A

Seminar-II Report

on

TOR

Submitted in Partial Fulfillment of
the Requirements for the Fourth Year
of

Bachelor of Engineering

in

Computer Engineering

to

North Maharashtra University, Jalgaon

Submitted by

Bhavesh Mohan Patil

Under the Guidance of

Prof. Dr. Girish K. Patnaik

DEPARTMENT OF COMPUTER ENGINEERING
SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY,
BAMBHORI, JALGAON - 425 001 (MS)

2016-2017

SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY, BAMBHORI, JALGAON - 425 001 (MS)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the SEMINAR-II entitled *TOR*, submitted by

Bhavesh Mohan Patil

in partial fulfillment of the Fourth Year of *Bachelor of Engineering* in *Computer Engineering* has been satisfactorily carried out under my guidance as per the requirement of North Maharashtra University, Jalgaon.

Date: October 5, 2016
Place: Jalgaon

Prof. Dr. Girish K. Patnaik
Guide

Prof. Dr. Girish K. Patnaik
Head

Prof. Dr. K. S. Wani
Principal

Acknowledgements

I thank God almighty for blessing me throughout the work. I would like to thank all those who have contributed to the completion of the case study and helped me with valuable suggestions for improvement.

I would like to thank my guide, Prof. Dr. Girish K. Patnaik for all help and support, for providing me with best facilities and atmosphere for the creative work guidance and encouragement.

I greatly thankful to our principal Prof. Dr. K. S. Wani for providing us facilities. I would also want to thank Mr.Sandip Patil and Mr.Satpal Rajput for there support and guidance during Latex workshop. I thank all staff members of my college and friends for extending their cooperation during my case study. Above all I would like to thank my parents without whose blessings,I would not have been able to accomplish my goal.

Bhavesh Mohan Patil

Contents

List of Figures

Abstract

TOR is a popular privacy enhancing system that is designed to protect the privacy of Internet users. TOR uses onion routing which is implemented by encryption in the application layer of a communication protocol stack, nested like the layers of an onion. Onion routing makes it more difficult for Internet activity to be traced back to the user. TOR is a circuit-based low-latency anonymous communication service. An anonymous communication service is a peer-to-peer distributed application in which the nodes or participants are anonymous or pseudonymous. Anonymity of participants is usually achieved by special routing overlay networks that hide the physical location of each node from other participants.

Chapter 1

Introduction

Privacy is the ability of an individual or group to seclude themselves, or information about themselves, and thereby express themselves selectively. The boundaries and content of what is considered private differ among cultures and individuals, but share common themes. When something is private to a person, it usually means that something is inherently special or sensitive to them. Onion routing provide anonymous communication service for individual person to maintain their privacy. Also it helps to protects us against a common form of Internet surveillance known as 'traffic analysis'.

This chapter gives introduction to TOR technology. Section 1.1 describes TOR. Section 1.2 describes summary.

1.1 TOR

TOR is free software for enabling anonymous communication. The name is derived from an acronym for the original software project name "The Onion Router". TOR directs Internet traffic through a free, worldwide, volunteer network consisting of more than seven thousand relays to conceal a user's location and usage from anyone conducting network surveillance or traffic analysis.[7]

TOR's use is intended to protect the personal privacy of users, as well as their freedom and ability to conduct confidential communication by keeping their Internet activities from being monitored. TOR enables users to surf the Internet, chat and send instant messages anonymously, and is used by a wide variety of people for both licit and illicit purposes.[2]

Onion routing is implemented by encryption in the application layer of a communication protocol stack, nested like the layers of an onion. TOR encrypts the data, including the destination IP address, multiple times and sends it through a virtual circuit comprising successive, randomly selected TOR relays. Each relay decrypts a layer of encryption to reveal only the next relay in the circuit in order to pass the remaining encrypted data on to it. The final relay decrypts the innermost layer of encryption and sends the original

data to its destination without revealing, or even knowing, the source IP address. Because the routing of the communication is partly concealed at every hop in the TOR circuit, this method eliminates any single point at which the communicating peers can be determined through network surveillance that relies upon knowing its source and destination.

TOR is not meant to completely solve the issue of anonymity on the web. Instead, it simply focuses on protecting the transportation of data so that certain sites cannot trace back the data to a given location. It is still possible for sites to backtrack to a location. TOR is not designed to erase a user's tracks but to simply make it less likely for sites to trace back to them.

1.2 Summary

This chapter gives primary information about Onion routing and TOR technology. It presents an overview about onion routing. Next chapter presents literature survey, history and related work about onion routing.

Chapter 2

Literature Survey

The core principle of TOR, 'onion routing', was developed in the mid-1990s by United States Naval Research Laboratory employees, mathematician Paul Syverson and computer scientists Michael G. Reed and David Goldschlag, with the purpose of protecting U.S. intelligence communications online. Initial work on Onion Routing begins, funded by ONR. Many ideas tossed about, some best forgotten, some disappear only to resurface in the generation 2 system, e.g., all onions are one hop. Contrastingly, the first design does not include any mixing. Realtime mixing is added to the design in the middle of Spring 1996 and does not disappear again until generation 2.[1]

In this chapter, Literature Survey is presented. Section 2.1 gives history, section 2.2 gives related work, Section 2.3 gives summary.

2.1 History

Another early discussion idea that was not reflected in published or deployed design till much later: Discussion in April and May of using DH keys rather than sending the onion key. The idea was not to have these in one hop onions as in generation 2, but to include public DH keys from the originator in the onion layers. These were to be combined with published public DH keys. Interestingly, this would still yield perfect forward secrecy if the private DH keys of the onion routers were rotated on.

Design using Onion Routing for location hidden use of cellular (mobile) phones and for private control of location information in active badges or other location tracking devices is published at the Security Protocols Workshop in April 1997. Early specific examples of both location hidden server and rendezvous points.

In 1999, Alan Berman Research Publication Award given for 'Anonymous Connections and Onion Routing'. This paper provides the most detailed specification published of generation 1 Onion Routing, although some features are added later. Work on Onion Routing

development is suspended. There is no new funding for it, plus most principals and all developers have left NRL for other pursuits. Nonetheless, research and analysis work continues.

In 2003, Funding from ONR for generation 2 development and deployment, DARPA for building in resource management and fault tolerance, and NRL internal funding from ONR for building survivable hidden servers. In October, TOR network is deployed, and TOR code is released under the free and open MIT license. Both the network and code development are managed through the TOR development site. By the end of 2003, the network has about a dozen volunteer nodes, mostly in the US with one in Germany.

In 2004, Location hidden services are deployed in the spring when the hidden wiki is set up.

2.2 Related Work

The alpha version of TOR, developed by Syverson and computer scientists Roger Dingledine and Nick Mathewson and then called The Onion Routing project, or TOR project, launched on 20 September 2002. The first public release occurred a year later.

13 August 2004, Syverson, Dingledine and Mathewson presented 'TOR: The Second-Generation Onion Router' at the 13th USENIX Security Symposium. In 2004, the Naval Research Laboratory released the code for TOR under a free license, and the Electronic Frontier Foundation (EFF) began funding Dingledine and Mathewson to continue its development.

In December 2006, Dingledine, Mathewson and five others founded The Tor Project, a Massachusetts-based 501(c)(3) research-education nonprofit organization responsible for maintaining TOR. The EFF acted as The Tor Project's fiscal sponsor in its early years, and early financial supporters of The TOR Project included the U.S. International Broadcasting Bureau, Internews, Human Rights Watch, the University of Cambridge, Google, and Netherlands-based Stichting NLnet.

In December 2015, The TOR Project announced that it had hired Shari Steele as its new executive director. Steele had previously led the Electronic Frontier Foundation for 15 years, and in 2004 spearheaded EFF's decision to fund TOR's early development. One of her key stated aims is to make TOR more user-friendly in order to bring wider access to anonymous web browsing.

In July 2016 the complete board of the TOR Project resigned, and announced a new board, made up of Matt Blaze, Cindy Cohn, Gabriella Coleman, Linus Nordberg, Megan Price and Bruce Schneier.

2.3 Summary

This chapter presents literature, history and related work of onion routing. Previous research work related to onion routing is presented in this chapter. This mainly includes stepwise development of onion routing. Next chapter describes methodology.

Chapter 3

Methodology

TOR aims to conceal its users identities and their online activity from surveillance and traffic analysis by separating identification and routing. It is an implementation of onion routing, which encrypts and then randomly bounces communications through a network of relays run by volunteers around the globe. These onion routers employ encryption in a multi-layered manner (hence the onion metaphor) to ensure perfect forward secrecy between relays, thereby providing users with anonymity in network location. That anonymity extends to the hosting of censorship-resistant content by TOR's anonymous hidden service feature. Furthermore, by keeping some of the entry relays (bridge relays) secret, users can evade Internet censorship that relies upon blocking public TOR relays.[6]

This chapter describes Methology for onion routing. Section 3.1 describes existing techniques, Section 3.2 describes therotical explanation, Section 3.3 describes algorithm, Section 3.4 describes summary.

3.1 Existing Techniques

The growing demand for anonymous communication has lead to a number of different approaches and designs. Starting with the simpler ones, this section ends with TOR, being the most sophisticated.

3.1.1 Proxies

A Proxy is a server that accepts requests and forwards them in the servers name to their final destination. By doing so, the origin of the request is hidden. There are many different kinds of proxies; however, the focus will lie on Web-proxies, i.e., proxies that are able to interpret and relay HTTP(S) traffic.After receiving the response to the initial request, the proxy forwards this response to the request initiator.

Many companies use proxies for security and performance reasons to forward requests

and cache Web server responses. Connections to proxy servers are generally unencrypted, connections from proxy servers to Web servers are always unencrypted. The exception to this rule is, of course, SSL encrypted traffic. Some very common proxies like squid and Apaches mod proxy even add a HTTP header field revealing the initiator by default.

There are a couple of anonymous proxies on the Web, some of them do encrypt the connection between initiator and proxy. The most prominent example is Anonymizer.com, which offers encrypted connections to commercial proxy servers to protect a clients privacy.The security in terms of anonymity regarding a single proxy server is very poor, even though the identity is hidden to the destination Web server and to routers between the Web server and the proxy.

Figure for proxy server is shown below:

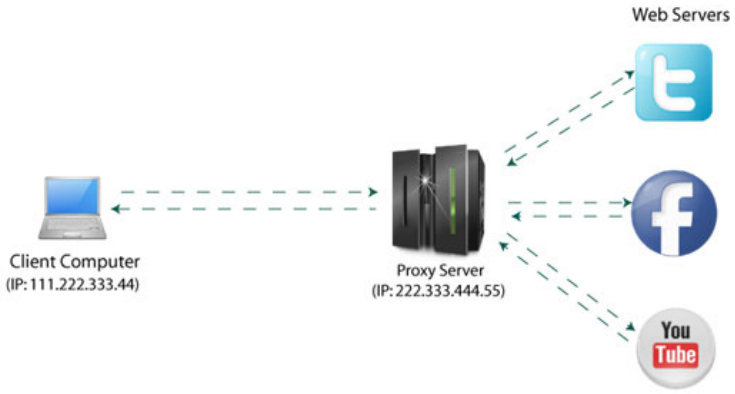

Figure 3.1: Proxy Server

3.1.2 Chaumian Mix-nets

David Chaum introduced the so called Mix-net system in 1981. A Mix-net is a chain of proxy or relay like servers. The sender identity is hidden to the receiver by relaying messages through a chain of proxies. In order to keep the proxies from reading the original content of the message and thus linking sender to recipient, the messages are encrypted. The following example will consider a chain of 3 proxies; however, the length of the chain may be freely chosen.

In order to protect the Mix-net from adversaries correlating incoming and outgoing traffic, thus eventually correlating sender and receiver, every proxy in the chain mixes the incoming messages and permutes the order in which messages are relayed. In general, the more connections a mix is permuting, the safer a single connection will be. Mix-nets are the basis for the majority of todays anonymity architectures.

Mix-nets offer improved anonymity compared to single proxies. However, for an adversary with sufficient power, it is not difficult to break the system, if the Mix-net consists of a fixed,

small set of mixes.

3.1.3 JAP

The AN.ON project of the University of Dresden developed the Java Anon Proxy, now called JAP. It is an implementation of Chaums Mix-nets and is mainly used in Germany. JAP uses a chain of mixes to hide the identity of a client to any Web server or adversary observing network traffic. Browser requests are encrypted and passed along the chain until the request is finally sent to the Web server. The response is sent back on the same chain, traveling in the reverse direction until it reaches the browser.

This approach provides anonymity by mixing the clients traffic with other clients traffic. After being delayed, mixed, and forwarded, it is difficult for an adversary to correlate exit traffic to the initiating client. One of the problems with this approach is that it does not scale very well. As user numbers increase, the quality of service in terms of latency and throughput decreases.

JAP and JonDonym both lack efficient measures against government-level blocking. As the number of available mixes is very limited, it is easy for restrictive countries or ISPs to block access to the mixes.

3.1.4 The Second-Generation Onion Router

TOR is the most popular anonymizing system used nowadays, being actively developed by The TOR Project. It offers a local proxy service that can relay TCP connections from proxy-aware applications. There are tools that even allow non proxy-aware applications to access TOR, e.g., Socat.

Data is transported in fixed-size cells in order to offer some protection from traffic analysis. TOR focuses on its low-latency property and because of that does not introduce artificial delays for mixing nor does it use cover traffic. The onion proxy uses central directory servers that provide information about available onion routers. This information includes server capacity, IP address, and public keys, which is needed to establish virtual circuits through the TOR network.[3]

TORs design was primarily chosen to provide anonymity, but not to resist censorship. Many people, however, are currently using TOR very effectively to circumvent government-level censorship. But, due to the limited number of onion routers and directory servers, it would be easy for the censors to block access to the TOR network based on IP blacklists.

3.2 Therotical Explanation

The TOR network is an overlay network; each onion router (OR) runs as a normal user-level process without any special privileges. Each onion router maintains a TLS connection to every other onion router. Each user runs local software called an onion proxy (OP) to fetch directories, establish circuits across the network, and handle connections from user applications. These onion proxies accept TCP streams and multiplex them across the circuits. The onion router on the other side of the circuit connects to the requested destinations and relays data.

Each onion router maintains a long-term identity key and a short-term onion key. The identity key is used to sign TLS certificates, to sign the ORs router descriptor (a summary of its keys, address, bandwidth, exit policy, and so on), and (by directory servers) to sign directories. The onion key is used to decrypt requests from users to set up a circuit and negotiate ephemeral keys.

The TLS protocol also establishes a shortterm link key when communicating between ORs. Short-term keys are rotated periodically and independently, to limit the impact of key compromise.

3.2.1 Cells

Onion routers communicate with one another, and with users OPs, via TLS connections with ephemeral keys. Using TLS conceals the data on the connection with perfect forward secrecy, and prevents an attacker from modifying data on the wire or impersonating an OR. Traffic passes along these connections in fixed-size cells. Each cell is 512 bytes, and consists of a header and a payload. The header includes a circuit identifier (circID) that specifies which circuit the cell refers to (many circuits can be multiplexed over the single TLS connection), and a command to describe what to do with the cells payload.

Relay cells have an additional header (the relay header) at the front of the payload, containing a streamID (stream identifier: many streams can be multiplexed over a circuit); an end-to-end checksum for integrity checking; the length of the relay payload; and a relay command. The entire contents of the relay header and the relay cell payload are encrypted or decrypted together as the relay cell moves along the circuit, using the 128-bit AES cipher in counter mode to generate a cipher stream.

Figure for cell format is shown below:

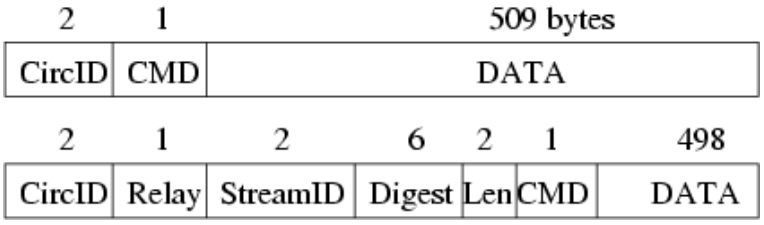

Figure 3.2: Cell Format

3.2.2 Circuits and Streams

Onion Routing originally built one circuit for each TCP stream. Because building a circuit can take several tenths of a second (due to public-key cryptography and network latency), this design imposed high costs on applications like web browsing that open many TCP streams.

In TOR, each circuit can be shared by many TCP streams. To avoid delays, users construct circuits preemptively. To limit linkability among their streams, users OPs build a new circuit periodically if the previous ones have been used, and expire old used circuits that no longer have any open streams. OPs consider rotating to a new circuit once a minute: thus even heavy users spend negligible time building circuits, but a limited number of requests can be linked to each other through a given exit node. Also, because circuits are built in the background, OPs can recover from failed circuit creation without harming user experience.

Figure for circuit creation is shown below:

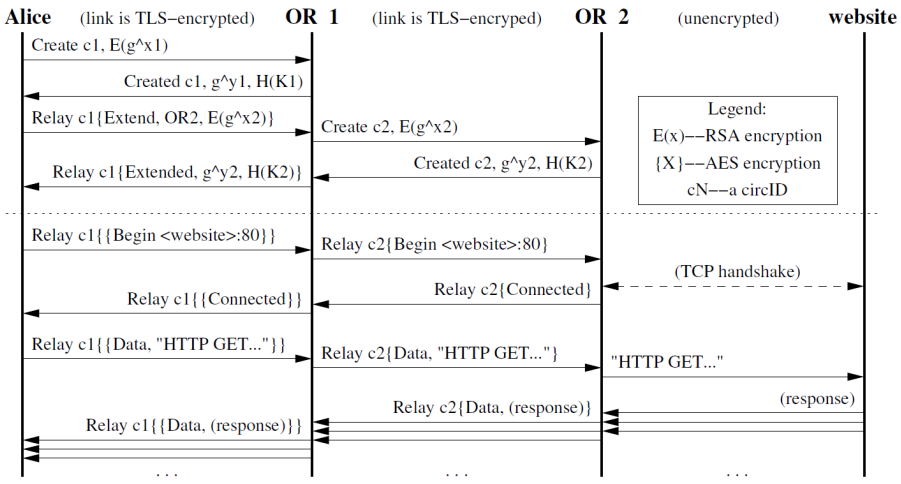

Figure 3.3: Alice Builds a Two-Hop Circuit and Begins Fetching a Web Page

Constructing a Circuit

A users OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time. To begin creating a new circuit, the OP (call her Alice)

sends a create cell to the first node in her chosen path (call him Bob). (She chooses a new circID CAB not currently used on the connection from her to Bob.) The create cells payload contains the first half of the Diffie-Hellman handshake (gx), encrypted to the onion key of the OR (call him Bob). Bob responds with a created cell containing gy along with a hash of the negotiated key K = gxy.Once the circuit has been established, Alice and Bob can send one another relay cells encrypted with the negotiated key.[1]

To extend the circuit further, Alice sends a relay extend cell to Bob, specifying the address of the next OR (call her Carol), and an encrypted gx2 for her. Bob copies the half-handshake into a create cell, and passes it to Carol to extend the circuit. (Bob chooses a new circID CBC not currently used on the connection between him and Carol. Alice never needs to know this circID; only Bob associates CAB on his connection with Alice to CBC on his connection with Carol.) When Carol responds with a created cell, Bob wraps the payload into a relay extended cell and passes it back to Alice. Now the circuit is extended to Carol, and Alice and Carol share a common key K2 = gx2y2.

To extend the circuit to a third node or beyond, Alice proceeds as above, always telling the last node in the circuit to extend one hop further. This circuit-level handshake protocol achieves unilateral entity authentication (Alice knows shes handshaking with the OR, but the OR doesnt care who is opening the circuit Alice uses no public key and remains anonymous) and unilateral key authentication (Alice and the OR agree on a key, and Alice knows only the OR learns it). It also achieves forward secrecy and key freshness. More formally, the protocol is as follows (where EPKBob () is encryption with Bobs public key, H is a secure hash function, and — is concatenation):

Alice to Bob : EPKBob (gx)

Bob to Alice : gy,H(K—handshake)

3.2.3 Relay Cells

Once Alice has established the circuit (so she shares keys with each OR on the circuit), she can send relay cells. Upon receiving a relay cell, an OR looks up the corresponding circuit, and decrypts the relay header and payload with the session key for that circuit. If the cell is headed away from Alice the OR then checks whether the decrypted cell has a valid digest (as an optimization, the first two bytes of the integrity check are zero, so in most cases we can avoid computing the hash).

If valid, it accepts the relay cell and processes it as described below. Otherwise, the OR looks up the circID and OR for the next step in the circuit, replaces the circID as appropriate, and sends the decrypted relay cell to the next OR. (If the OR at the end of the circuit receives an unrecognized relay cell, an error has occurred, and the circuit is torn

down). OPs treat incoming relay cells similarly: they iteratively unwrap the relay header and payload with the session keys shared with each OR on the circuit, from the closest to farthest. If at any stage the digest is valid, the cell must have originated at the OR whose encryption has just been removed.

To construct a relay cell addressed to a given OR, Alice assigns the digest, and then iteratively encrypts the cell payload (that is, the relay header and payload) with the symmetric key of each hop up to that OR. Because the digest is encrypted to a different value at each step, only at the targeted OR will it have a meaningful value. This leaky pipe circuit topology allows Alices streams to exit at different ORs on a single circuit. Alice may choose different exit points because of their exit policies, or to keep the ORs from knowing that two streams originate from the same person.

3.2.4 Opening and Closing Streams

When Alices application wants a TCP connection to a given address and port, it asks the OP (via SOCKS) to make the connection. The OP chooses the newest open circuit (or creates one if needed), and chooses a suitable OR on that circuit to be the exit node (usually the last node, but maybe others due to exit policy conflicts; see Section 6.2.) The OP then opens the stream by sending a relay begin cell to the exit node, using a new random streamID. Once the exit node connects to the remote host, it responds with a relay connected cell. Upon receipt, the OP sends a SOCKS reply to notify the application of its success.

The OP now accepts data from the applications TCP stream, packaging it into relay data cells and sending those cells along the circuit to the chosen OR. Theres a catch to using SOCKS, howeversome applications pass the alphanumeric hostname to the TOR client, while others resolve it into an IP address first and then pass the IP address to the TOR client. If the application does DNS resolution first, Alice thereby reveals her destination to the remote DNS server, rather than sending the hostname through the TOR network to be resolved at the far end. Common applications like Mozilla and SSH have this flaw.

Closing a TOR stream is analogous to closing a TCP stream: it uses a two-step handshake for normal operation, or a onestep handshake for errors. If the stream closes abnormally, the adjacent node simply sends a relay teardown cell. If the stream closes normally, the node sends a relay end cell down the circuit, and the other side responds with its own relay end cell. Because all relay cells use layered encryption, only the destination OR knows that a given relay cell is a request to close a stream. This two-step handshake allows TOR to support TCP-based applications that use half-closed connections.

3.2.5 Integrity Checking on Streams

Because the old Onion Routing design used a stream cipher without integrity checking, traffic was vulnerable to a malleability attack: though the attacker could not decrypt cells, any changes to encrypted data would create corresponding changes to the data leaving the network. This weakness allowed an adversary who could guess the encrypted content to change a padding cell to a destroy cell; change the destination address in a relay begin cell to the adversarys webserver; or change an FTP command from dir to rm *. (Even an external adversary could do this, because the link encryption.

Because TOR uses TLS on its links, external adversaries cannot modify data. Addressing the insider malleability attack, however, is more complex. We could do integrity checking of the relay cells at each hop, either by including hashes or by using an authenticating cipher mode like EAX, but there are some problems.

First, these approaches impose a message-expansion overhead at each hop, and so we would have to either leak the path length or waste bytes by padding to a maximum path length. Second, these solutions can only verify traffic coming from Alice: ORs would not be able to produce suitable hashes for the intermediate hops, since the ORs on a circuit do not know the other ORs session keys. Third, TOR design is vulnerable to end-to-end timing attacks; so tagging attacks performed within the circuit provide no additional information to the attacker.[5]

Thus, we check integrity only at the edges of each stream. (Remember that in our leaky-pipe circuit topology, a streams edge could be any hop in the circuit.) When Alice negotiates a key with a new hop, they each initialize a SHA-1 digest with a derivative of that key, thus beginning with randomness that only the two of them know. Then they each incrementally add to the SHA-1 digest the contents of all relay cells they create, and include with each relay cell the first four bytes of the current digest. Each also keeps a SHA-1 digest of data received, to verify that the received hashes are correct.

3.2.6 Rate Limiting and Fairness

Volunteers are more willing to run services that can limit their bandwidth usage. To accommodate them, Tor servers use a token bucket approach to enforce a long-term average rate of incoming bytes, while still permitting short-term bursts above the allowed bandwidth.

Because the TOR protocol outputs about the same number of bytes as it takes in, it is sufficient in practice to limit only incoming bytes. With TCP streams, however, the correspondence is not one-to-one: relaying a single incoming byte can require an entire 512-byte cell. (We cant just wait for more bytes, because the local application may be awaiting a reply). Therefore, we treat this case as if the entire cell size had been read, regardless of

the cells fullness.

3.2.7 Congestion Control

Even with bandwidth rate limiting, we still need to worry about congestion, either accidental or intentional. If enough users choose the same OR-to-OR connection for their circuits, that connection can become saturated. For example, an attacker could send a large file through the Tor network to a webserver he runs, and then refuse to read any of the bytes at the webserver end of the circuit.

Without some congestion control mechanism, these bottlenecks can propagate back through the entire network. We dont need to reimplement full TCP windows (with sequence numbers, the ability to drop cells when were full and retransmit later, and so on), because TCP already guarantees in-order delivery of each cell. We describe our response below.

3.2.8 Algorithm

In algorithm section intermediate node algorithm and random walk algorithm are described. This algorithms are use to select intermediate node in TOR network and shortest path from source to destination.

- *Intermediate Node Algorithm*

In our first algorithm, a node is randomly selected that is neither the source nor the destination node. Using the adjacency list information, we calculate the shortest path from the source node to the randomly selected node and the shortest path from the randomly selected node to the destination node. If the total number of hops necessary to pass through this randomly selected node is within our Max and Min boundaries, then we are done. Otherwise, we continue to select other nodes in the graph to be the random node in the middle. Naturally, when this loop iterates the random selection of the node would not select any previously selected node that failed to meet the boundary limits. The pseudocode below is probably easier to follow:

do
X = randomly select a node in the graph;
/* where X != src, X != dest and X does not equal any
previously rejected nodes */
total number of hops = shortest-path from src to X
+ shortest-path from dest to X;
while(Min > total number of hops or total number of hops > Max);

- *Random Walk Algorithm*

Our second algorithm combines random walking of the graph with a shortest-path segment to keep the total number of hops under the pre-specified maximum. Unlike the previous algorithm, this algorithm requires that all-pairs shortest-paths is computed (instead of single-source shortest-path). The algorithm is as follows. A route length is randomly selected. In our algorithm below, we selected a number, Length, between Min and Max, where Min is equal to 6 and Max is equal to twice the number of nodes in the graph (2*20). Then, a coin is tossed to determine whether the random walk from the source or the random walk from the destination takes one random hop to an adjacent node. The coin may or may not be a fair coin. That is, it may be desirable to have more random walking close to the source node or close to the destination node. In either case the coin toss can easily reflect this preference. In the algorithm below, we use a fair coin to select which random walk proceeds. After an adjacent node is selected, the total number of hops including this new node plus the shortest-path to connect the two random walks is computed before the node is incorporated into the route. The above steps continue until the route length is reached or once the next random step exceeds the route length. If the later occurs, then we simply back up one hop (do not include the last random hop in the route). The pseudocode for this algorithm is below (for simplicity, storing the actual route in a data structure has not been included).

```
LengthFromSrc = 0;
LengthFromDest = 0;
TotalNumberHops = 0;
X = SRC; /*Last Node Visited from Random walk starting at SRC;*/
Y = DEST; /*Last Node Visited from Random walk starting at DEST;*/
/* Randomly select a route length */
do
Length = rand( )
while( Length < Min );
while( TotalNumberHops < Length )
Next = Toss Coin to Pick Random Walk from SRC or from DEST;
if( Next == RandWalkFromSrc )
Z = Randomly select an adjacent node to X;
TotalNumberHops = 1 + LengthFromSrc + LengthFromDest + shortest-path from Z to Y;
if( TotalNumberHops > Length )
break;
X = Z; /*include the node in the route*/
```

Store X in the route data structure

LengthFromSrc++;

else /* Next = RandWalkFromDest */

Z = Randomly select an adjacent node to Y;

TotalNumberHops = 1 + LengthFromSrc + LengthFromDest + shortest-path from Z to X;

if(TotalNumberHops > Length)

break;

Y = Z;

Store Y in the route data structure

LengthFromDest++;

3.3 Summary

This chapter describes existing techniques, therotical explanation and algorithm for TOR technology. Therotical explanation is useful to understand the structure and working of onion routing. Next chapter presents discussion on onion routing.

Chapter 4

Discussion

The TOR community works on new features, additional security mechanisms, tools and applications to make the system better. Nevertheless, according to some studies there are issues with the TOR environment by design, which might leak critical information regarding users privacy. Redirecting users to special servers via telecoms operators can constitute a man-in-the-middle attack, as an example. It can be done by intercepting the traffic between a TOR user and the legitimate server, although it has been argued that only the US National Security Agency (NSA) has this sort of capability.

This chapter presents discussion on onion routing. Section 4.1 describes comparison between anonymous services, Section 4.2 describes advantages and disadvantages, Section 4.3 describes users of TOR, Section 4.4 describes summary.

4.1 Comparison Between Anonymous Services

In this section onion routing compared with anonoymous services. The information provided by this comparison can be used to choose better anonoymous service.

4.1.1 Comparison Between Onion Routing and Mix-net

Mix networks get their security from the mixing done by their component mixes, and may or may not use route unpredictability to enhance security. Onion routing networks primarily get their security from choosing routes that are difficult for the adversary to observe, which for designs deployed to date has meant choosing unpredictable routes through a network. And onion routers typically employ no mixing at all.

Mixes are also usually intended to resist an adversary that can observe all traffic everywhere and, in some threat models, to actively change traffic. Onion routing assumes that an adversary who observes both ends of a communication path will completely break the anonymity of its traffic. Thus, onion routing networks are designed to resist a local adversary,

one that can only see a subset of the network and the traffic on it.

Onion Routing gets its security from the fact (or assumption) that it is difficult for an adversary to position itself on networks such that it is able to view all the nodes in the route.But if an adversary is able to see the entire path, Onion Routing loses its security. Mixing however, is specifically designed to provide security even if an adversary can see the entire path. A Mix Node must collect more than one message before sending any out - otherwise the node is behaving as an onion router node with a time delay. The more messages collected, the more uncertainty is introduced as to which message went where.

4.1.2 Comparison of TOR and I2P

TOR and Onion Routing are both anonymizing proxy networks, allowing people to tunnel out through their low latency mix network. The two primary differences between TOR and I2P are again related to differences in the threat model and the out-proxy design (though TOR supports hidden services as well). In addition, Tor takes the directory-based approach - providing a centralized point to manage the overall 'view' of the network, as well as gather and report statistics, as opposed to I2P's distributed network database and peer selection.

The I2P/Tor outproxy functionality does have a few substantial weaknesses against certain attackers - once the communication leaves the mixnet, global passive adversaries can more easily mount traffic analysis. In addition, the outproxies have access to the cleartext of the data transferred in both directions, and outproxies are prone to abuse, along with all of the other security issues we've come to know and love with normal Internet traffic.

TOR is Much bigger user base; much more visibility in the academic and hacker communities; benefits from formal studies of anonymity, resistance, and performance; has a non-anonymous, visible, university-based leader. I2p Designed and optimized for hidden services, which are much faster than in TOR. TOR Designed and optimized for exit traffic, with a large number of exit nodes. TOR is Big enough that it has had to adapt to blocking and DOS attempts on other hand I2p Small enough that it hasn't been blocked or DOSed much, or at all.

4.1.3 Comparison Between Onion Routing and Crowds

The difference here is that with Crowds each pair of nodes has a different symmetric key used to encrypt the data between nodes. A path key is established for each path that will encrypt the actual request and response. This is forwarded along the path by using the symmetric key of each pair of nodes. But there is no layering here. With the randomness of Crowds you wouldn't want to encrypt the data 10 times. The latency could grow out of control.

Also, TOR does not require an account to establish an initial connection to the network. Now once Crowds begins using DH this account might no longer be necessary, and anonymous connections to a crowd could be possible.The other main difference is how circuits are chosen. While TOR relies on choosing a circuit of 3 Crowds relies on randomness to determine how many nodes are taken. It could be 2, 3, or 15. Depends on that coin flip.

4.2 Advantages and Disadvantages

Advantages of Onion routing are as follows:

1. TOR is free software for enabling anonymous communication.
2. TOR enables users to surf the Internet, chat and send instant messages anonymously.
3. TOR aims to conceal its users identities and their online activity from surveillance and traffic analysis by separating identification and routing.
4. It is a decentralized system that allows users to connect through a network of relays rather than making a direct connection.
5. The benefit of this method is that your IP address is hidden from the sites you visit by bouncing your connection from server to server at random.

Disadvantages of Onion routing are as follows:

1. TOR cannot and does not attempt to protect against monitoring of traffic at the boundaries of the Tor network.
2. TOR does not provide protection against end-to-end correlation.
3. TOR slowing down the browsing because of the numerous hops your data is relayed through.

4.3 Who Uses TOR

TOR was originally designed, implemented, and deployed as a third-generation onion routing project of the Naval Research Laboratory. It was originally developed with the U.S. Navy in mind, for the primary purpose of protecting government communications. Today, it is used every day for a wide variety of purposes by the military, journalists, law enforcement officers, activists, and many others.

Users of onion routing are as follows:

Normal people use TOR:

1. They protect their privacy from unscrupulous marketers and identity thieves.
2. They protect their communications from irresponsible corporations.
3. They research sensitive topics.
4. They protect their children online.
5. They skirt surveillance.

Journalists and their audience use TOR:

1. Reporters without Borders tracks Internet prisoners of conscience and jailed or harmed journalists all over the world.
2. Citizen journalists in China use TOR to write about local events to encourage social change and political reform.
3. Citizens and journalists in Internet black holes use TOR to research state propaganda and opposing viewpoints

Law enforcement officers use TOR:

1. Online surveillance
2. Sting operations
3. Truly anonymous tip lines

Business executives use TOR:

1. Security breach information clearinghouses
2. Seeing your competition as your market does
3. Keeping strategies confidential
4. Accountability

Militaries use TOR:

1. Field agents
2. Hidden services
3. Intelligence gathering

4.4 Summary

This chapter compares between various anonymous services, advantages and disadvantages and users of TOR. This chapter provides an overview on advantages and disadvantages of onion routing. Next chapter presents conclusion on onion routing.

Chapter 5

Conclusion

TOR provides insight into the challenges of deploying a real anonymity service, this work will encourage additional research aimed at providing tools to enforce accountability while preserving strong anonymity properties, protecting users from unknowingly disclosing sensitive/identifying information, and fostering participation from a highly diverse set of routers. It can therefore be concluded that there are many under-researched issues related to the TOR network that need to be taken into account both by its users and by governments designing national policies and reviewing national legal frameworks.

Bibliography

[1] Dingledine, Roger (20 September 2002). 'Pre-alpha: run an onion proxy now!'. or-dev (Mailing list), 17 July 2008.

[2] Glater, Jonathan D. (25 January 2006). 'Privacy for People Who Don't Show Their Navels'. The New York Times, 13 May 2011.

[3] Dan Goodin, Arstechnica, 22 January 2014, Scientists detect 'spoiled onions' trying to sabotage Tor privacy network'.

[4] Keith D. Watson, The Tor Network: A Global Inquiry into the Legal Status of Anonymity Networks, 2012.

[5] Ball, James; Schneier, Bruce; Greenwald, Glenn (4 October 2013). 'NSA and GCHQ target Tor network that protects anonymity of web users'. The Guardian, 5 October 2013.

[6] Pagliery, Jose (17 May 2016).'Developer of anonymous Tor software dodges FBI, leaves US'. CNN, 17 May 2016.

[7] Lawrence, Dune (23 January 2014). "The Inside Story of Tor, the Best Internet Anonymity Tool the Government Ever Built Bloomberg Businessweek, 28 April 2014.

[8] Fowler, Geoffrey A. (17 December 2012). "Tor: An Anonymous, And Controversial, Way to Web-Surf". The Wall Street Journal, 30 August 2014.

[9] 'Tor Project: FAQ'. www.torproject.org.