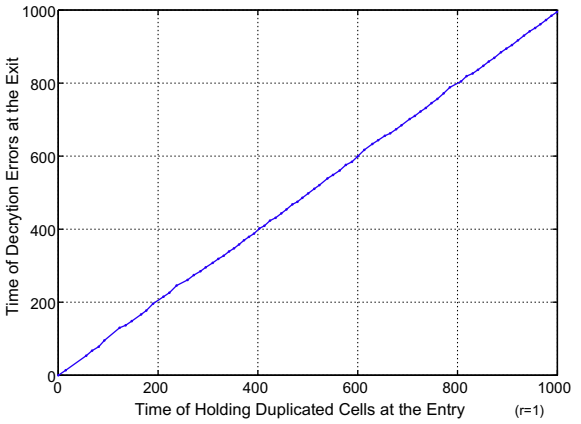


Fig. 13. Correlation between time of holding duplicated cells and time of cell recognition errors.

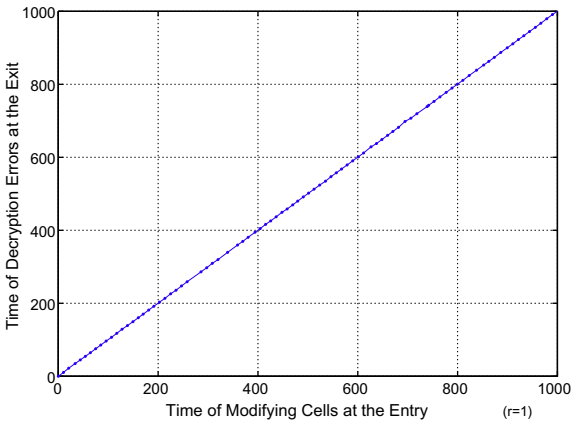


Fig. 14. Correlation between time of modified cells and time of cell recognition errors.

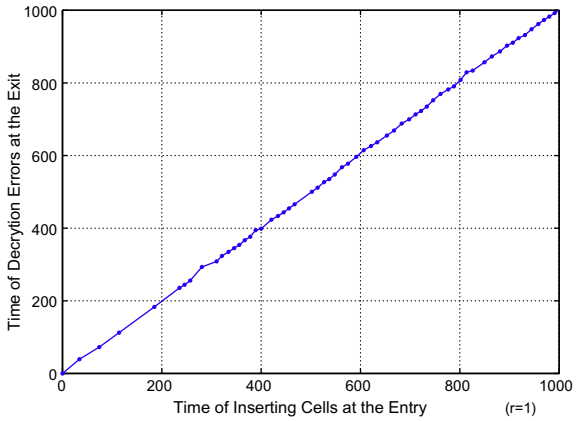


Fig. 15. Correlation between time of inserted cells and time of cell recognition errors.

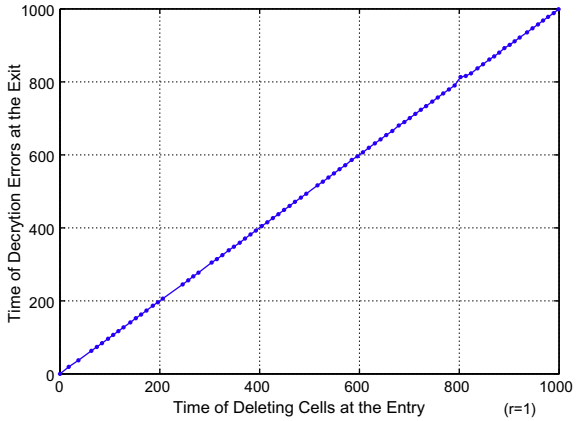


Fig. 16. Correlation between time of deleted cells and time of cell recognition errors.

accuracy, these protocol-level attacks are very efficient, since the attacker only needs to manipulate one cell and recognize the error caused by the manipulated cell. Note 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.

4.3. Analysis of the impact of protocol-level attacks

We investigate the impact of these protocol-level attacks on Tor. We can see from the attacks described in Section 3, if a TCP stream traverses a pair of the malicious entry and exit onion routers, the attacker can confirm the communication relationship quickly and accurately by launching a protocol-level attack. In order to fully understand the impact of such attacks on Tor, we need to evaluate the probability that a TCP stream traverses both the malicious entry onion router and exit onion router, given that a number of routers in Tor are malicious and controlled by attacker. Combined with experimental evaluation in Section 4.4, our analysis shows that the attacks investigated in this paper can drastically degrade the anonymity service that Tor provides by confirming the communication relationships of a large number of flows from senders to receivers within Tor, even if the attacker can only control a relatively small number of onion routers. Our analysis is general and can be applied to other anonymous communication systems. In the following, we will first present two schemes that the attacker may use to increase the attack impact on Tor and we then compare these two schemes with a brute-force scheme.

4.3.1. Scheme 1: Injecting malicious onion routers

From the attacks described in Section 4.4, the attackers need to control a number of onion routers and the communication relationship between the sender and receiver who transmit their data anonymously via a pair of malicious entry and exit onion routers can be linked. In order to do so, we first consider the scheme in which the attacker intends to inject malicious onion routers into Tor. We assume that the existing onion routers are secure and honest. Although Tor recently amended its routing algorithm to require that no two routers on a circuit may be from the same class B address space, this type of attack can still be readily deployed by a government that wants cyber censorship and possesses ample resources such as IP addresses across different regions. In order to increase the probability P that a circuit chooses malicious onion routers as entry and exit routers, the attacker shall increase the probability that such malicious routers are used for either an entry or exit router. In order to do so, the attacker can choose the malicious routers that can have a long uptime and high bandwidth.[4] This will increase the probability that malicious onion routers are selected for either an entry or exit router [28].

Assume that the attacker injects k malicious routers and the bandwidths of all onion routers comprise a set $\{B_1, \ldots, B_k, B_{k+1}, \ldots, B_{k+N}\}$, where $B_1 \geqslant \ldots \geqslant B_{k+N}$, that is, the malicious onion routers $\{B_1, \ldots, B_k\}$ have the maximum bandwidth within the set. Then the onion router with bandwidth B_i will be chosen with a probability $a_i = B_i / \sum_{i=1}^{k+N} B_i$, based on weighted bandwidth routing algorithm used by Tor [28]. The probability P that a circuit chooses the malicious routers as entry and exit routers becomes,

$$P(k) = \sum_{i=1}^{k} \left(a_i \sum_{j=1, j \neq i}^{k} \frac{a_j}{1 - a_i} \right). \tag{2}$$

From (2), we have two observations. First, $P(k)$ is an increasing function of a_i that measures the bandwidth that malicious routers are able to contribute. That is, the higher bandwidth that malicious routers have, the higher $P(k)$ becomes. Second, $P(k)$ is an increasing function of the number of malicious routers. That is, the larger number of malicious routers that attackers can control, the higher $P(k)$ becomes. Appendix B presents the proof of $P(k)$'s property.

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

Algorithm 1. Selection of malicious onion routers

Require:
(a) p ($\in[p_1, p_2]$), the ratio of malicious onion routers in Tor,
(b) N, the total number of onion routers in Tor,
(c) $P(R_{1,i}, R_{2,j})$, the probability that a circuit chooses the malicious exit onion router $R_{1,i}$ and entry onion router $R_{2,j}$,
(d) $\mathcal{P}[1 * (k-1)]$, an array storing the calculated probability that a circuit chooses the malicious entry and exit routers,
(e) M, an array storing the maximum probability in array P.
Ensure: the result in array M is maximum value in array P.
for $p = p_1$ to p_2 **do**
  $k = round(p * N)$
  for $g = 1$ to $(k-1)$ **do**
    Select g best onion routers from $R1$ as exit onion routers
    if the onion routers selected from $R1$ are EE routers **then**
      Remove the EE routers from $R2$
      Calculate the probability $a_{2,j} = B_{2,j} / \sum B_{2,j}$
    end if
    Select $k-g$ best routers from $R2$ as entry onion routers
    if the onion routers selected from $R2$ are EE routers **then**
      Remove the EE routers from $R1$
      Calculate the probability $a_{1,i} = B_{1,i} / \sum B_{1,i}$
    end if
    $\mathcal{P}[g] = \sum_{i=1}^{g} \sum_{j=1}^{k-g} P(R_{1,i} R_{2,j})$
    $= \sum_{i=1}^{g} \sum_{j=1}^{k-g} (a_{1,i} a_{2,j})$
  end for
  $M \Leftarrow \max(\mathcal{P})$
end for

4.3.2. Scheme 2: Compromising existing Tor routers

We now study how the attacker may choose some of the exit onion routers and compromise them[5] in order to maximize the probability P. Notice that there are four types of routers in Tor, i.e., pure entry onion routers, pure exit onion routers, onion routers that can be either an entry or exit router (denoted as *EE* router) as well as routers that can only be used as middle onion routers.

We assume that the set of pure exit onion routers and EE routers is $R1 = \{R_{1,1}, R_{1,2}, \ldots, R_{1,n}\}$, and the set of pure entry routers and EE routers is set $R2 = \{R_{2,1}, R_{2,2}, \ldots, R_{2,m}\}$, respectively. The bandwidth of onion routers in the set $R1$ is a set $B1 = \{B_{1,1}, B_{1,2}, \ldots, B_{1,n}\}$, where $B_{1,1} \geqslant B_{1,2} \geqslant \ldots \geqslant B_{1,n}$. The bandwidth of onion routers in the set $R2$ is a set $B2 = \{B_{2,1}, B_{2,2}, \ldots, B_{2,m}\}$, where $B_{2,1} \geqslant B_{2,2} \geqslant \ldots \geqslant B_{2,m}$. Obviously, the intersection of $B1$ and $B2$ belongs to EE routers.

In this scheme, the attackers compromise the existing Tor routers. The attackers know the set of entry and exit routers and can selectively choose to compromise some of those routers. In order to maximize the attack impact, the attackers can optimize the selection of victim routers. Algorithm 1 describes the attack algorithm. Given a limited percentage of Tor routers that attackers are able to compromise, Algorithm 1 iterates all possible combinations of victim routers, calculates the corresponding probability that a circuit is compromised, finds the optimal subset of Tor routers, and calculates the corresponding maximum probability that the circuit chooses a malicious entry and exit routers. In Section 4.4, we will use real-world network data collected from Tor to investigate these two schemes for protocol-level attacks and evaluate their impact on Tor.

4.3.3. Comparison with a brute force attack

A brute force attack against Tor refers to an attack that all the routers along a circuit need to be compromised in order to further link the communication relationship between Alice and Bob via a hop-by-hop fashion. In the following, we will show that if the default path length used by Tor increases, the probability that all routers along a circuit are malicious will decrease. Assume that the attacker controls k malicious routers and the bandwidths of the onion routers comprise a set $\{B_1, \ldots, B_k, B_{k+1}, \ldots, B_N\}$, where $B_1 \geqslant \ldots \geqslant B_N$. Then the onion router with bandwidth B_i is chosen with the probability $a_i = B_i / \sum_{i=1}^{k+N} B_i$, based on the weighted bandwidth routing algorithm by Tor. Let P_j be the probability that all routers along a circuit of path length j are malicious. Specifically, if the default path length is 2, the probability P_2 that all routers along a circuit are malicious becomes,

$$P_2 = \sum_{i=1}^{k} \left(a_i \sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} \right). \tag{3}$$

When the default path length is 3, P_3 can be calculated as follows,

$$P_3 = \sum_{i=1}^{k} \left(a_i \sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} \left(\sum_{l=1, l\neq i, l\neq j}^{k} \frac{a_l}{1-a_i-a_j} \right) \right). \tag{4}$$

Obviously, $\sum_{l=1, l\neq i, l\neq j}^{k} \frac{a_l}{1-a_i-a_j} < 1$ with $k < N$. Therefore, we have

$$a_i \sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} > a_i \sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} \left(\sum_{l=1, l\neq i, l\neq j}^{k} \frac{a_l}{1-a_i-a_j} \right). \tag{5}$$

and

$$P_2 > P_3. \tag{6}$$

The above procedure can be generalized and we have

$$P_m > P_n \text{ if } m < n. \tag{7}$$

The above analysis shows that the protocol-level attacks are more efficient than the brute-force attack. If the default path length adopted by Tor increases, the probability that all routers along a circuit are malicious will decrease. In comparison, the path length of circuit will not have an

[5] Compromising a large number of Tor routers is quite a challenge for the attacker. This case is included for work completeness.

impact on our investigated protocol-level attacks because these attacks only require malicious entry and exit routers rather than all onion routers along the circuit.

4.4. Evaluation based on empirical data

Now we use the analytical results to evaluate the impact of these protocol-level attacks based on empirical data collected from Tor. Our results show that by compromising a small number of onion routers, the attacker can confirm the communication relationships of a large number of flows associated with senders and receivers within Tor. Our results are consistent with the observations from Bauer et al. [36] based on small-scale experiments conducted on PlanetLab [37], which shows that the attacker can compromise approximately 46.46% circuits with 9% malicious routers.

In the scheme of compromising existing Tor routers in Section 4.3.2, by applying Algorithm 1, we can derive the probability P that a circuit chooses the malicious onion routers as entry and exit routers. Fig. 17 shows the probability P given $p \in [1\%, 10\%]$, the percentage of malicious routers within Tor. Then we select the maximum P and find the corresponding number of malicious exit onion routers, EE routers, and entry onion routers as shown in Fig. 18. In Fig. 18, all malicious EE routers are used as exit routers.

From Fig. 18, we can see that the number of entry routers is nearly equal to the number of exit routers in order to maximize P. Here we give an intuitive explanation. Assume that the number of onion routers in Tor is N, the number of malicious onion routers is $2q (2q \leqslant N)$, and the probability of selecting an onion router is $1/N$. Let e be the number of malicious entry onion routers. Note that a malicious router can be used as either an entry or exit router. The probability that a circuit chooses the malicious routers as the entry and exit routers can be calculated by,

$$P(e) = \frac{e}{N} \cdot \frac{2q - e}{N - e}. \tag{8}$$

In order to derive the value of e to maximize $P(e)$, we have

$$P(e)' = 0. \tag{9}$$

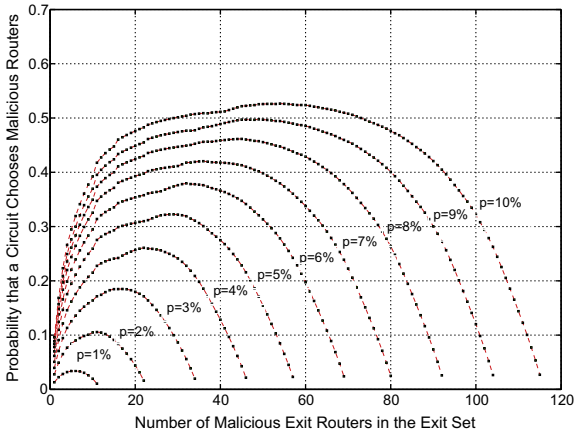


Fig. 17. Percentage of malicious routers increasing to 10%.

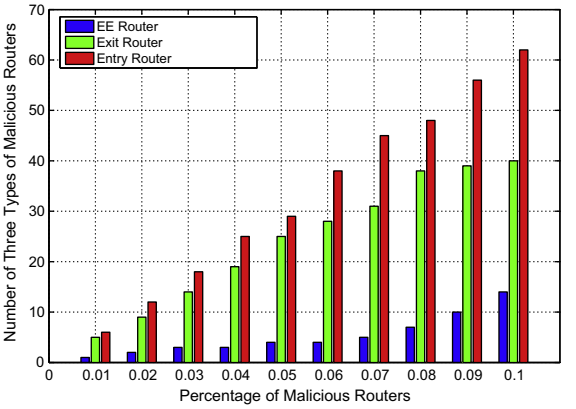


Fig. 18. Optimal number of three types of malicious routers.

Since

$$\left(\frac{e}{N} \cdot \frac{2q - e}{N - e}\right)' = \frac{2Nq(q - e)(N - e) + Na(2q - e)}{(N^2 - Ne)^2}, \tag{10}$$

then

$$2Nq(q - e)(N - e) + Na(2q - e) = 0, \tag{11}$$

$$e^2 - 2Ne + 2Nq = 0. \tag{12}$$

Therefore, e can be calculated from Eq. (12) as follows:

$$e = N\left(1 - \sqrt{1 - \frac{2q}{N}}\right). \tag{13}$$

Note that we ignore the unreasonable value $e = N\left(1 + \sqrt{1 - \frac{2q}{N}}\right) > N$. According to Taylor's theorem, $\sqrt{1 - h} = 1 - \frac{1}{2}h + o(h)$. Eq. (13) can be reformatted as,

$$e \approx N\left(1 - \left(1 - \frac{1}{2} \cdot \frac{2q}{N}\right)\right) = q. \tag{14}$$

Consequently, when the probability P that a circuit chooses the malicious onion routers as entry and exit routers reaches maximum, the number of entry routers is approximately equal to the number of exit routers.

Fig. 19 shows the results of the probability that a circuit chooses the malicious onion routers as entry and exit routers by two schemes: injecting malicious routers and compromising existing Tor routers, discussed in Section 4.3. In the first scheme, P becomes around 60.58% by injecting only 9%(115/(1164 + 115)) EE routers on Tor. In this scheme, the attacker is requested to deploy 115 extra onion routers, 9% of total Tor routers. In the second scheme, by compromising only 6% (105/(1164 + 105)) onion routers, including 56 best entry routers, 39 best EE routers as well as 10 best exit routers (56 + 39 + 10 = 105), P can reach to around 49.76%. From the data, we can see that our investigated protocol-level attacks can seriously degrade anonymity service provided by Tor.

We now compare these protocol-level attacks with the brute-force attack that needs to compromise all onion routers on Tor as discussed in Section 4.3.3. Recall that our investigated protocol-level attacks require the attacker to

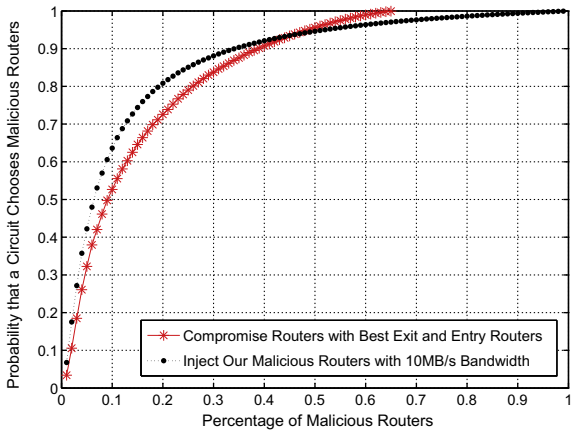


Fig. 19. Probability that a circuit chooses the malicious routers as entry and exit routers vs. percentage of malicious Tor routers.

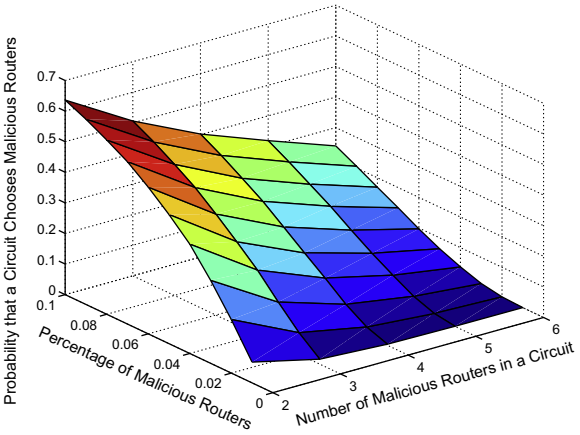


Fig. 20. Probability that a circuit chooses the malicious routers in Scheme 1 (Section 4.3.1) vs. path length and percentage of malicious routers.

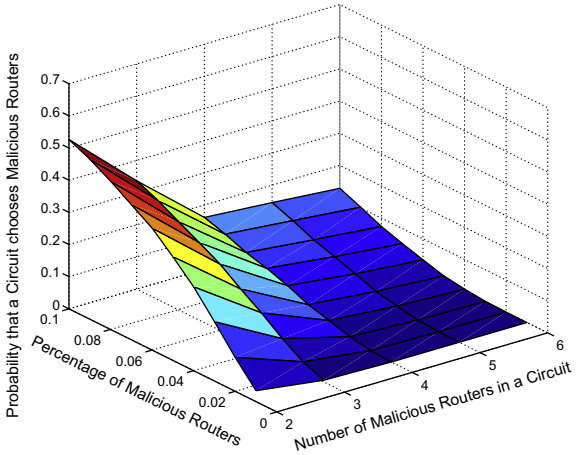


Fig. 21. Probability that a circuit chooses the malicious routers in Scheme 2 (Section 4.3.2) vs. path length and percentage of malicious routers.

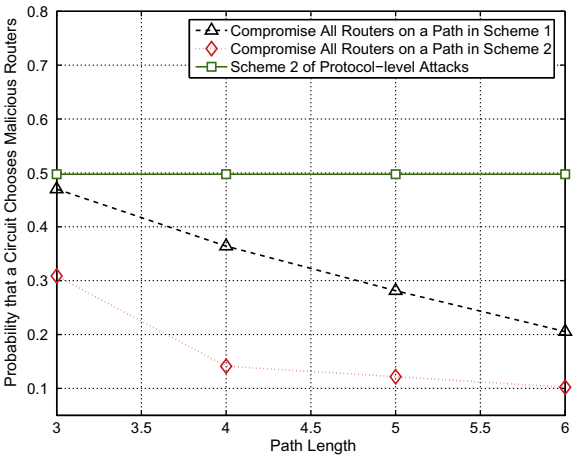


Fig. 22. Probability that a circuit consists of malicious routers vs. path length.

compromise only the exit onion routers and entry onion routers. As the path length in Tor increases, the probability of compromising two onion routers is much higher than compromising all routers on a path. We use the analysis in Section 4.3.3 to derive the probability that *OP* selects malicious routers as all the routers in a circuit as the path length increases. Figs. 20 and 21 show the probability of compromising all routers on a path in terms of the path length and the percentage of malicious Tor routers. Fig. 22 compares the probability of compromising only entry and exit routers in the protocol-level attacks and the probability of compromising all routers on a path as the path length increases from 3 to 6, given 9% malicious routes in Tor. The data show that the probability of compromising all routers on a path will decrease dramatically with the increasing path length, while the probability of compromising only entry and exit routers in terms of protocol-level attacks is constantly around 49.76% when the path length increases.

5. Guideline of countermeasures

We have demonstrated the threat of several protocol-level attacks against Tor. We now discuss possible countermeasures to these attacks. We recognize that defending against protocol-level attacks is a great challenge and Tor already suffers from a variety of timing based attacks, which are hard to defend against in light of tradeoff between system usefulness (performance) and the achieved privacy-preserving. In the following, we take the effort of discussing possible countermeasures against protocol-level attacks for the reference of designing future anonymous communication systems.

5.1. Minimizing number of compromised entry routers

Recall that protocol-level attacks require an attacker to fully control at least one entry router. To achieve this, the attacker may advertise 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 [36]. 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 [38]. A reputation-oriented defensive scheme can be developed 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 eliminate these protocol-level attacks, since the attackers may still contribute servers with high bandwidth if enough bandwidth resources are available.

5.2. 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 largely 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 incur more overhead to onion routers. A Tor relay requires using a pair of memory buffers for reading and writing data from each TCP stream and already uses much memory [39]. The length of the extra buffer is also a challenging issue given that a protocol-level attack may buffer cells and replay them much later.

We may also re-design Tor to resist the protocol-level attacks. Recall that these attacks work because it is currently impossible to detect the decryption error until the final layer is removed at the exit router. Decryption errors are found by examining the "recognized" and "integrity" fields once the final layer of encryption is removed. However, if the relay cell format were modified so that each layer of encryption has its own "recognized" and "integrity" field that can be verified upon removal of its respective layer of encryption, it would be possible to detect the decryption error at the middle node. This might prevent these attacks, since the exit node does not detect the decryption error directly. Nevertheless, one issue with this approach is that the cell size is now proportional to the hop count, which may leak information about a router's position in the circuit. Another issue with this approach is that the malicious exit router still may derive indirect information about the error, since the circuit gets destroyed.

Another way to detect these protocol-level attacks is to have Tor's clients and exit routers monitor connections with anomaly behavior. Since these attacks break connections 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.

5.3. Using bridge relays

In order to identify clients, it is necessary for the entry to tell that the previous hop is a client. This may not be always easy. Tor introduced bridge relays in the 0.2.0.3-alpha release (07-29-2007) to provide censorship resistance in case that directory servers and Tor routers are blocked by an ISP or government [40]. Bridges would appear to be indistinguishable from clients if attackers use the methods described in Section 3 for identifying clients. Therefore, using bridges can resist the protocol-level attacks to some extent. However, bridges introduce another hop to a circuit and will degrade the circuit performance. They are recommended for clients within censored regions. Moreover, according to the design document of Tor bridges [41], bridge relays are just like normal Tor relays except that they do not publish their server descriptors to the main directory authorities. If attackers inject malicious bridges into the Tor network, Tor still suffers protocol-level attacks.

6. Conclusion

In this paper, we deeply investigated several protocol-level attacks on Tor, which allow 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 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 these attacks are validated. Our data show that these attacks may drastically degrade the anonymity service that Tor provides, if the attacker is able to control a small number of Tor routers. These attacks may also be used to threaten the availability of the anonymity service by Tor. Due to Tor's fundamental design, defending against these attacks remains a challenging task that we will investigate in our future research.

Acknowledgments

We acknowledge anonymous reviewers of earlier versions of this paper. This work is supported in part by National Key Basic Research Program of China under Grants Nos. 2011CB302801 and 2010CB328104, National Natural Science Foundation of China under Grants Nos. 61272054, 61070161, 61003257, and 61070222, by US

National Science Foundation under Grants Nos. 1116644, 0942113, 0958477, 1117175 and 0943479, by China National Key Technology R&D Program under Grants Nos. 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, by the General Research Fund of the Hong Kong SAR, China Nos. CityU 114609 and CityU 114012 and CityU Applied R & D Grants (ARD) No. 9681001, and by Shenzhen (China) Basic Research Project No. JCYJ20120618115257259, and by 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 the National Science Foundation. The authors would like to acknowledge Ms. Larisa Archer for her dedicated editorial help to improve the paper.

Appendix A. Protocol-level attack algorithms

Algorithm 2 gives a formal description of our proposed protocol-level attack algorithms stated in Section 3.

Algorithm 2. Protocol-Level Attacks

Require:
(a) The attacker controls both entry onion router and exit onion router,
(b) N_c, the number of *CELL_CREATE* cells at the entry onion router,
(c) N_r, the number of *CELL_RELAY* cells at the entry onion router,
(d) N_d, the number of *CELL_CREATED* cells at the entry onion router,
(e) $\rightarrow$, the forward path,
(f) $\leftarrow$, the backward path,
(g) N'_r, the number of *CELL_RELAY* cells at the exit onion router,

Calculate N_c, N_r, and N_d at the *entry* onion router
STEP1:
if ($\overrightarrow{N_c} == 1$ & & $\overrightarrow{N_r} == 2$) & & ($\overleftarrow{N_d} == 1$ & & $\overleftarrow{N_r} == 2$) **then**
  Identify a specific created circuit from an OP
  GOTO STEP2
else
  Continue to calculate N_c, N_r, and N_d
end if
STEP2:
if $\overrightarrow{N_r} == 3$ **then**
  Identify a *CELL_RELAY_BEGIN* cell from an OP
  Continue to calculate N_r
end if
if $\overrightarrow{N_r} == 4$ **then**
Identify the first *CELL_RELAY_DATA* cell from an OP
GOTO STEP3
end if
STEP3:
Select one of strategies: (i) replaying the cell, (ii) modifying the cell, (iii) inserting the faked cell, and (iv) deleting the cell
Launch the protocol-level attack, record the timestamp of the manipulation and the source IP address
GOTO STEP4
STEP4:
Inspect each cell at the exit onion router
if Detect a cell recognition error **then**
  Record the timestamp of the cell and the destination IP address
  Confirm the communication relationship between Alice and Bob using timestamp based on Eq. (1)
end if

Appendix B. Property of $P(k)$

In this appendix, we show that $P(k)$ is an increasing function of the number of malicious routers.

$$P(k+1) = \sum_{i=1}^{k+1} a_i \sum_{j=1, j\neq i}^{k+1} \frac{a_j}{1-a_i} \tag{15}$$

$$= \sum_{i=1}^{k} a_i \sum_{j=1, j\neq i}^{k+1} \frac{a_j}{1-a_i} + a_{k+1} \sum_{j=1}^{k} \frac{a_j}{1-a_{k+1}} \tag{16}$$

$$= \sum_{i=1}^{k} a_i \left(\sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} + \frac{a_{k+1}}{1-a_i} \right) + \frac{a_{k+1}}{1-a_{k+1}} \sum_{j=1}^{k} a_j \tag{17}$$

$$= \sum_{i=1}^{k} a_i \sum_{j=1, j\neq i}^{k} \frac{a_j}{1-a_i} + \sum_{i=1}^{k} a_{k+1} \frac{a_i}{1-a_i} + \frac{a_{k+1}}{1-a_{k+1}} \sum_{j=1}^{k} a_j \tag{18}$$

$$= P(k) + a_{k+1} \sum_{i=1}^{k} \left(\frac{a_i}{1-a_i} + \frac{a_i}{1-a_{k+1}} \right). \tag{19}$$

Since

$$a_{k+1} \sum_{i=1}^{k} \left(\frac{a_i}{1-a_i} + \frac{a_i}{1-a_{k+1}} \right) > 0, \tag{20}$$

we have

$$P(k+1) > P(k). \tag{21}$$

References

[1] Q.X. Sun, D.R. Simon, Y. Wang, W. Russell, V.N. Padmanabhan, 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, 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, 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 4 (2) (1981).

[5] G. Danezis, R. Dingledine, 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, G. Tsudik, Mixing email with babel, in: Proceedings of the Network and Distributed Security Symposium (NDSS), February 1996.

[7] M. Reiter, A. Rubin, Crowds: anonymity for web transactions, ACM Transactions on Information and System Security 1 (1) (1998).

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

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

[10] B.N. Levine, M.K. Reiter, C. Wang, 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, 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, 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, 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, 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, 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] Y. Guan, X. Fu, D. Xuan, P.U. Shenoy, R. Bettati, W. Zhao, Netcamo: Camouflaging network traffic for qos-guaranteed critical applications, in: IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, Special Issue on Information Assurance, vol. 31(4), July 2001, pp. 253–265.

[17] W. Dai, Pipenet 1.1, 2009. <http://weidai.com/pipenet.txt>.

[18] D.M. Goldschlag, M.G. Reed, P.F. Syverson, Hiding routing information, in: Proceedings of Workshop on Information Hiding, 1996.

[19] X. Wang, D.S. Reeves, S.F. Wu, 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.

[20] X. Wang, 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.

[21] X. Wang, S. Chen, 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.

[22] P. Peng, P. Ning, 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.

[23] N. Kiyavash, A. Houmansadr, N. Borisov, Multi-flow attacks against network flow watermarking schemes, in: Proceedings of USENIX Security, 2008.

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

[25] S.C.X. Wang, 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.

[26] 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.

[27] R. Dingledine, N. Mathewson, P. Syverson, Tor: anonymity online, 2008. <http://tor.eff.org/index.html.en>.

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

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

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

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

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

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

[34] Tor: anonymity online, 2008. <http://tor.eff.org/>.

[35] A transparent socks proxying library, 2008. <http://tsocks.sourceforge.net>.

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

[37] The Trustees of Princeton University, Planetlab–an open platform for developing, deploying, and accessing planetary-scale services, 2008. <http://www.planet-lab.org/>.

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

[39] Theonionrouter/torfaq, 2008. <http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ>.

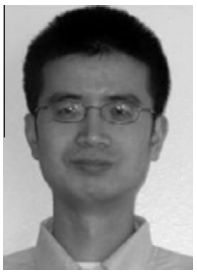
[40] The Tor Project, Inc., Tor: Bridges, 2009. <https://www.torproject.org/bridges>.

[41] R. Dingledine, Behavior for bridge users, bridge relays, and bridge authorities, November 2007. <https://git.torproject.org/checkout/tor/master/doc/spec/proposals/125-b%ridges.txt>.

Zhen Ling is a PhD candidate in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He received the BS degree in Computer Science from Nanjing Institute of Technology, China, in 2005. He joined Department of Computer Science at the City University of Hong Kong from 2008 to 2009 as a research associate, and then joined Department of Computer Science at the University of Victoria in 2011 as a visiting scholar. His research interests include network security, privacy, and forensics.

Dr. Junzhou Luo is a full Professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He received his B.S. degree in applied mathematics from Southeast University in 1982, and then got his M.S. and Ph.D. degree in computer network both from Southeast University in 1992 and in 2000 respectively. His research interests are next generation network, protocol engineering, network security and management, grid and cloud computing, and wireless LAN. He is a member of the IEEE Computer Society and co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design.

Dr. Wei Yu is an assistant professor in the Department of Computer and Information Sciences, Towson University, Towson, MD 21252. Before that, He worked for Cisco Systems Inc. for almost 9 years. He received the BS degree in Electrical Engineering from Nanjing University of Technology in 1992, the MS degree in Electrical Engineering from Tongji University in 1995, and the PhD degree in computer engineering from Texas A&M University in 2008. His research interests include cyber space security, computer network, and distributed systems.



Dr. Xinwen Fu is an assistant professor in the Department of Computer Science, University of Massachusetts Lowell. He received his BS (1995) and MS (1998) in Electrical Engineering from Xi'an Jiaotong University, China and University of Science and Technology of China respectively. He obtained his PhD (2005) in Computer Engineering from Texas A&M University. From 2005 to 2008, he was an assistant professor with the College of Business and Information Systems at Dakota State University. In summer 2008, he joined University of Massachusetts Lowell as a faculty member. His current research interests are in network security and privacy.

Prof. Weijia Jia is currently a full Professor in the Department of Computer Science and the Director of Future Networking Center, ShenZhen Research Institute of City University of Hong Kong (CityU). He received BSc and MSc from Center South University, China in 1982 and 1984 and Master of Applied Sci. and PhD from Polytechnic Faculty of Mons, Belgium in 1992 and 1993 respectively, all in Computer Science. He joined German National Research Center for Information Science (GMD) in Bonn (St. Augustine) from 1993 to 1995 as a research fellow. In 1995, he joined Department of Computer Science, CityU as an assistant professor. His research interests include next generation wireless communication, protocols and heterogeneous networks; distributed systems, multicast and anycast QoS routing protocols.

Dr. Wei Zhao is currently the Rector of the University of Macau. Before joining the University of Macau, he served as the Dean of the School of Science at Rensselaer Polytechnic Institute. Between 2005 and 2006, he served as the director for the Division of Computer and Network Systems in the US National Science Foundation when he was on leave from Texas A&M University, where he served as Senior Associate Vice President for Research and Professor of Computer Science. He was the founding director of the Texas A&M Center for Information Security and Assurance, which has been recognized as a Center of Academic Excellence in Information Assurance Education by the National Security Agency. Dr. Zhao completed his undergraduate program in physics at Shaanxi Normal University, Xian, China, in 1977. He received the MS and PhD degrees in Computer and Information Sciences at the University of Massachusetts at Amherst in 1983 and 1986, respectively. Since then, he has served as a faculty member at Amherst College, the University of Adelaide, and Texas A&M University. As an elected IEEE fellow, Wei Zhao has made significant contributions in distributed computing, real time systems, computer networks, and cyber space security.