SafeExit: Exit Node Protection for Tor

By

Jonathan Lutes

Submitted to the Department of Electrical Engineering and Computer Science and the
Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Master of Computer Science

Committee members

Bo Luo, Chairperson

Arvin Agah

Prasad Kulkarni

Date defended:

The Thesis Committee for Jonathan Lutes certifies
that this is the approved version of the following thesis :

SafeExit: Exit Node Protection for Tor

Bo Luo, Chairperson

Date approved: ________________________

Abstract

TOR is one of the most important networks for providing anonymity over the internet. However, in some cases its exit node operators open themselves up to various legal challenges, a fact which discourages participation in the network. In this paper, we propose a mechanism for allowing some users to be voluntarily verified by trusted third parties, providing a means by which an exit node can verify that they are not the true source of traffic. This is done by extending TOR's anonymity model to include another class of user, and using a web of trust mechanism to create chains of trust.

Contents

Chapter 1

Introduction

The TOR network is a system designed to anonymize internet traffic. Based on onion routing technology, it relies on a series of relays which establish ephemeral chains, referred to as "circuits", which forward encrypted traffic among each other. TOR users need not serve as relays, allowing people to use the network who are unable to operate full servers, for whatever reason. However, due to the complications that arise from forwarding anonymous traffic to the open internet, not all relays need to serve in this capacity, creating a third class of entities within TOR: Exit nodes.

Average-case performance within the network is highly dependent on the number of relays. Because the geographic distances between relays is typically high, TOR connections tend to have high latency compared to the ordinary internet. If a relay is congested, all connections passing through it will be degraded, harming network performance even more- quite possibly to unacceptable levels. One major point of concern is that the number of exit nodes is only a portion of the available relays, making them a point of congestion. Additionally, some attacks on TOR itself require compromised exit relays, so a greater number and diversity of relays may offer some additional defense to the network as a whole.

However, running an exit node poses a few problems for anyone who is inclined to do so. While not fundamentally more demanding than running an ordinary TOR relay, TOR is used by a wide variety of users desiring privacy- and, unfortunately, some are conducting various forms of illegal or otherwise harmful activity. The exit node is the one which forwards all traffic, desirable or not, to the internet, and so it may suffer some of the consequences of traffic it did not originate. One major issue is that the anonymity provided by TOR means that all traffic appears to originate at the exit node. In some locations, the exit node operators are not legally responsible for abuse of their relay, but in others, they may be.

In this paper, we propose a method for a new class of exit nodes, here termed "SafeExit Nodes" which are able to both provide evidence that they were not the originators of forwarded traffic, and still provide protections of the user's anonymity. This is aimed to allow a set of people who would like to operate a TOR relay, but are unable or unwilling to due to possible issues related to abuse of the service. It is designed to operate alongside ordinary TOR without interfering with it, while also allowing users to use the additional exit nodes. The users who make use of this service are ones whose models of anonymity are somewhat more permissive than the protection traditional TOR provides, but still require reasonable assurance of anonymity. This increase in flexibility may also improve performance of traditional TOR by shifting some traffic to the new nodes, creating a more diverse TOR network overall.

Chapter 2

Background and Preliminaries

2.1 Anonymous Routing

Anonymous internet communication requires that the identity of the communicating party be unlinked to their behavior. In a more specific sense, this means that the anonymous party should not reveal potentially identifying information to the servers they communicate with, including information within the protocol. Because all IP internet communication requires a valid return address, this typically requires that anonymity be achieved by forwarding traffic through a proxy. However, this by itself provides little anonymity- all trust is placed within the proxy to not reveal the user's identity. As such, numerous approaches have been developed to allow traffic to be anonymously routed to the open internet by forwarding encrypted traffic among several intermediaries.

TOR is an extremely popular anonymizing network, based on the concept of onion routing as described by Dingledine et al. (2004). Like all anonymity networks, its ultimate goal is to provide a means for users to hide their identity online. For the purposes of TOR and similar networks, Identity" equates to "IP address", because it provides a traceable link back to the rest of the user's identity. While many such possible links exist, the IP address is typically out of the user's control, because knowledge of it is required for internet routing. TOR does not protect the user from their own mistakes, such as using an anonymous connection to access an account associated with their user identity, but rather provides them the means to produce the anonymous connection in the first place, while leaving more obvious information leaks for the user to control.

TOR accomplishes this by creating a specially constructed proxy chain. TOR creates two broad classes of participants, users and routers. While any TOR user can potentially become a router, this is not mandatory. This is based on the idea that many users may be unwilling or unable to act as part of the network, but may still be allowed to participate in it. This allows an asymmetric network, where not all machines are equal participants.

This approach is beneficial, because it places minimal restrictions on who uses TOR. However, it places a greater burden on those machines which are greater participants within the network, because their aggregate performance of these nodes in the main determining factor for the performance of the network as a whole.

TOR achieves its anonymity via the concept of onion routing. This involves establishing a chain of proxies where no one participant has sufficient knowledge of the chain to link the identity of a user to a given connection. This is not to say that TOR attempts to hide the fact that it is being used- this is not one of its goals. Instead, it provides unlinkability between actions occurring at one end of a TOR tunnel and the identity of the user actually initiating those actions on the far side of the tunnel.

All TOR nodes who will forward traffic have published IPs and public keys, so that users can easily find them. When a given user wishes to work through TOR, they first choose a path of three nodes through the network. Then, they create a TOR tunnel through the three nodes. This is done by using onion encryption. That is, session keys with each node are established by encrypting the user's half of a key exchange with the public key of the last node, which is then bundled with forwarding instructions pointing to the last node and the key negotiation with the second node, which is encrypted again by the second node's key. This is in turn bundled with forwarding instructions to the second node, and the negotiation for the first node, then the entire bundle is encrypted with the first node's public key.

This creates a bundle of layered encryption, where each layer can only be removed by one node, and contains directions on where to send the next layer down. This has the interesting effect of creating a chain where each node knows only its direct predecessor and successor. A length of three nodes is used because this means that no pair of colluding nodes can unmask the user- the first two nodes know the user and exit node but not the contents of the communication, the last two nodes know the first node and the contents of the communication but not the user's identity. If the first and last nodes are colluding, the tunnel is subject to a well-known timing attack, which will be discussed later, but under other circumstances the two colluding nodes will not be aware that they are handling the same tunnel.

It is important to note that each node only uses PKI for setup operations; while it is necessary to authenticate the TOR nodes and ensure the confidentiality of the messages while in transit, the establishment of temporary keys for use with symmetric algorithms is crucial. The reasoning behind this is simple practicality- encryption is necessary for all TOR communication, but PKI operations are far too slow to be run over all data while maintaining reasonable performance. As such, faster symmetric cryptography is used once the tunnel is established.

Once a tunnel is established, TOR effectively acts as a TCP proxy from the view of client applications. Because all TOR traffic can only be returned by association with a tunnel ID, TOR is only able to support connection-oriented traffic. In practice, only the TCP protocol is supported. Applications which require UDP for connectionless communication are unable to use TOR, because the protocol cannot be supported without special network handling. While in theory, outbound-only UDP could be supported, such support would be generally impractical.

A critical part of TOR which has been previously touched on is the existence of two classes of forwarding node: standard routers and exits. Because not all users who are willing to participate in forwarding are willing or able to forward traffic to the open internet, any given node can choose whether or not they wish to forward to the open internet . This was done for a variety of reasons, ranging from the possibility of jurisdictions where forwarding traffic within TOR would be legal, but forwarding to the open internet would not, and the simple concern that the actions of TOR users will be traced to the exit node, and the operator blamed. These concerns have been shown to valid on more than one occasion, at least in certain jurisdictions.

One side-effect of this division is that there are many more regular nodes than there are exit nodes in the deployed TOR network. This leads to the possibility of congestion occurring at the exit nodes, as the entire traffic of the network is funneled into this subset of nodes. The crucial role these nodes play in the network gives them disproportionate importance to the network, both in terms of how they impact network performance and in terms of the impact on the network as a whole should they be attacked. The fact that one exit node must be part of every tunnel makes their aggregate performance that absolute maximum performance for the network. Because load balancing is not a primary design consideration, congestion can occur far before the theoretical maximum has been reached.

The fact that the position of an exit node in a tunnel is known, as well as their presence as a distinct subset of the network makes exit nodes a major target for attacks. In a very well known attack described by Murdoch & Danezis (2005), compromised exit nodes are used to search for traffic stream watermarked with timing, so that the actual activity is connected back to a particular user. The timing attack renders tunnels longer than three nodes undesirable, because they place greater strain on the network while offering no defense against such an attack. Exit nodes would also form an obvious target for a DDoS attack, if an attacker had the resources to disrupt a large number of exits.

This core functionality is not the only feature. Probably the most important additional feature is the ability for TOR users to host services entirely within the TOR network. This feature was designed to use as much of the existing TOR protocol and structure as was feasible, despite the differences required to offer this feature. The driving goal of allowing these "onion services" (recently renamed from "hidden services") was to address an underlying problem with TOR- it was originally client-oriented. That is, while it allowed a user to easily access any location on the internet without revealing their identity to any involved party, it was focused on clients wishing to initiate connections to existing web services rather than allowing anonymous hosting of services.

Hosting web services anonymously poses its own set of challenges. While standard TOR does not encounter serious addressing issues because the target services see traffic forwarded from an exit node as being just another connection, in order for someone to host a service which accepts incoming connections, it needs to have a unique identifier of some sort. TOR handles this by having each hidden service generate a public key keypair to serve as such an identifier as described in Dingledine et al. (2004).

The hidden service send signed requests to several nodes, referred to as "introduction points", and then registers the association with a DHT. However, to avoid possible legal complications in some jurisdictions, these introduction points do not simply forward traffic to the hidden service. While the hidden service does establish TOR tunnels to these nodes, these tunnels are only used in a negotiation process with user in which another node is chosen to serve as a "rendezvous point". Both the user and the the hidden service establish tunnels to this node, and use it to forward to forward traffic between the two tunnels. While this process is somewhat complicated, it provides a reliable way for users and hidden services to connect while protecting the identity of both, as well as preventing complications for the introduction points.

Additionally, in order to prevent long-term tunnels providing a means to trace a user, TOR automatically forms new circuits every few minutes. Ideally, this process is transparent to the user, and it consists of TOR negotiating a new circuit and routing new traffic through it.

Changes have also been made to TOR to handle situations which were not foreseen initially. Probably the largest is the addition of guard nodes Øverlier & Syverson (2006). One danger of the aforementioned timing attack is that in the initial design of TOR, a user who used TOR for an extended period of time would be very likely to eventually choose a hostile first router which was attempting a timing attack. While there is is only so much that can be done to resist such an attack, one decision that was made was to have the TOR software select a small subset of "entry guards", which are the only initial routers that it will use during that run. The idea behind this is that if a new first hop is chosen randomly every time a new circuit is generated, the probability of a hostile node being chosen over that session of TOR rises with time. Meanwhile, if a set of potential first routers is chosen initially and not altered, the probability of selecting a hostile first router over the life of the tunnel is limited by the probability of choosing a hostile first router in the initial set of guards.

2.2 Preliminaries

The main benefit of SafeExit is granularity, both in the strength of anonymity offered and the strength of safety provided to the exit node. The standard TOR design offers full anonymity, and only simple safeguards for exit nodes. While this means that any exit node contributes its resources to all classes of user, it does not offer the ability for an exit node to offer any other service, or for a user to specify a minimum level of service requirement below the maximum level of service offered. However, this cannot be done at the expense of anonymity such as what a discreet partitioning scheme would do.

For the purposes of this approach, IP is considered sufficient as identification. This comes with several immediate issues. The first of which is that any proxy, including TOR, can be used to create a false identity in this system. However, in this model, such an approach to circumventing the identity system is not considered in depth. However, it should be noted that the trusted entities (TEs) within the network have the ability to blacklist any IP that it distrusts. Said blacklists are dependent on the policy of a given entity, but they should typically include TOR itself, to prevent looping repeatedly through TOR and draining network resources. However, like blocklists at ordinary exits, this should be something which can be modified at the discretion of the operator.

Attempting to detect non-TOR intermediate proxies is a different problem than the one than is being addressed with this design. While it could be prevented by requiring users to possesses semi-permanent PKI keypair, this needlessly exposes the user to risk of unmasking. While Safe Exit does use PKI key for identity verification, they are also only used as ephemeral session keys which are not directly tied to identity.

Also of note is that, because the Safe Exit trust network is not carrying data streams for the user, somewhat higher latency is acceptable. While very high delays in the setup procedure are undesirable, they are likely to be less noticeable to a user than, for example, increased delays in opening a web page. This allows an increase in allowable complexity, especially due to the slow operations being mostly pushed to setup.

2.3 Web of Trust

The web of trust model has existed in one form or another for some time. In essence, it relies on the idea that trust can be propagated through a network of nodes. The basic idea is that if A trusts B, and B trusts C, there exists a path by which A is able to indirectly trust to C to some level. On one extreme, trust may be expressed as a binary relationship, wherein there are only two states of trust: trust and non-trust. Variations on this system may express trust as a non-binary relationship, with the concept of partial trust playing a critical role as described in Guha et al. (2004). Calculating this partial trust is the matter of a good deal of research.

In a very simple binary trust system, a node typically distrusts all nodes by default, and builds a list of nodes it trusts- typical whitelist behavior. In order to make this trust meaningful over an insecure medium such as the internet, nodes need to provide public keys for identity. Key distribution is often a complication in such systems, and out-of-band transmission is often used to forge direct links.

However, once a link has been formed, trust propagation can be performed by having a trusted node sign the public key of another node it trusts. Such a measure shows that it is willing to vouch for the accuracy of the key. This principle is the core of the web of trust- all of the nodes involved form a graph where the links represent a trust relationship, and knowledge of the other links can be communicated to neighboring nodes. This means that any node in a connected portion of the graph is able to communicate the public key of any other node in that portion of the graph, with the key being transfered over links where the nodes at each hop can verify the integrity of the payload.

Such systems are often implemented with the main aim of providing good scalability when propagating partial degrees of trust. Many schemes for this have been proposed (Blaze et al.

(1996),Guha et al. (2004), Haenni & Jonczy (2007)), but most boil down to the concept that a node which is highly trusted by a highly trusted neighbors should be trusted less than the neighbor, but still quite highly. Meanwhile a node which is highly trusted by a neighboring node which is not particularity trusted will still be trusted less than the immediate neighbor, which is still not a great deal of trust. The various schemes mainly differ in how the specifics of the graph are handled, with each providing different levels of performance and propagation behaviors.

2.4 Definitions

TE: Trusted Entity user node: the machine using TOR; in this case, one using SafeExit exit node: in this case, an exit node running SafeExit.

Chapter 3

Basic Model

3.1 Overview

In the basic model, all trust is placed on a Trusted Entity, or TE. This node represents a server run by a third party, presumably one trusted by both the SafeExit node and the user. For this simple model, trust is defined is a simple state of trusted or untrusted; there is no web of trust in this simplified version. This is not intended to be used for real world privacy; since all the trust is placed in a single node, this simplistic model is vulnerable to numerous attacks. Rather, it demonstrates the core of the SafeExit protocol without the details of the more elaborate versions complicating the model.

3.2 Trust Assumptions

This model is highly dependent upon several assumptions of trust. The most basic of which is that some of the trusted entities (TEs) in the network are, indeed, trustworthy. In both the base case and the full model, if just one TE is actually trustworthy, the user's anonymity should remain secure; if two or more TEs are trustworthy, a well-behaved user should be as anonymous as they would be under regular TOR. Related closely to this is the assumption of a well-behaved user; a user who puts an exit node at risk by abusing its use of the network is, by design, placed at risk.

However, because the user is able to exert control over the TE, (in the basic model), or the TEs nearest itself (in the full model), the user is able to limit this risk to themselves to a level they consider acceptable. Finally, it is assumed that the pool of users is large enough to provide enough traffic for both regular TOR and SafeExit users to blend into.

Defining what constitutes "trustworthy" is left up to the user, although it should be in line with the level of anonymity they desire. For example, an anonymous blogger would likely want to choose a TE in a jurisdiction with legal protections for free speech, operated by an entity with a strong ideological opposition to censorship, and historically good data security practices. This entity can then provide the exit node operator with a third-party agreement that the connection was not sourced at the exit node, but is also highly unlikely to disclose the blogger's identity. Where the advanced model differs from this basic model the most is in terms of user safety, since allowing one node to unmask a user, no matter how much said user trusts the node, provides no major anonymity advantage for the TOR user compared to a single-hop proxy server.

3.3 Protocol

The basic model involves a regular user node, a TOR tunnel with a SafeExit node as the exit, and a single TE. No web of trust is implemented in this case.

Initial state: A regular TOR tunnel establishment has been performed, so an encrypted connection exists between the user and the exit node. The TE is not currently involved.

The protocol can be split into three phases: Certificate Acquisition, Communication, and Revocation.- They are defined as follows:

3.3.1 Certificate Acquisition

1. (optional) Exit node $\rightarrow$ User (via TOR tunnel): LIST = (m, (T)) where m is the message LIST, T is a TE identifier, and $[T]$ represents a null-terminated list of TE identifiers.

2. User $\rightarrow$ TE (via an SSL tunnel): $CertRequest$ = $encrypt_{TEpublic}(m,\ n_1,\ ID_{Exit},\ )$, where

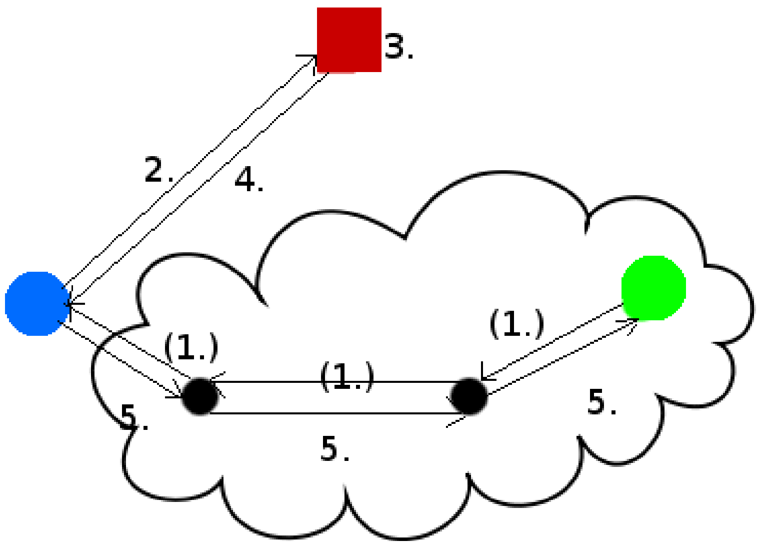

Figure 3.1: Illustration of the establishment procedure of the basic model

TEpublic is the TE's public key, m is a message indicating that a certificate is being requested, n_1 is a nonce, and ID_{Exit} is the exit node identifier

3. TE: If the user's IP is not blacklisted, produce certificate = $sign_{TEPrivate}($ $encrypt_{ExitPublic}(SID,$ Timestamp, $SessionKey_{Exit})$), where TEPrivate is the TE's private key, ExitPublic is the exit node's public key, SID is a stream ID, Timestamp is a time stamp, and $SessionKey_{Exit}$ is half of a public key pair.

4. TE$\rightarrow$user (vis SSL tunnel): $sign_{TEPrivate}(m$, certificate, $SessionKey_{User}$) where TEPrivate is the TE private key, m is a message indicating that this message contains a certificate, certificate is the previously generated certificate, and $SessionKey_{User}$ is the user half of the same session key as is found in the certificate.

5. user $\rightarrow$ Exit Node: certificate

The list of TEs trusted by a particular exit node is not considered secret in this version of the model. This list can either be assumed to be published, or, alternately, provided by the exit node on request. Regardless of how it is acquired, the list of TEs is acquired, the user's node then picks one from this list, and the suer then requests a certificated from the TE. The assumption, for the purposes of this model, is that IP is sufficient to serve as identity for a SafeExit use; as such, the user need only contact the TE, establish an SSL tunnel, and then proceed to provide it with the identifier of the exit node to which the certificate will be addressed, and a nonce (in order to make the request unique). This information allows the TE to produce the certificate.

The certificate itself is ultimately destined for the exit node, and as such, the entire certificate in encrypted with the exit node's public key to prevent interception or tampering. The certificate first includes a session ID (SID), which is simply a random number used to identify the session. This is followed by a timestamp, used to defend against a replay attack by allowing the exit node to reject expired certificates. Finally, half of a PKI keypair is included, acting as half of a session key for the user and TE; this must be a PKI key in order to prevent the exit node from fabricating messages from the user in its logs. The entire bundle is signed by the TE, to prevent forgery.

The entire certificate bundle is then sent to the user, as well as the user's half of the session key. This is signed by the TE. The user, upon receiving this, is in charge of forwarding the certificate to the exit node via the TOR tunnel. Upon receiving the certificate and verifying its signature, the exit node is ready to begin operating normally.

Note that the TE has the ability to blacklist any user that the owner does not wish to verify; this list should, for the typical node, include the list of TOR exit nodes to prevent a trivial circumvention of the SafeExit by running through conventional TOR and back into SafeExit. This particular behavior would most likely be an attack on TOR intended to use excessive TOR bandwidth- the combined delay of a regular TOR tunnel plus a SafeExit tunnel would exceed that of simply using a regular TOR connection, so there would be no practical reason to do so.

3.3.2 Communication

1. user $\rightarrow$ exit node: $MESSAGE = sign_{UserSessionKey}(m)$,$sign_{UserSessionKey}$(hash($m$), $length_m$, where m is a standard TOR message and $length_m$ is the length of m

The TE should also log the certificate it has been provided, and at least the signed hash and length of each message. The full signed message should at least be logged temporarily, since it provides non-refutable proof that the exit node did not send the message.

All communications to the exit node should be signed by the user's session key. This provides firm evidence to the exit node that the user which received the certificate from the TE and provides a way to link all user communications back to the user. At the same time, it provides the user protection from the exit node- the exit node is unable to fabricate user communications. The asymmetrical nature of the PKI keypair is necessary for this; in a symmetric pair, the exit node could fabricate user messages arbitrarily.

The key weaknesses of this are several. First, the TE knows both halves of the PKI key, and so is able to fabricate messages. This is a flaw with the simplified nature of this model, but the TE is also assumed to be completely trustworthy in this case. Another potential weakness is the reliance on the exit node to keep proper logs, but no or improper logs merely reduce the exit node's behavior to that of an ordinary TOR node; while this defeats SafeExit, the exit node has nothing to gain by doing so. The other weakness is the use of a PKI key for user$\rightarrow$exit node signing, but the anonymity implications are minimized by making the key a per-tunnel session key. More typical symmetric session keys will not work, because they would allow a malicious exit node to frame a user for the exit node's activity; the PKI session key removes the ability of a malicious exit node to perform any such attacks.

It should be noted that in this model, the exit node's list of trusted nodes is not private. The model could be simplified by publishing the list alongside the exit node's public key (assuming that the list was somewhat static), saving bandwidth and a round trip through the TOR tunnel. However, in the more advanced model presented later in this paper, the exit node's list of trusted TEs is not considered public knowledge. The list is provided by the TE in this case to decrease the necessary modification to the base TOR network (if an exit node-submitted list of TEs were to be published, the TOR directory servers would have to store considerably more data than just the keys of the SafeExit nodes), and to increase similarity to the more elaborate approach, with the added benefit of allowing rapid alteration of the trusted TE list to be possible without repeated signing and uploading of the list.

The user then chooses a TE from the list which it trusts, if one exists. If it does not trust any EA, it sends a message to the exit node terminating the tunnel. If it does trust at least one TE, it should send the TE the message $AUTHREQ$ = *encrypt*(TE public key, (m, e)) , where m is a message containing $AUTHREQ$, e is an exit node identifier, and TE public key is the TE's public key. The TE is able to identify the IP of the user node from the source of the connection, and the exit identifier allows it to deliver verification to the exit node.

The message$AUTHREQ$ is encrypted with the TE's private key to prevent an eavesdropper from being able to see the user's chosen exit node. The TE should then respond to the user node with USERVERIFY= [m, k_1, $certificate$] where $certificate$= *hash*(TE private key, *encrypt*(TE private key, *encrypt*(exit node public key, [e, t, T, exp])), m is a message containing CERT, e is an exit node identifier, t is a TE identifier, T is an integer timestamp, exp is an expiration timestamp, and k_1 is the user side of a PKI keypair.

This serves to provide the user with a certificate proving that they have had their identity verified with the TE, in the form of half of a PKI keypair. In addition, it provides them with a certificate, signed by the TE and encrypted with the exit node's public key, which is to be sent to the exit node. This certificate contains some verification information, and the other half of the PKI certificate. When decrypted by the exit node, this will provide verification that the TE has, in fact, received the verification. No message need be sent directly to the TE, as the TE's side of the communication if effectively bundled into the circuit.

The TE should also send the exit node the message EXITVERIFY = [m, *sign*(TE private key, *encrypt*(exit node public key, k_2))] , where k_2 is the exit node side of a PKI keypair.

This provides a way to provide the exit node with the other half of the PKI keypair, so the exit node is able to directly verify the validity of the connection to the client.

The TE should also log some information about the connection, so that it is able to reliably link the participants if needed, even after the active connection has terminated. Specifically, it should log CERTIFICATELOG = [*hash*(*encrypt*(TE private key, *encrypt*(exit node public key, [e, t, T]))), k_1, k_2, u, *eid*] , where u is the user's IP, *eid* is the identifier of the exit node, and the other values are as defined above.

This serves to create a record of the previous verification step. The certificate itself is stored as transmitted, and the PKI keypair is also preserved, along with the IP of the node requesting the service. This allows the IP of the user node to be retrieved if needed. The exit ID is stored so that the exit node half of the connection is also known.

The message $CONNECTIONCERTIFICATE$= (*hash* (*sign*(k_1 from previous message, (m, *hash*(TE private key, *encrypt*(TE private key, *encrypt*(exit node public key, (e, t, T, exp)))))))) , m is a message containing CONCERT, e is an exit node identifier, t is a TE identifier, T is an integer timestamp, and exp is an expiration timestamp.

This takes the certificate previously created by the TE, signs it with the client's half of the session key, and sends it to the exit node, in order to provide proof that not only is the client in possession of a certificate from the TE, but also the key for this connection. When the exit node receives this, it can decrypt the certificate to gain access to its half of the PKI key. However, it must first verify the signature of the data within the certificate, to ensure that it came from the chosen TE. If this verification completes, it can then use the bundled PKI key to verify the user node's signature on the entire message. This complex verification is necessary, since the communication from the user node to the exit node contains the TE's certificate, the signature verifying that the user is in possession of its half of the TE's PKI key, and the (encrypted) exit node half of the PKI key.

The Exit node, on receiving certificate, calculates *sign*=(*hash*(*encrypt*(TE private key, (m, *encrypt*(exit node public key, (e, t, T, exp)),*hash*))))) , and determines if *sign* is equal to *decrypt*(k_2, *hash*), where k_2 is the same KPI key previously provided to the exit node by the TE. If so, it calculates $sign_2$=$encrypt$(exit node public key, (e, t, T, exp)), $hash_2$, and determines if $sign_2$ is equal to $decrypt$(TE public key, $hash$). If so, it calculates $decryptedcert$ = ($decrypt$(exit node private key, (m, $encrypt$(exit node public key, (e, t, T, exp))))).

This step is simply the act of verifying the signature on the certificate from both the client and the TE. If the certificate is invalid, the exit node should terminate the tunnel, as either data has been corrupted, or the connection is being tampered with.

3.4 Issues

This setup has several issues. The most crucial of which by far is the ability for the TE to easily unmask a user, due to the fact that it knowns both the real identity of the user and the user's exit node. This is effectively the same security level of a two-hop TOR tunnel (effectively constituting the exit node and the TE), which compromises network integrity regardless of the reliability of the TE; an attacker who can compromise the TE can determine that the TE is connected to both the user and the exit node from the source and destination of communications. An untrustworthy TE is also able to compromise anonymity to a significant degree. The TE is able to at least indicate with some certainty that there is a reasonably high chance that traffic sourced from the exit node's IP is actually coming form the user, which greatly reduces the anonymity provided by TOR.

The attacks possible if an exit node were to only allow connections where the user accepted colluding TEs is somewhat mitigated by the user having control over the final choice, since an exit node only advertising hostile TEs is likely to have most connection attempts fail when a user refuses all potential TEs due to not trusting any of them. However, if an attacker is able to make a compromised TE look trustworthy, this attack becomes more dangerous because a user who trusts even one compromised TE may be directed to use it by the malicious node. In a likely variation of the attack, if an attacker were to compromise an otherwise trustworthy TE, they could set up bad exit nodes which only directed users to use the compromised TEs.

Chapter 4

Security Analysis of Basic Model

This basic model only demonstrates the concept of SafeExit in its simplest possible form; it is not intended to be deployed in the real world. The simplified form of the model does not provide sufficient protection against many attacks, and as such needs to be modified in order to be useful. However, the flaws of this design inform what needs to be modified in order to make a better system.

The first obvious flaw is the amount of trust place din the TE; the TE, acting on its own, can reveal which user is using which exit node. Quite simply, the user has asked for a certificate verifying their desire to use that exit node, and knows the IP of the user who placed the request. While the TE does not know what the user is doing, simply linking them to an exit node is a massive breach in anonymity.

The user knows as much as in regular TOR. Because all TOR routers and exit nodes are already published, their IPs and public keys are already available. The only real increase in information on the TOR infrastructure is that SafeExit users and exits must choose a set of TEs to trust, which may be the basis for profiling attacks not present in standard TOR.

The exit node functions largely as in normal TOR, but now logs the certificate provided by the TE. This certificate is necessary to revoke anonymity. If it were stolen by a third party, they would gain the critical information needed in revocation; therefore, a threat to the secrecy of the logs is a direct attack on the anonymity of the entire system. If the exit node's key was also compromised, the attacker could effectively impersonate the node, and attempt to stage revocations on their behalf. However, a malicious exit node could already conduct malicious revocation attempts.

A malicious exit node could also repeatedly reject TEs from a user, with the intent to acquire the user's whole list of TEs. The amount of data released in a single instance of this attack could be reduced by limiting the number of TEs a single exit node can reject before the attempt fails entirely. However, even a small set of trusted nodes could be used to partially identify a user, depending on how unusual their set of trusted nodes is. Given that the set of TEs trusted by a user my be highly unique, a fingerprint formed from this list could severely compromise user anonymity.

Between the user and the exit node, a standard TOR tunnel secures all communications. This tunnel is neither more nor less secure than basic TOR, and inherits any known flaws in conventional TOR. Because all SafeExit messages are merely conveyed via the TOR tunnel, they do not alter the tunnel's behavior beyond a small shift in the patterns of data sent during connection establishment.

There is no direct communication between the TE and exit node, so no eavesdropping is possible.

An eavesdropper between the user and the TE can learn that the TE is part of the user's list of trusted TEs. However, because the connection is encrypted using PKI, an attacker cannot impersonate the TE or intercept plaintext communications between the user and TE.

Collusion between a user and an exit node mainly allows a means for a user to potentially forward traffic through that node using an invalid certificate. While the exit node cannot forge a valid certificate, it could potentially create a fake, supposedly corrupted certificate, and associate the user's traffic with it. However, a certificate being corrupted would be highly suspicious, and would create suspicion of collusion if it were to coincide with abuse of the exit node. This, in turn, would potentially remove the protection that SafeExit would provide to the exit node, making willing collusion unlikely.

Collusion between a user and a TE would result in fairly limited damage. In order for the exit node to approve the certificate, it would have to be valid, so the most a TE could do when colluding with a user is refuse to store correct records. The greatest concern here would be the alternative approach of storing falsified records of the connection, and blaming an innocent user for the activity. While this falls into the scope of attacks which a malicious TE could perform, it deserves special mention as a particularly damaging attack.

The only other meaningful collusion in this version of the model is for a malicious exit node and malicious TE to collude. This is, in, short, disastrous to the basic model, since it links the entity that knows what the user is doing (the exit node) with the entity that knows who the user is (the TE). This catastrophic compromise of anonymity requires almost no resources, beyond the existence of the colluding nodes. This attack is the largest reason why the advanced model is necessary.

Chapter 5

Improved Approach

The improved SafeExit approach makes use of the Web of Trust to allow chains of TEs to be used in the verification step. A chain of TEs just two long should provide the same degree of separation that TOR already provides in a three-hop tunnel (the knowledge of the user's real identity is separated from their traffic by two TEs and the exit node). If this is enforced as a minimum tunnel length, it should offer the same kind of anonymity safeguards seen in TOR itself, if the trust assumptions of this model hold true.

In order to prevent possible exploitation of the knowledge of the trust path, the nodes should be minimally aware of the chosen path. However, in order for both the user and the exit node both need to be able to trust the chosen nodes to a certain degree, and so they both must take part in the creation of the chain. However, as previously discussed, a trust chain of just two nodes (the exit node still constitutes a third node) is sufficient to provide the same level of separation as TOR itself, which simplifies negotiations greatly. One solution implied here is that the two chosen TEs should use the user and exit node (and the associated TOR tunnel) as intermediaries when establishing identity. One such approach is to have the nodes conduct a Diffie-Helman handshake Diffie & Hellman (1976) to establish a shared session key. This is, of course, vulnerable to a man-in-the-middle attack, since the identities of the the TEs have not yet been verified. In this case, one of the TEs should provide a certificate containing its signed ID, nonce, and a timestamp to the other, which should then be reciprocated. If the certificate contains an incorrect nonce or incorrect timestamp, the process should halt.

One issue here is whether the user's TE should send the information first, or whether the exit node's TE should. This is an issue because until one party reveals their identity, verification cannot proceed. This opens the party which transmits first to a small attack, wherein a party impersonating the other TE determines the identity of the TE which sent first. This is not an attack on the TEs, so much as a possible attack on either the user or the exit node, wherein an attempt is made to get them to reveal their list of trusted TEs without making a connection first. This attack is already possible on exit nodes through simple persistence (if an attacker were to use the same exit node repeatedly, they would likely see all TEs the exit node trusts), but not necessarily on users (a bad exit node might be able to determine this from a user, but it cannot force the user to reveal this information unless the user chooses to repeatedly connect to the same exit node). However, a solution is discussed later which bypasses this entirely be requiring neither party to reveal their chosen trusted TE.

This does not, however, address the issues of the web of trust itself. The general construction of a web of trust requires that nodes are able to sign their trust of another node's key, and are able to present this trust to other nodes. If trust is considered to be binary over the scope of this network, that is, any given node either trusts a node to forward connections from it or does not trust it, a relatively simple approach may be used. In this approach, each node uses a "reachability list" in order to propagate trust. In essence, While a node should know which other nodes it trusts, and should manually choose to establish this relationship, it can distribute a list of nodes and keys it knows can be reached through itself to the node it just introduced to the network. Each entry in the list should be individually signed before inclusion in the list and the old signatures (from other nodes in the chain) should be included in this signature. An exception should be made if the node previously signed the chain, in which case the old chain should be used in the reachability list to avoid loops. In cases where more than one chain to the same exit node is given, both should be preserved, but only one included in the reachability list sent to other nodes. Which one is sent should be periodically rotated, so that other paths are advertized.

The structure of a keychains should be fairly simple, [*id*, key_node, $signature_1$ ... $signature_n$], where *id* is the IP of the node the chain is authenticating, key_node is the key of the node, and the $signature_n$ fields each contain a signature in the chain of all previous fields. The key chain would be effectively worthless alone, but each one of these certificates should be transmitted alongside a large batch of other signatures, which will provide the public keys of all nodes in the chain. Verifying such a signature is a time-consuming task, but will likely be performed rarely in an established node; TEs are unlikely to appear or change trust relationships with enough frequency to create computational burden due to re-verifications of trust chains. Initial establishment of a TE in the network will require the verification of each path, but this is a one-time series of operations.

This implementation of a web of trust allows a node to choose who it trusts, while providing others a more useful piece of information for general synchronization- the list of know reachable nodes, with full key chain signatures. Each the entire list need not be signed before being sent, because each keychain has already been signed. On receiving this list, a should transmit updated information if any new nodes are made available, also potentially including new paths. At worst, the transmission complexity for this list is order $log(mnl)$, where n is the number of nodes in the connected portion of the system, and m is the number of immediate neighbors, and l is the average path length for all paths being transmitted. Given that m is likely relatively small, this is unlikely to pose a great burden on a server. Storage complexity is higher, but is bounded because the only duplicate path entries that need to be stored are ones where a path decisions is presented to the node itself, that is, when two neighbors both advertize routes to the same destination. If a node receives, at different times, two paths to the same exit node with the next hop passing through the same neighbor, storing the knowledge of the new route serves no purpose- a route to all nodes is known, and both paths have the same next hop, so the details of the path are irrelevant to the node.

Building the web of trust is not a new idea, but is a necessary step to creating a self-regulated network of trust; however, it does not address how the web of trust is used. Creating a SafeExit tunnel begins, as in the basic case, with the creation of a TOR tunnel between the user and the exit, and an announcement from the user to a TE that they wish to use a SafeExit connection. However, at this point, a complication arises- in order to choose TEs, it needs to be confirmed that the TEs are in a connected part of the trust network. However, because neither party is aware of the other's TE in advance, it is not possible to simply send the TE identifiers without revealing which TEs are chosen; optimally, neither the user nor the exit node should know the other's TE.

This necessitates a more complex establishment procedure. First, it should be assumed that any node in a connected portion of the web of trust is known to all others, and is verifiable via a chain of certificates. As such, for the purposes of reachability, any node within the connected portion of the graph should be equally capable of verifying reachability, and to reach one node in the connected portion of the graph implies that all nodes in that connected portion are reachable. This means that a third party, not directly trusted by either the user or the exit node, could be used as a sort of connectivity tester, and third-party arbitrator.

5.1 Protocol

This random connected node, referred to as a connectivity establisher, or CE, is able to be revealed to the other party while only leaking the information that the chosen TE is one of *n* TEs in the same connected portion of the graph, which is the highest level of anonymity achievable when using the connected subgraph. For the purposes of tunnel creation, it should be a node which is also able to serve is a neutral midpoint to the chain, allowing the endpoints to communicate without revealing their chosen closest TEs. While the CE and the eventual tunnel midpoint do not technically need to be the same node, re-using is considerably simplifies the model, and any given central midpoint will meet the criteria of being in the same graph as the endpoints. This is important to avoid leaking information about the precise set of TEs which an endpoint trusts; if this data is leaked, it may be possible to exploit it to an attacker's advantage; the number of users using a particular set of TEs is almost certainly smaller than the number of nodes who trust any of the TEs in the connected portion of the graph. The CE is probably best chosen by the user's TE, because the user is able to negotiate with the CE before the TOR tunnel is even established. While the benefit of doing this is likely small, it does offer the chance for a latency reduction should the exit node to CE tunnel take less time to negotiate than the user to CE tunnel. This negotiation is fairy simple; the user need only send a request get_{nodes} = GETNODES to one of its trusted TEs, which just a signal requesting a list of connected nodes of which the TE is aware. The TE will respond with its list of known nodes (which will be identical for all TEs in the connected graph). Basic information for each node should be publicly available, including a node identifier to IP mapping and the public keys of all nodes. The user can then select a CE from the list.

A more advanced model does not involve the user in TE selection, but rather has whichever TE the user selects as a first hop randomly choose the CE from the list of known contacts. This removes the need for the user to receive a list of potential contacts from the TE. However, the user should be able to reject any CEs that are on a blacklist of nodes it explicitly distrusts. This approach fully leverages the web of trust, making intermediary nodes of secondary importance. The importance of the CE remains the same- it serves as a sort of neutral meeting point for both participants, while ensuring a reasonable path length. In this model, there is a bare minimum of three nodes involved- the user's first TE, the exit node's first TE, and the CE; This is greater separation between the user and the open internet than base TOR. While this is not strictly necessary to keep the same separation of TOR itself, it is required to isolate the knowledge of which nodes the user and exit node explicitly trust.

The protocol for this more advanced model has three major parts: CE Selection, tunnel-forming, and use. Revocation may be considered a fourth part, but is discussed separately.

5.1.1 CE Selection

1. User: Select TE from list of trusted TEs
2. User $\rightarrow$ TE: m = GetCE
3. TE: Select any known connected node for use as CE

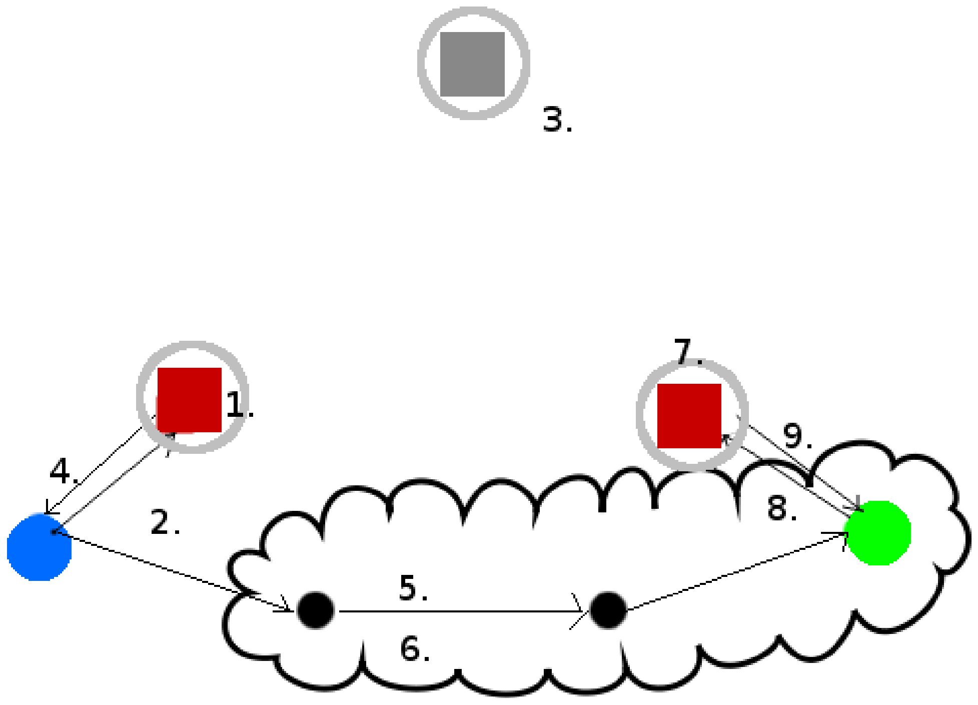

Figure 5.1: Illustration of the establishment procedure of the improved model

4. TE→ user: m = CEID

5.
   - User: Repeat CE acquisition if provided CE blacklisted
   - Otherwise: User: establish TOR tunnel to SafeExit node

6. User→ exit: CEID

7.
   - if CE blacklisted by exit: exit→ user: BadCE
   - Otherwise: Select TE from list of trusted TEs

8. Exit→ TE: ConnectToCE, CEID

9.
   - If CEID not connected to TEExit, TE→ exit: NoConnection; Exit: Select new TE, go back to 8
   - Otherwise: Exit→ user: GoodCE

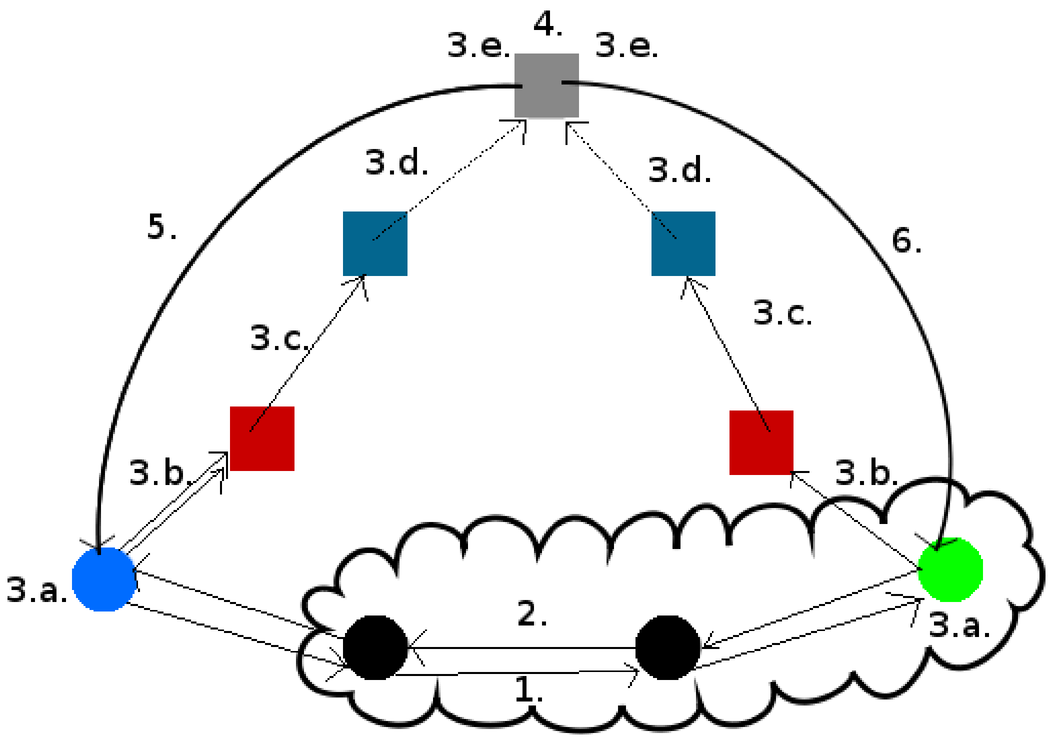

Figure 5.2: Illustration of the tunnel building process for the improved model

5.1.2 Tunnel-forming

1. SEExit$\rightarrow$ user (can be part of last message of previous phase): sign(SEExit$_{public}$, conID)
2. User: Create session PKI pair; user $\rightarrow$ SEExit: Skey$_{public}$, sign(SKey$_{private}$, conID
3. In parallel:

   Exit Side:

   (a) SEExit: create m=encrypt$_{CEPublic}$ (encrypt$_{SKey_{Private}}$(conID), Skey$_{Public}$ timestamp)

   (b) SEExit $\rightarrow$ NextHop: m, SID_0; both log SID_0, where nexthop is the TE used in the CE connectivity test

   (c) Nexthop$_n$ $\rightarrow$ nexthop$_{n+1}$: m,SID_{n+1}; log SID_n, SID_{n+1}

   (d) ...

   (e) When CE receives message: CE: Decrypt m

User Side:

(a) User: create m = $encrypt_{CEpublic}(conID, timestamp)$

(b) User $\rightarrow$ TE$_1$: m, SID_0; both log SID_0

(c) TE$_n$ $\rightarrow$ TE$_{n+1}$: m, SID_{n+1}; log SID_n, SID_{n+1}

(d) ...

(e) When CE receives message: Decrypt m

4. CE: decrypt user's copy of ConID

   - if user's ID does not match exit's ID: fail

5. CE$\rightarrow$ user (via intermediary nodes): sign$_{CE_{Private}}$(OK, nonce, timestamp)

6. CE$\rightarrow$SEExit (via intermediary nodes): sign$_{CE_{Private}}$(OK, nonce, timestamp)

When this phase is complete, the CE can generate a certificate consisting of $certificate$ = (ID, *nonce*, $timestamp$, *expiration*, *signature*), where *nonce* is a nonce, $timestamp$ is the current timestamp, *expiration* is the time at which the validation expires, and *signature* is the node's signature on the rest of the message. While simple, this certificate is only asserting the identity of the CE, to be used for reachability. The timestamp and nonce are to render the certificates immune to replay attacks, while the expiration time is used to limit the length of time in which the certificate is valid. This certificate is sent back through the SafeExit tunnel and back to the user node. This will then be forwarded through the TOR tunnel to the exit node. It should be noted that the lack of explicit encryption on the certificate is due to all traffic being encrypted through these tunnels. While roundabout, all this does is ensure isolation of any information about the user's route to the CE, while also providing the required verification. Once received by the exit node, the node can verify the signature, and if it is valid, forward it to the exit node's choice of first TE, which can then determine if it recognizes the ID as a connected TE.

The user's TE should then return the result of the connection attempt back to the user. If it is not reachable , the user may repeat the procedure with other trusted TEs until it finds one which can reach the CE. If this fails, the user may either terminate the connection, or request that the exit node chooses a new TE and CE. If it is successful, connectivity to the CE has indirectly shown that connectivity exists to the exit node's TE, making use of the graph structure.

Once connectivity has been confirmed, actual verification chain is to be established. This requires a potentially time-consuming verification chain creation process, but the final result is that each hop of the verification procedure is handled by a somewhat unpredictable series of nodes, where all hops are through manually-created trust links. This ensures that the validity of each hop has been manually validated (since the trust relationship requires manual validation), but by making the path unpredictable, it renders it difficult to attack the TE network with limited resources. Because the only way to be part of a path is to be considered trustworthy by existing nodes, the ability to create a large number of fake TEs (a Sybil attack) does not correspond with the ability to integrate these nodes into the trust graph. In essence, this is quite similar to the onion routing used by TOR itself, although the number of possible path configurations is smaller.

The main issue is the question of how the authentication proceeded through this chain. Both the user and the exit node have a stake in ensuring that the verification id correct, because the user needs to be sure that the chain is secure enough that their identity will not be revealed, but the exit node needs to be sure that it is possible for the chain to show that activity from the user was not initiated by the exit node. While any TE in the reachable network should be trustworthy, real-world trust is not necessarily binary; while the web of trust model employed does not model it in order to increase possibly routes, real trust is likely to degrade with each hop away from the TEs which the participants actually trusted. As such, only a cooperative generation scheme using the two chosen TEs provides the assurances required by both the user and the exit node.

This leads to the solution of using the CE as an independent point in the chain, which both parties in turn make contact with. By routing the chain through this one node, both parties are able to pick part of the tunnel, thus ensuring that neither one is able to gain control over path selection.

As a result, both end TEs need to agree on the full path before it is used. The created route must then be agreed upon by both endpoints before it is used, so both nodes have a chance to reject any route that fails to meet their criteria. Neither end point, however, requires total knowledge of the route.

One way to do this if for one of the endpoints to request a route to the other, sending a route request req = (encrypt(id_{tunnel}, key_{dest}), (list of hops), encrypt(id_{dest}, key_{next})), where id_{tunnel} is an identifier for the far end to use to identify which tunnel request has gone through, id_{dest} is the identifier of the desired endpoint, key_{dest} is the public key of the destination, and key_{next} is the key for the next hop. The list of hops initially starts empty, but each node should add to it,appending its own ID and a signed hash of the entire previous list, including the ID, such that the list is = encrypt($list_{old}, ID_{current}, key_{local}$)This same structure is used throughout the path establishment phase, with only the hop key changing. This creates an onion-encrypted list of nodes that have been chosen, while allowing each hop to see the destination in turn, so they are able to route towards it.

Both end-point TEs should build a TOR-style onion routing path to the CE. This needs to be slightly modified, however, since each hop also needs to have circuit IDs for both the next and previous nodes. This can be done having each hop produce an ID for the next hop, and send it (encrypted with the next hops' key) along with the rest of the route establishment request. This ID needs to be logged by the TE; the core of a TE-level log for a midpoint is little more than an ID pair, the IDs of the next and previous hops in the path, and an initiation time. The initiation time is not yet know, however, since the chain is not active, and will be filled in later.

Once the path through the TEs is established, the CE should send each half of the path a message indicating route formation success. The CE should generate a pair of certificates, one for the user and one for the exit node, in order to prove that the tunnel has successfully been created. The message should be encrypted between hops, and on receiving the success message. Each TE should fill in the activation timestamp at this point. Setting the activation timestamp when the entire tunnel is complete rather than when a single path is initialized allows asymmetrical initiation of the paths.

The user node and exit node should both create nonces, and encrypt them with the temporary shared key from the initial request. These encrypted nonces should then be placed in onion routed packets to the CE. When the CE receives each packet, it should forward it down the next half of the tunnel. As with standard onion routing, the second half of the tunnel should add encryption with each layer, rather than removing it. The end TEs should forward the final packet to the user and exit nodes. Once received, the packets should be decrypted, and the inner nonce decrypted. Both nodes should finally send each other the decrypted nonces back through the TOR tunnel, to verify that the tunnel is fully established.

5.2 Usage

Once the verification session is established, the entire setup may be thought of as two tunnels- the main TOR tunnel, which moves data, and the SafeExit tunnel, which logs the linkage and may provide some out-of-band communication for pertinent information. Most important among this information is tunnel teardown and certificate revocation. Tunnel teardown occurs when the TOR tunnel is being closed, whether due to the natural expiration of the TOR tunnel, or some early closing event. In both cases, the SafeExit tunnel also needs to be closed. This requires that the participating nodes forward requests through the tunnel, and act in response to the signaling. The number of signals which may be sent is relatively small, but the systems should be extensible to allow for future expansion upon the protocol.

Tunnel teardown consists of two parts, initialization and confirmation. The intent of this procedure is to provide a way for a tunnel to be torn down, but to prevent such a teardown from occurring before both sides are ready. The user should be able to initiate teardown at any point in time. However, the exit node must also be informed of the teardown prior to the teardown actually taking effect. While it is less important that the user be assured that the tunnel is still active, it is still useful to do so, so the protocol is symmetrical.

When tunnel teardown is initiated, the end initiating the teardown is responsible for tearing down the tunnel on its half of the tunnel, that is, all nodes between the CE and itself. However, the first step to inform the other endpoint of the intention to tear down. This should be done by using a signaling message which informs the other endpoint of this. This message only needs to contain a TEARDOWN REQUEST signal message encrypted with the pre-established session key for the destination and sent through the tunnel. On receiving this message, the other node should send back an acknowledgment, encrypted with its own half of the session key.

In the case that no acknowledgment is received by the initiator, one or more retries should be attempted. These should consist of retransmitting the teardown signal as before. In the event that no acknowledgment is received by the sender, the sender may initiate an incomplete teardown on its own, under the assumption that the tunnel has been interrupted when multiple attempts to contact the other side fail.

In both the success and failure cases, the node initiating the shutdown needs to build a bundle of onion-encrypted shutdown messages, with the innermost messaged directed to the CE, and the outermost directed to the nearest TE. Besides the next layer of the onion, the only other piece of information strictly necessary is a shutdown signal. To prevent replay attacks, a nonce should be added to each layer to produce encrypt (SHUTDOWN, nonce, encrypt(...)). However, unlike traditional onion signaling, each node needs to append the connection ID it shares with the destination node. This requires that after decrypting the layer of the bundle intended for a specific node, that node must then append the ID of the connection it shares with the next hop, and then encrypt it with the key of the next node.

On receiving the shutdown signal, each node should log when the connection was terminated, and remove the tunnel ID pairing from the active list, while ensuring that all required data is logged. In the event of a shutdown signal not reaching the node, the tunnel should close when the tunnel would have expired normally. Much like when the tunnel expires, each node should log the data, but instead of logging only the time of expiration, should also log the time of last communication received through the tunnel and set the status flag in the log accordingly. How this differs from a normal expiration closure is simply that the time between the last communication was unusually long, causing a timeout during the standard closure; its significance is primarily as operational edge case.

A small addition to the model may be the logging of the conditions under which a tunnel terminated, whether this was a normal shutdown, an expiration, or and expiration with communication loss. This may be useful in gathering network health metrics, since a large number of unusual terminations may indicate issues within the network. However, this may pose a side-channel correlation attack, if an attack is somehow aware that a user is using a particular node, and is able to force a particular abnormal disconnection state. However, this is really only of concern in the case where the attacker is able to get the logs for a node, but is unable to compromise its active operations.

Chapter 6

Analysis of the Advanced Model

The advanced model is intended to address the key flaws of the basic model, and create a system which is potentially usable in the real world. The key points of analysis cover the anonymity of a correctly functioning system, the effects of eavesdroppers, the impact of compromised or malicious nodes, and the effects of colluding nodes.

6.1 Anonymity

In a properly functioning network, knowledge of the user's identity is kept separate from the user's activities by a chain of TEs. The exit node has full knowledge of the user's actions, but no knowledge of the user's identity. The data stream travels via normal, unmodified TOR, so any leak of information that applies to TOR applies to this model, as does the overall anonymity of TOR itself. Of greater concern is the revocation chain of trust introduced by the SafeExit model. This chain can be though of as having three main points of interest- the user's chosen first TE, the exit node's chosen first TE, and the CE. A fourth point of interest is an intermediate TE, although they are of far less importance due to their extremely limited role in the connection.

The user's first TE knows the user's identity, the next TE in the chain, and the CE. The CE effectively allows the chain of TEs to be circumvented for some purposes, but the damage there is limited. Because each hop has different forwards and backwards connection IDs, the connection

ID between the first TE and the next hop is different from the connection ID between the CE and the previous hop, unless the CE is the next hop. Because all logs identify only the per-hop connection IDs, circumventing the full chain of TEs would require a collusion attack, as discussed later.

The exit node's first TE is minimally aware of activity on its own. It is aware of the exit node of the TOR tunnel, by necessity, and should also be in possession of the temporary public key for the connection. It also knows the identity of the CE and the next hop in the chain (if this hop is not already the CE). This is analogous to the knowledge possessed by the user's TE, although the importance is somewhat lessened- exit nodes are already publicly known and, in fact, their identity must be known to the user in order to form the TOR tunnel. Also, like its user-side counterpart, while it is aware of the connection ID for the next hop, it is not aware of the connection ID for the CE, preventing the trust chain from being skipped.

The CE forms a critical point in the SafeExit chain- it is the midpoint at which the sides meet, and it is aware of both the user's chosen TE and the exit node's chosen TE, although it is aware of neither the user's identity, nor the identity of the exit node. It is another entity which stores a copy of the public side of the PKI exchange. Finally, it contains knowledge of the next hop and local session ID for both the user-directed and exit node-directed halves of the tunnel.

The intermediate nodes do not know the identity of the user, exit node, CE, user's first TE, or the exit node's first TE. While the nodes adjacent to the CE or the end TEs do know their identities, they only known them as other TEs, and do not know their importance to the tunnel as a whole. All an intermediate TE knows is the next hop, the previous hop, the next hop's connection ID, and the previous hop's connection ID. They should also hold additional copies of the user's public session key.

6.2 Eavesdropping

An eavesdropper should not be able to learn any details of what the user is communicating, nor should they be able to gain any information regarding the identities of the user and exit node. What is not hidden is the fact that any two parties are communicating with each other, although the contents of the communication should be unknown. This is a reasonable assumption, given that TOR itself makes the same assumptions; an attacker who could exploit this would already be able to directly attack TOR. As before, because standard TOR is used once SafeExit verification is complete, all security issues relating to TOR proper will also effect SafeExit.

All SafeExit links are encrypted, preventing an eavesdropper from intercepting any plaintext. This limits leaks to the fact that the parties are communicating, and the timing of these communications. Even if the session key for a given link was known to the attacker, the amount of data observable on one link is rather low- all traffic is also encrypted end-to-end. Compromises of each node's long-term PKI key are discussed later; this section examines only compromises of the link, not the nodes on either side of it.

On the connection between the user and the first TE, one piece of information is potentially leaked by virtue of the parties communicating, in spite of the encryption- the fact that the first TE is one of the user's trusted TEs. an eavesdropper watching the user for an extended period of time may be able to build up the full list of the user's trusted TEs. Fortunately, this list should not be visible to any other parties in the chain, preventing a simple fingerprinting attack, but an eavesdropper's ability to acquire this knowledge should still be noted.

The link between the exit node and it's chosen first TE suffers from a similar situation- an attacker cannot determine the contents of the communication due to the encryption, but it can determine that the entity being communicated with is one of the exit node's trusted TEs. While this is arguably less compromising to an exit node (which is already publicly registered as such) as compared to the user, the same protections should limit the usefulness of this knowledge to an attacker.

Both links to the CE function identically, and the amount of data compromised is the same for both. As with the other links, the data sent over the link is encrypted, so an eavesdropper is unable to see plaintext data. However, the CE is not an endpoint in the link, and as such, it it only communicates with other TEs. In this case, an eavesdropper is able to potentially determine that the CE and the next TE are communicating, but should not be able to determine that the CE is the CE of this link, as opposed to another node. One potential attack that should be noted is that if the encrypted packets are not padded to uniform length, this does not hold true; an attacker who had compromised the links on both sides of the CE will be able to determine that the packets are of equal size, which can only occur at the CE.

An intermediate link, that is, one between two ordinary TEs, should behave almost identically to a link between a CE and a TE. The only observable difference is that an unpadded packet's size will only dictate the direction of the link; unpadded packets on the side closer to the TE will be smaller, because a layer of encryption and data has been stripped off. As before, padded packets prevent this attack.

6.3 Compromised Nodes

One of the major points of concern in any network which requires trust is when a party to which trust has been assigned is either malicious, or has been compromised by an external malicious party. For the purposes of this analysis, the two scenarios are considered functionally identical- that is, it is assumed that an attacker which compromises a node is able to gain full control over, and intercept all information known to said node.

The key points of compromise are the same ones as before: the user's first TE, the exit node's first TE, the CE, and any other intermediate nodes. One attack that applies to all of these is a simple disruption of service; any compromised node can refuse to participate properly in the network. However, this attack is somewhat limited, because a failure to connect will only result in another attempt being made via a different route. While this could potentially be used to manipulate route selection, neighboring nodes are able to detect the high connection failure rate. Node operators are likely to notice the high number of failed connections from such a malicious node, and potentially remove their trust link to this node.

If the user's first TE is compromised, the attacker is able to know the user's IP. However, despite knowing the user's IP, it still only knows the identity and hop ID for the next hop, as well as the identity of (but not the user-facing hop ID for) the CE. All other data passing through the node is encrypted. The first hop is probably the most dangerous to have compromised from the user's perspective, since it is able to link their IP to the timing of their use of the SafeExit connection. However, it is unable to link the user to which exit node they were using, limiting the usefulness of an attack.

Similarly, the exit node's first hop knows the identity of the exit node, the identity of the CE, and both the identity and connection ID for the next hop. However, unlike the user's first hop, the identity of the exit node, and the fact that it is a SafeExit exit node, are both publicly known. The identity of the compromised node as one of the exit node's trusted nodes is of less concern than it is to the user, because the exit node's identity is already known.

A compromised CE is a point of concern, because of its critical role in the network. However, it still has limited information about the connection passing through it. The CE is aware of the identities of not only the next and previous hops, but also the user's first TE and the exit node's first TE. However, it is still only aware of the connection IDs for the next and previous hop, like other nodes in the middle of the tunnel. This means that a compromised TE is able to link the first TE and the last TE of the tunnel, although it is unable to determine which log entry would correspond to the actual connection on either end. This does make a compromised CE somewhat more dangerous than the typical node, but the information is not able to compromise the identity of the user on its own unless the attacker is not only able to get the logs of the exit node and the first TE, but also there are very few tunnels passing through both of these additional nodes at that time.

The intermediate nodes, as intermediaries, simply know the identity and connection Id for the next and previous hop. This poses even less danger than the other nodes, because they do not know the identity of the endpoints (or, if the next or previous hop is an endpoint, they have no way of knowing that offhand). A relative timing attack might be employed, where the time it takes for a presumed request/response pair might be used to roughly estimate the distance to the endpoints, but this would likely be more unreliable than in core TOR, because of the ability of a node to introduce higher latency.

6.4 Collusion

The effects of nodes colluding is a matter of great concern. As with all collusions attacks, the more nodes involved, the greater the danger of unmasking the user- if the entire tunnel passes through cooperating malicious nodes, the attacker knows everything about the tunnel, and therefore the only thing the attacker still needs to compromise the user is to also be able to compromise the exit node. In a network comprised mostly of trustworthy nodes, this attack is unlikely to succeed due to the low probability of the user and exit node both choosing exclusively compromised nodes. This is rendered more unlikely due to the user's ability to reject any CE that they distrust, as well as the user's ability to control the path to the CE. Smaller collusions, however, are always a point of concern for networks such as this. In a perfect system, no grouping of nodes smaller than the entire tunnel would be able to compromise the user.

Timing attacks are somewhat less threatening than in TOR, because the data moving through this tunnel is not actual user internet traffic; no communication is destined for a location outside of the set of participating nodes, and the number of total messages is fairly low. Because no traffic is destined past the end of the network, to truly connect a user's actions to their identity, the exit node and the user's first CE must both be colluding for any attack to occur. Because no traffic from TOR passes through the network, no internet server accessed by the user is able to watermark traffic to be detected by a SafeExit collusion.

In the simplest case of a colluding exit node and first TE, a timing attack exists, more or less equivalent to a known attack on TOR (Murdoch & Danezis (2005)). However, the structure of

SafeExit makes it less vulnerable to such an attack. First, the amount of traffic is low, so the ability of a participant to watermark it via timing is lower; only a few packets are exchanged over the entire lifetime of the tunnel. Additionally, the network is potentially much higher latency than TOR. The attack on TOR relies on the insertion of enough artificial delay to erase the timing watermark being detrimental to the performance of TOR. SafeExit, however, is only used in TOR circuit creation, and so is used rarely enough that an increase in timing noise should have a negligible impact on the usability of the network.

The collusion of the user's first TE with the CE is fairly limited. While this does allow the two to link the user's true identity, known by the first TE, with the identity of the exit node's first TE, not enough information is provided to even identify the exit node. It does reduce the number of meaningful hops between the user and the exit node, but each leg of the tunnel is independent of the other, so the connection of the whole remains uncompromised. If a compromised exit node (with an uncompromised TE) is being used, these colluding nodes do increase the probability of a timing attack working; the colluding CE can conduct the attack, minimizing the number of nodes which might interfere with the timing.

The collusion of the first TE and the last TE poses a similar timing attack danger to a colluding first TE and exit node at first glance, essentially removing one hop. However, because the TE receives all messages sent through the Safe Exit tunnel, but does not need to pass them on to the exit node, there are more messages available with which to conduct timing attacks. This makes the attack somewhat more threatening than merely reducing the effective tunnel length by one hop.

The collusion of the CE and the last TE poses a much lower threat than the collusions involving the first TE, because they are unable to acquire direct knowledge of the user's identity. However, because the CE is aware of the identity of the user's first TE, this collusion chain is capable of creating a link between a chosen first TE and a chosen exit node. Given that a user has both a list of trusted exit nodes and a list of trusted first TEs, This could be used to infer patterns of TE and exit node choice which might inform a targeted attack. If the exit node is compromised, user activity can be linked with the chosen first TE, allowing users with distinctive TOR usage habits to leak their list of trusted TEs. This may be used to generate a set of nodes which could be compromised for a targeted attack, or used to infer information about the user behind a pattern of browsing habits. This could also be used to distinguish users within smaller subsets, such as distinguishing users with similar usage patterns but different TE choices, or users with similar sets of trusted TEs but very different browsing patterns. While none of this directly leaks the user's identity, it could be used to probabilistically infer a user's identity when combined with external information.

The collusion of the user's first TE, the CE, and the exit node is somewhat more concerning than the collusion of the exit node and the user's first TE. While a similar timing attack could be attempted, this combination of colluding nodes is directly dangerous. An attacker with control of all three of these nodes possesses knowledge of the user's identity (via the first TE), knowledge of their activities (via the exit node), and knowledge of a link between the two (via the CE). If only one user is using that particular combination of three nodes, the user's identity can be unmasked immediately. If multiple users are using the same combination of nodes, the attacker can attempt to generate a timing attack to link their connection IDs on the first and last TEs.

6.5 Active Attacks

Active attacks rely on malicious nodes intentionally interfering with the operation of the network, as well as outsiders attempting to interfere with the network. Many general attacks, such as DoS, apply. Attacks on TOR itself also apply, but are not discussed here unless there is a variation that has unique implications for SafeExit.

The most obvious attack on Safe Exit is a DDoS, wherein many malicious users (likely representing machines controlled by a botnet) attempt to overwhelm a node, and thus effectively remove it from the network. The most obvious targets of these are individual TEs; if the attackers imitate a large number of users attempting to create SafeExit tunnels rooted at the node, it will quickly become overwhelmed, and legitimate connection attempts become statistically unlikely to succeed.

Far more dangerous is using this to attempt to manipulate network topology; because any TE can accept user connections, it might be possible to use knowledge of the network topology in combination with these attacks to disconnect portions of the network, which would disrupt service between TEs which had become disconnected. A more connected network would be much less vulnerable to this kind of attack, while a poorly-connected network, or one with bottlenecks, would be highly vulnerable.

One potential threat of an active attack is that a particular user's list of trusted nodes provide their only gateway into the network, if a user's set of trusted nodes were to become known, an attempt could be made to disable or compromise all of these nodes, cutting the user off from SafeExit without directly attacking the user. This is likely a counterproductive attack in this simple form, since the user would still be able to access normal TOR. More dangerous is the possibility that an attacker could disable all nodes trusted by a targeted user except for ones controlled by the attacker. This attack poses a far greater threat, because it provides the groundwork to launch a passive attack requiring a connection via a compromised node.

Disabling all of the nodes at a bottleneck is a threat, if the attacker is aware of TE topology. If the topology had two large groups of nodes who only share trust with a few nodes, attacking the few shared points of trust might disconnect the graphs. This is likely to disrupt many SafeExit users, but cannot effectively be used as part of an identity compromise attack. It may compromise user and TOR performance slightly, by shifting users who would prefer to use SafeExit to use regular TOR, but is unlikely to actually halt service. There is some threat that it could be used to manipulate a user or set of users to use certain, compromised paths. This would function by simply eliminating the ability to create paths through the bottleneck, thereby increasing the odds that any given node on the same side of the bottleneck as the user will be selected in a path.

Using this attack has a higher probability of disrupting SafeExit use to some users than a similar attack on TOR itself would. Because regular Tor nodes can connect to each other freely, and any exit node can be used as an exit, there is limited topology to attack. An attack on a large number of exit nodes or intermediate nodes would impact TOR performance, but any remaining nodes would be capable of connecting with each other freely.

Another important active attack involves a node attempting to impersonate another node in order to conduct a man-in-the-middle attack. This is effectively defeated by the messages being signed end-to-end; because the public keys of all nodes are known, the only way to conduct a successful impersonation attack would require that the private key of that node be compromised, and the attacker would have to be able to intercept messages sent to the IP associated with that key. This kind of attack is common to any form of PKI signature, and so it is beyond the scope of SafeExit.

Most other obvious potential active attacks are easily halted by the heavy use of signatures in SafeExit. A node which forges information will quickly be detected by users, and the default action would be to make another attempt at connection, choosing a different node from the node that delivered bad data. This has the added benefit of routing traffic around nodes which are malfunctioning without malicious intent, making the network more robust.

Chapter 7

Additional Improvements

7.1 High Latency

Unlike a TOR data tunnel, a SafeExit tunnel is primarily used to relay signaling anonymously, as opposed to data. Additionally, while TOR attempt to minimize latency in order to make it more useful for web browsing, SafeExit messaging need not be constrained by the same design principles. Given that a SafeExit tunnel need only occasionally be constructed, and the messages it sends do not consistently have a direct impact on the experience of a user using the system. These techniques are not used in TOR data tunnels, in no small part due to the effects of added latency being very perceptible when applied to a web browsing data stream; latency great enough to have impact on a non-trivial timing attack will tend to create unacceptable performance for the user in such cases.

All of these factors combine to allow SafeExit to use techniques which are not used in TOR data tunnels. One approach would be to add artificial latency of random duration to all messages sent over the SafeExit tunnel, in order to hide timing patterns and possibly reorder communications. While the length of delay will still need to be short, a total latency in the range of seconds is likely to have minimal impact on the performance of the system as a whole. This approach may not defeat all forms of timing attack, but it should complicate them, especially in comparison to regular TOR. The maximum amount of latency added may be controlled by each TA, but may even be in the area of a second or more per node.

This technique is not used in TOR by intentional design- one of the distinguishing features of TOR is that it uses low-latency tunnels in order to make the data transport more useful for all internet traffic. However, because SafeExit exchanges are limited to intermittent signaling, adding sufficient delay to mask signals may be feasible. Such an extension is probably best implemented as an optional, rather than mandatory, security option on the client end- if a user requests delay, then it should be provided, with users who are not using any kind of delayed messaging serving as cover traffic. This strikes a compromise between user who might be willing to sacrifice several seconds to set up a tunnel in exchange for increased traffic monitoring resistance and users who are unwilling to do so.

The main detriment to the usefulness of this potential extension is that because SafeExit already requires trust in the node chain, it sits at an anonymity level below traditional TOR, so users who are willing to opt into its use will have lower anonymity needs as well. Because TOR is currently vulnerable to the kind of global (or at least widespread) observation attack that high latency aims to defeat, the number of users who would be willing to sacrifice higher anonymity for better performance but are also willing to delay connection by several seconds to potentially increase anonymity is likely fairly low.

Another critical point of concern is the question of how much latency would need to be added to destroy the signal of a timing attack. More research is needed in order to determine exactly how well a timing signal survives latency, especially considering that the existing timing attacks on TOR have not been designed with high latency resistance as a design goal.

7.2 Masked Usage

One potential point of concern over the use of Safe Exit is that the users distinguish themselves from ordinary TOR users to an observer- simply by contacting the TE, the user indicates to anyone who is watching that they are not just using normal TOR, but also SafeExit. One way in which ordinary this effect might be mitigated is for ordinary TOR users to occasionally go through a false protocol exchange with a TE. The purpose of this exchange is to have any attacker examining the traffic believe that they are seeing a user make use of SafeExit, when in reality the user is only using normal TOR. This extension could be particularly helpful, since it would mask the security needs of the individual from an attacker as well.

This would require that a special flag be inserted in the encrypted portion of the fake message, telling the TE to complete the exchange in precisely the same manner as if it were an ordinary SafeExit connection, but forward nothing. This is obviously only resistant to an observer watching the user and not the user and the TE, but could be elaborated to defeat more complex attacks by increasing the number of steps in the procedure which are simulated.

In order to keep the system symmetrical, and keep SafeExit users from depleting the K-anonymity of standard TOR, it may also be helpful to have SafeExit users randomly use a standard TOR connection. Combined with the previously described false SafeExit connection, this provides masking of the true anonymity level the user has requested- normal TOR users seem to occasionally use SafeExit; SafeExit users occasionally use normal TOR. This also provides plausible deniability for a change in behavior; if a user occasionally uses normal TOR instead of SafeExit due to an increased need for anonymity, this would appear to an outside observer to merely be a random automatic use of normal TOR.

The core basis of this extension is that idea that all users of TOR should appear identical if an attacker is observing their connection. Because the operation of TOR itself is not altered by using SafeExit, and attacker's only means of determining whether a given user has opted to use SafeExit is to observe the exchanges of the SafeExit protocol. By simulating this exchange when SafeExit is not used, an attacker cannot reliably associate the SafeExit communications with SafeExit use. Having users who have opted to use SafeExit also occasionally use non-SafeExit nodes provides a similar safeguard for core TOR users- lack of a SafeExit signature does not conclusively correlate to lack of SafeExit use.

Care must be taken in implementation to avoid long-term statistical differences between SafeExit-requesting users and regular TOR users; if an attacker can easily determine which mode is in use for a given session via statistical analysis, this feature has limited usefulness.

7.3 Anonymity Level Integration

While this is dependent on other work adding more discrete levels of anonymity to TOR, masked usage might be used as part of a framework for integrating the use of other anonymity levels into TOR. This could allow for a variety of settings altering the balance between anonymity and connection performance. Such a system would have to be carefully implemented to prevent monitoring the user's connections allowing an attacker to determine the mode in which they operating.

The specific properties of these hypothetical alternate modes would vary depending on both their design and specific implementation. In general, care should be taken to avoid modes that alter the behavior of data between the user and the first relay, to prevent a mode detection attack. Out-of channel communication, like SafeExit, can be handled similarly to the approach given above, although higher-anonymity modes that require out-of channel communication suffer from the inability to revert to less-anonymous versions without compromising the requested level of anonymity.

That is, assuming that using a lower anonymity mode than is request is unacceptable, the presence of a high-anonymity out-of-channel signal allows a weak partitioning attack. While the attacker cannot assert that the presence of the signal implies that the source of the signal requested the higher anonymity mode, they can assert that all users who have requested the higher anonymity mode either display the signal, or are operating at a higher mode than requested (assuming that such a higher mode exists). With masking, this does not interfere to SafeExit (because the out-of channel communication only occurs in a lower-anonymity mode than standard TOR, which does not use out-of-channel signaling); however, it is a consideration which needs to be addressed if this is to be expanded into a full system.

Chapter 8

Related Works

8.1 Core TOR

As a TOR extension, Safe Exit would not be possible without the vast amount of work which has already been put forth into TOR.

8.1.1 Tor: The Second Generation Onion Router

In this seminal paper by Dingledine et al. (2004), the core of TOR was initially laid out. Despite numerous minor changes in the operational version of TOR which have occurred since the initial publication, the general design principles in this paper remain crucial to TOR today.

Among other things, it describes the goals of TOR, the basic network structure, the process of creating circuits, the justification for each layer of encryption, and a security analysis of this form of onion routing. While innovations such as entry guards and hidden services were not yet introduced, all these are are additions to the work presented in this paper.

8.1.2 Low-Cost Traffic Analysis of Tor

Murdoch & Danezis (2005) present one of the most important attacks on TOR in this paper, one which remains a major issue with the network to this day. This attack consists of a hostile website generating a signal in the form of a distinct delay pattern, which can then be received by a hostile TOR entry node.

This attack of particular concern, since it has been shown that the pattern is likely to survive reasonable message delays within TOR, preventing simple solutions that do not compromise the usability of TOR itself. Additionally, the signal can be injected by a hostile exit node as well as a web site, allowing two colluding hostile nodes to connect the identity of any user which uses one as an entry and one as an exit.

8.1.3 Browser-Based Attacks on TOR

The previous work on using delay-based signals to identify TOR users was expanded upon by Abbott et al. (2007) in this paper, which presents a usable attack embedded in a website using Javascript, which can be embedded in an innocent web page by a malicious exit node. This attack extends the previous one by extending the attack window across multiple TOR circuits- if the user has left the page with the attack script running, the script will be detected by malicious entry points after TOR generates new circuits.

Additionally, a method is presented in which the HTML refresh tag may have its refresh rate dynamically generated, injecting the signal into automatic page refreshes, although the authors note that this is more likely to be detected by the user. Finally, it presents a means for a node to increase the odds of detecting the timing signal in the attack via a strict exit policy.

8.1.4 The Sybil Attack

Because TOR can be compromised if an attacker were able to ensure that a user's first and last TOR relays are both controlled by the attacker, a major concern is the Sybil attack, presented in this paper by Douceur (2002). The attack functions when a single attacker produces multiple nodes in the network, often by spoofing multiple identities. This is addressed in the implementation of TOR by restricting tunnels from being created with multiple nodes in the same IP block, and in SafeExit, is potentially negated by the requirement of manual link creation. However, it remains one of the more important potential attacks on anonymity networks.

8.1.5 Shining Light in Dark Places: Understanding the Tor Network

This paper from McCoy et al. (2008) analyses how TOR is used in the real world setting, including both analysis of traffic patterns and geographic distribution of TOR routers. The analysis of actual traffic reveals that most TOR connections are web traffic, although (at the time) Bittorrent consumed a large percentage of bandwidth. It also categorizes some misuse, both by users and potentially by malicious exit nodes.

Of particular interest is the geographical distribution of routers, which reveals that a fairly small number of countries have most of the TOR routers. This trend is a major motivation for this work, since increasing the number and variety of nodes in TOR, especially exit nodes, will improve the usefulness of the network.

8.1.6 Website Fingerprinting in Onion Routing Based Anonymization Networks

This paper presents a machine learning-based attack on TOR. Panchenko et al. (2011) show a means by which websites access patterns may be fingerprinted in advance using a support vector classification engine. This allows an attacker who is able to eavesdrop on the user's first-hop connection to TOR to detect access to specific sites via TOR, with a reasonable probability of success in the test scenario.

However, even in the test scenario, only about half of sites were vulnerable to this fingerprinting, although the false positive probability was less than 1% in the test cases. While this attack is concerning in itself, it also further serves to highlight the importance of studying more elaborate timing attacks.

8.1.7 An Improved Algorithm for Tor Circuit Scheduling

Tang & Goldberg (2010) present a potentially major performance improvement for TOR in this paper. By observing that a major cause of excessive latency in the network was bulk data transfer connections starving interactive connections of resources, a prioritization scheme was developed which allows low-bandwidth interactive streams to avoid severe delays without greatly interfering with the bulk data streams.

This approach uses the recent amount of data transmitted through a circuit to establish priority, using an exponentially weighted moving average. This metric means that bulk data transfers behave mostly as they did before, but interactive circuits tend to avoid large delays caused by bulk transfers clogging a router's output queue.

This idea is of particular note because a version of it has been implemented in the deployed TOR network, where it saw great success in keeping the large volume of Bittorrent traffic the network was seeing at the time from significantly degrading web browsing performance. Despite the reduced popularity of tunneling Bittorrent over TOR for a variety of reasons, this approach still prevents large downloads of any kind from degrading the performance of web browsing.

8.1.8 Locating Hidden Servers

While the focus of this paper from Øverlier & Syverson (2006) is the detection of hidden services, it also provides motivation for and proposes guard nodes, as later adopted by TOR. Because servers typically have the longest continuous connection durations, they were especially vulnerable to the possibility of cycling through many initial relays and encountering a malicious relay. Guard nodes reduce the vulnerability of such long-term TOR sessions by limiting the number of possibly contacts with malicious nodes.

8.1.9 How Much Anonymity does Network Latency Leak?

This paper by Hopper et al. (2010) provides a larger-scale experiment using timing to unmask TOR users, using various round trip times within the network. While the conclusions were in line with earlier work, the larger data set is extremely useful in confirming that timing attacks are cause for practical, rather than merely theoretical, concern. Most of these attacks are able to reduce the pool of possible users who might be associated with a connection, even if they cannot completely unmask the user.

8.2 Additional TOR

While SafeExit does not touch upon certain features of TOR due to its nature, these features are nonetheless significant to both TOR and the larger field of anonymous communications.

8.2.1 Performance Measurement and Statistic of Tor Hidden Services

This paper from Loesing et al. (2008) provided performance measurements of TOR hidden services, and some analysis of the factors impacting the performance. While hidden services face different performance issues than standard TOR, the analysis still provides insight into the performance of the network.

8.2.2 A Tune-up for Tor: Improving Security and Performance in the Tor Network

This paper from Snader & Borisov (2008) points out a potential means of network manipulation in TOR, wherein both performance issues and route manipulation can occur due to routers self-reporting their capabilities. A method of estimating actual performance is proposed, which is intended to prevent poor estimates or intentional misrepresentation of a router's capabilities from interfering with the network. Additionally, this paper suggests that different users may wish to use different router selection algorithms based on their desired performance/anonymity balance.

8.3 Other Anonymity Networks

While TOR is almost certainly the most widely used low-latency anonymity network, several more exist and are under active development. While they differ from TOR on many levels, and have somewhat different goals, the ideas within them are still important to the field.

8.3.1 Freenet: A Distributed Anonymous Information Storage and Retrieval System

This paper from Clarke et al. (2000) proposed Freenet, one of the major currently deployed anonymity networks. It takes a rather different approach from TOR, creating a separate network from the open internet. Unlike TOR and I2P, Freenet is less focused on conventional web hosting, and more focused on redundant, distributed, encrypted file storage.

The general idea behind the network is that each user in the network volunteers some amount of data storage to the network. This storage is used to stored encrypted data bundles for the network, which are duplicated as they are accessed. Data persistence is not guaranteed, and rarely accessed files will eventually be lost as more-frequently accessed data replaces their copies in all the data stores. As deployed, web pages hosted in Freenet come with a series of conventions in their design, which prevent pages from being partially lost, as well as simple page versioning system which allow an updated web page to supersede an older version.

Because Freenet is based on storage rather than transport, it cannot host conventional web services with server-side software producing dynamic content, which both TOR and I2P can do. However, it gains the benefit of allowing a user to anonymously upload data, and then disconnect from the network without halting others from accessing the data, leaving a much smaller window for an attack on a specific uploading user. Additionally, the lack of dynamic content prevents or mitigates some known attacks on TOR.

Freenet, as a peer-to-peer system, also requires users to have some trust in their immediate neighbors. As such, the network offers a "darknet" mode, in which a user must explicitly choose their peers. This has the immediate benefit of offering the ability to only have neighbors which the user trusts to not be malicious, although it also has the downside of requiring that the user know such peers.

8.3.2 I2P

I2P represents a similar idea to TOR, but with somewhat different goals and considerably different implementations. While both TOR and I2P provide low-latency anonymity, I2P is designed specifically to host location-hidden web services, somewhat analogous to TOR's hidden services. The network is also fully decentralized, with most users forwarding traffic. Because all traffic is kept inside the network, there are no exit nodes, as such (unless you consider proxies to the regular internet or TOR being offered as I2P services to be "exits"). Like Freenet, I2P can also operate in a darknet mode, where only trusted nodes are used as neighbors.

On a deeper level, I2P also uses a rather different method of creating tunnels, wherein a user creates separate incoming and outgoing tunnels. These are effectively reservation of network resources, allowing a one-way onion-encrypted connection among peers. The incoming tunnels are publicly advertized as valid, ephemeral addresses for a user, and the outgoing tunnels are similarly ephemeral paths for outgoing traffic. This is intended to increase the complexity of carrying out some kinds of attacks. A connection to an I2P service requires that the user send data to the service's incoming tunnel via the user's outgoing tunnel, and traffic from the service be sent through the service's outgoing tunnel to the user's incoming tunnel. All traffic is encrypted end-to-end and hop-by-hop throughout the process. The current incoming tunnels for a node are stored via a distributed hash table, removing the need for a central authority to manage the current network state.

It is worth noting that I2P's authors prefer to remain anonymous, and despite it being a popular anonymity network, there is relatively little academic literature dedicated to it.

8.3.3 Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study using I2P

This paper from Herrmann & Grothoff (2011) presents a means of using I2P's router performance algorithms to increase the probability of a user or service becoming vulnerable to attack. Because I2P prefers using neighbor which it detect as being high-performance suing a local detection algorithm, an attacker can make their nodes more likely to be chosen as the nearest hop in an incoming or outgoing tunnel for a specific target. By doing so, an attacker gains the ability to disrupt service to a user, or even begin attempts at unmasking, However, the authors suggested several means to mitigate the effectiveness of the attack.

8.4 Trust and Anonymity

Many papers referenced during the design of SafeExit are not specifically related to any anonymity network, although they were still relevant to the design of SafeExit.

8.4.1 Decentralized Trust Management

In this paper, Blaze et al. (1996) present a language designed to allow trust policies to be accurately and concisely expressed. This language allows all parties in a system to manage which credentials they trust, and specify trusted third-party sources of relevant credentials. The language is designed so that the underlying verification procedures are completely encapsulated from the larger-scale issue of what needs to be verified and which sources of identity credentials are trusted.

8.4.2 Propagation of Trust and Distrust

Guha et al. (2004) present a framework of directed trust propagation, as well as an analysis of a large network using0 such trust propagation. This model provided for non-binary trust analysis among many users, taking into account both positive trust and negative trust (distrust) to produce a complex aggregation of trust values.

8.4.3 A Probabilistic Trust Model for GnuPG

This paper by Haenni & Jonczy (2007) presents a modification to the existing GnuPG trust evaluation rules, whereby it treat trust propagation as probabilistic, as opposed to the existing manual trinary trust system. By replacing the three trust classes of a user (trusted, marginally trusted, and untrusted) with a probabilistic trust level, much more nuanced amounts of trust can be expressed than the small number of discreet classes allows.

8.4.4 New directions in cryptography

This seminal paper in digital cryptography from Diffie & Hellman (1976) provides a classic key exchange method. Using this method, two parties can exchange sufficient information for each one to create an identical key, without either one disclosing the key itself across the link they are communicating over. While the basic model is vulnerable to a man-in-the-middle attack due tot he lack of authentication of the source of the key parts, this can be negated using PKI to authenticate the communicating parties.

Chapter 9

Conclusion

The TOR anonymity network is currently one of the most popular systems to allow anonymous internet access, and is certainly the most well-researched system at present. However, the network benefits greatly from both more diverse users, and from more routers contributing to the network. Most important to the network are exit routers, which are crucial to the network, but make up a relatively small number of routers due to the complications with hosting them.

Because exit routers are critical to the TOR network, encouraging more people to volunteer to run them would be highly beneficial to the network. However, there is concern that misbehaving TOR users may create problems for those running exit nodes, and due to the nature of TOR, it is difficult to prove that the exit itself was not the source of the traffic. This is potentially a major reason why people may hesitate to run a TOR exit, because bad user behavior is possible regardless of exit policy, and may create problems for the person or organization hosting the exit.

SafeExit is an attempt to provide a way for routers which would otherwise be unable or unwilling to serve as exits and contribute more to TOR as limited exits. While SafeExit nodes are less useful to the network as a whole than a regular exit node, they provide a way for more people to operate as exit nodes, which is ultimately beneficial to the network.

This does require the creation of a parallel structure of verification nodes, which are designed to provide a verifiable chain leading back to the user while preventing malicious tracing through the length of the chain. However, the demands of running a SafeExit node should be relatively low, unless the node is extremely popular or at a popular junction in the network. However, TOR itself would remain unchanged for those who do not use the system, avoiding major network overhauls. Additionally, because no data is transported over the additional SafeExit infrastructure, operating a TE should be lower-risk than operating a standard TOR router.

This comes at a compromise, because it is only usable by users who have less of an anonymity requirement than what normal TOR provides. However, users who use this lower level of anonymity would be able to potentially receive increased performance, creating a trade-off. While some users do require TOR's full anonymity, they would still potentially be able to benefit from SafeExit due to the load taken by SafeExit exits. Additionally, a similar signaling structure could be used as part of other opt-in extensions to TOR, potentially allowing users more control over their anonymity/performance balance.

9.1 Future Work

Because all of this work is theoretical, a test implementation of the protocol would be useful for testing real-world performance. It would also provide a means for more thorough security analysis, and once this analysis is completed, a baseline for a deployable implementation.

As with any security-related design, multiple third-party security analyses would be beneficial before widespread deployment of the system. Post-implementation, further analysis of both the performance of implementation itself would be crucial, in order to ensure that no security issue crept into the implementation.

While there are relatively few barriers to operating a regular TOR relay (certainly in comparison to an exit), expanding the number of standard relays would also be beneficial to the network. Because there are fewer reasons for people not to run an intermediate relay, this may not be something that can be addressed through technical means, however.

While SafeExit seeks to improve TOR performance by allowing new exit routers to participate and by allowing users with lower anonymity needs to offload some of their traffic to these new exits, there is room for expansion in several directions. A complementary system wherein users can opt into a mode where they are willing to sacrifice performance for increased anonymity is a natural extension. While previous research has determined that inserting delays to traffic which are long enough to thwart traffic analysis attacks would render TOR unusable for most users, there may be room to add it as a user-selectable mode for users who understand the performance impact and desire the increased anonymity.

References

Abbott, T. G., Lai, K. J., Lieberman, M. R., & Price, E. C. (2007). Browser-based attacks on tor. In *Privacy Enhancing Technologies* (pp. 184–199).: Springer.

Blaze, M., Feigenbaum, J., & Lacy, J. (1996). Decentralized trust management. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on* (pp. 164–173).: IEEE.

Clarke, I., Sandberg, O., Wiley, B., & Hong, T. W. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (pp. 46–66).

Diffie, W. & Hellman, M. E. (1976). New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6), 644–654.

Dingledine, R., Mathewson, N., & Syverson, P. (2004). *Tor: The second-generation onion router*. Technical report, DTIC Document.

Douceur, J. R. (2002). The sybil attack. In *Peer-to-peer Systems* (pp. 251–260). Springer.

Guha, R., Kumar, R., Raghavan, P., & Tomkins, A. (2004). Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web* (pp. 403–412).: ACM.

Haenni, R. & Jonczy, J. (2007). A new approach to pgp's web of trust. In *EEMA'07, European e-Identity Conference*.

Herrmann, M. & Grothoff, C. (2011). Privacy implications of performance-based peer selection by onion routers: A real-world case study using i2p. In *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS 2011).*

Hopper, N., Vasserman, E. Y., & Chan-Tin, E. (2010). How much anonymity does network latency leak? *ACM Transactions on Information and System Security*, 13(2).

Loesing, K., Sandmann, W., Wilms, C., & Wirtz, G. (2008). Performance measurements and statistics of tor hidden services. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on* (pp. 1–7).: IEEE.

McCoy, D., Bauer, K., Grunwald, D., Kohno, T., & Sicker, D. (2008). Shining light in dark places: Understanding the tor network. In *Privacy Enhancing Technologies* (pp. 63–76).: Springer.

Murdoch, S. J. & Danezis, G. (2005). Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on* (pp. 183–195).: IEEE.

Øverlier, L. & Syverson, P. (2006). Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*: IEEE CS.

Panchenko, A., Niessen, L., Zinnen, A., & Engel, T. (2011). Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society* (pp. 103–114).: ACM.

Snader, R. & Borisov, N. (2008). A tune-up for tor: Improving security and performance in the tor network. In *NDSS*, volume 8 (pp. 127).

Tang, C. & Goldberg, I. (2010). An improved algorithm for tor circuit scheduling. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 329–339).: ACM.