Towards an Analysis of Onion Routing Security

Paul Syverson * Gene Tsudik † Michael Reed * Carl Landwehr ‡

Abstract

This paper presents a security analysis of Onion Routing, an application independent infrastructure for traffic-analysis-resistant and anonymous Internet connections. It also includes an overview of the current system design, definitions of security goals and new adversary models.

Keywords: Security, privacy, anonymity, traffic analysis.

1 Introduction

This paper presents a security analysis of Onion Routing, an application independent infrastructure for traffic-analysis-resistant and anonymous Internet connections. It also includes an overview of the new system, definitions of security goals and new adversary models. Although the conceptual development and informal arguments about the security of Onion Routing have been presented elsewhere [9, 15, 16, 10], we have not previously attempted to analyze or quantify the security provided against specific attacks in detail. That is the primary contribution of this paper.

The primary goal of Onion Routing is to provide strongly private communications in real time over a public network at reasonable cost and efficiency. Communications are intended to be private in the sense that an eavesdropper on the public network cannot determine either the contents of messages flowing from Alice and Bob or even whether Alice and Bob are communicating with each other. A secondary goal is to provide anonymity to the sender and receiver, so that Alice may receive messages but be unable to identify the sender, even though she may be able to reply to those messages.

An initial design has been implemented and fielded to demonstrate the feasibility of the approach. This prototype, which uses computers operating at the Naval Research Laboratory in Washington, D.C., to simulate a network of five Onion Routing nodes, attracted increasing use over the two years it was available. While in operation, users in more than sixty countries and all seven major US top level domains initiated up to 1.5 million connections per month through the prototype system; cf. also Figure 1, which shows connections per day averaged over the preceding 30 days. This demand demonstrates both an interest in the service and the feasibility of the approach. However, the initial prototype lacked a number of features needed to make the system robust and scalable, and to resist insider attacks or more extensive eavesdropping. A design for a second generation system that addresses these issues is complete, and the processes required to release the source code for public distribution have been initiated. Several companies have contacted NRL to with intent to commercially license Onion Routing.

This paper analyzes the protection provided by the second generation design. We start by describing, briefly, the architecture and features of the second generation system relevant to our analysis. In section 3 we define security goals for anonymity and/or traffic-analysis-resistance. In section 4 we give some assumptions about the configuration of our network. In section 5, we set out our adversary model. In section 6, we present a security assessment based on the definitions and assumptions made in earlier sections. Finally, we compare Onion Routing to systems with similar goals, most specifically with Crowds [17].

*Center for High Assurance Computer Systems, Code 5540, Naval Research Laboratory, Washington DC 20375, USA. {*lastname*}@itd.nrl.navy.mil

†Information and Computer Science Dept., University of California, Irvine CA 92697-3425, USA. gts@ics.uci.edu

‡Mitretek Systems, Inc., 7525 Colshire Drive, McLean VA 22102, USA. Carl.Landwehr@mitretek.org (Work by this author was primarily performed while employed at the Naval Research Laboratory.)

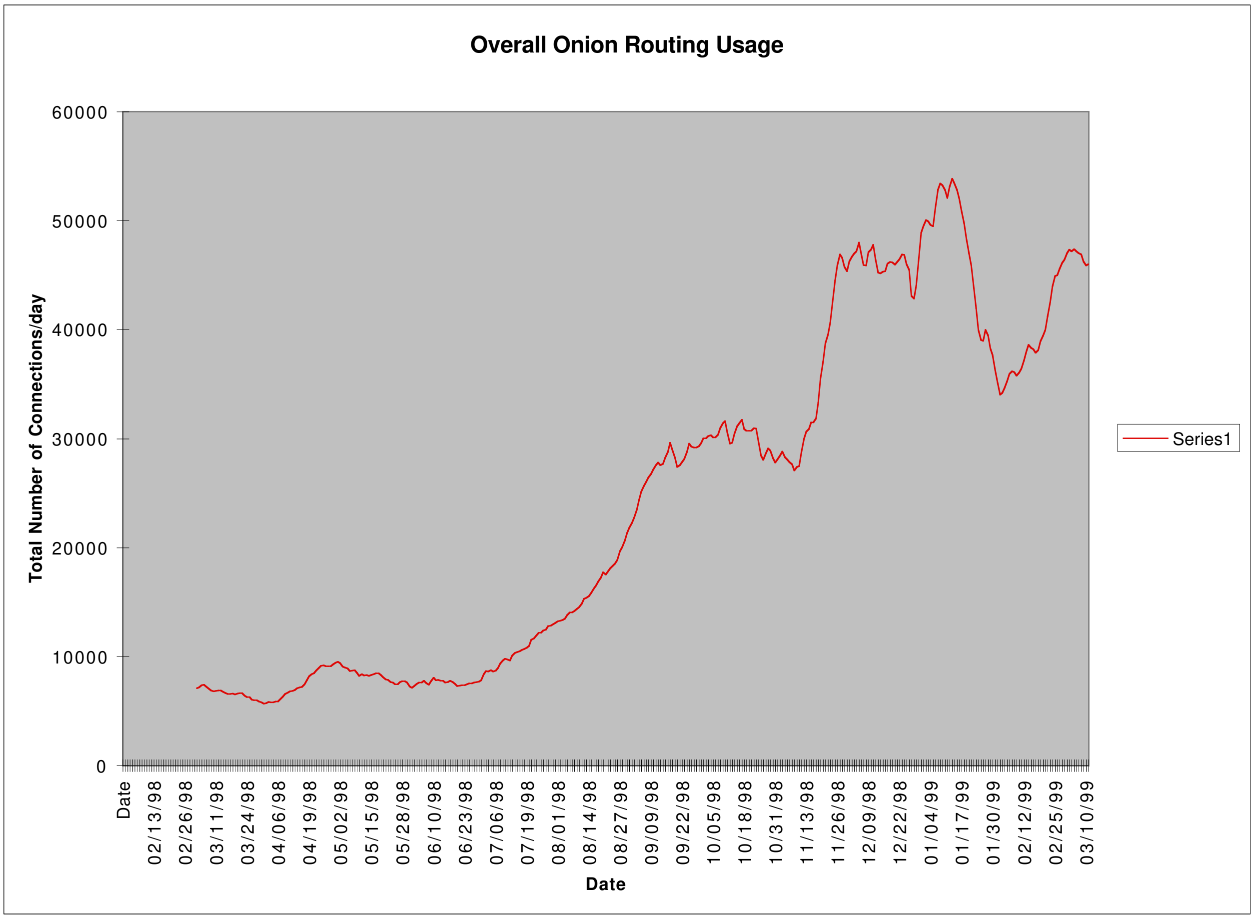

Figure 1: 30 Day Rolling Average of Onion Routing Usage: 3/1/98 – 3/1/99

2 Onion Routing Overview

This section provides a brief overview of Onion Routing for readers not familiar with it. Conceptual development of Onion Routing as well as a description of the design for the previous system can be found in [9, 16]. Brief description of different aspects of the current design can be found in [10, 20]. Readers familiar with Onion Routing may wish to skip to the next section.

Onion Routing builds anonymous connections within a network of onion routers, which are, roughly, real-time Chaum Mixes [3]. A Mix is a store-and-forward device that accepts a number of fixed-length messages from different sources, performs cryptographic transformations on the messages, and then forwards the messages to the next destination in an order not predictable from the order of inputs. A single Mix makes tracking of a particular message either by specific bit-pattern, size, or ordering with respect to other messages difficult. By routing through numerous Mixes in the network, determining who is talking to whom is made even more difficult. While Chaum's Mixes could store messages for an indefinite amount of time waiting to receive an adequate number of messages to mix together, a Core Onion Router (COR) is designed to pass information in real time, which limits mixing and potentially weakens the protection. Large volumes of traffic (some of it perhaps synthetic) can improve the protection of real time mixes.

Onion Routing can be used with applications that are proxy-aware, as well as several non-proxy-aware applications, without modification to the applications. Supported protocols include HTTP, FTP, SMTP, rlogin, telnet, NNTP, finger, whois, and raw sockets. Proxies have been designed but not development for Socks5, DNS, NFS, IRC, HTTPS, SSH, and Virtual Private Networks (VPNs).

The proxy incorporates three logical layers: an optional application specific privacy filter, an application specific translator that converts data streams into an application independent format of fixed length cells accepted by the Onion Routing (OR) network, and an onion management layer (the onion proxy) that builds and handles the anonymous connections. The onion proxy is the most trusted component in the system, because it knows the true source and destination of the connections that it builds and manages. To build onions and hence define routes the onion proxy must know the topology and link state of the network, the public certificates of nodes in the network, and the exit policies of nodes in the network.

Onion Routing's anonymous connections are protocol independent and exist in three phases: connection setup, data movement, and connection termination. Setup begins when the initiator creates an onion, which defines the path of the connection through the network. An onion is a (recursively) layered data structure that specifies properties of the connection at each point along the route, e.g., cryptographic control information such as the different symmetric cryptographic algorithms and keys used during the data movement phase. Each onion router along the route uses its private key to decrypt the entire onion that it receives. This operation exposes the cryptographic control information for this onion router, the identity of the next onion router in the path for this connection, and the embedded onion. The onion router pads the embedded onion to maintain a fixed size and sends it onward. The final onion router in the path connects to a responder proxy, which will forward data to the remote application.

After the connection is established, data can be sent in both directions. The initiator's onion proxy receives data from an application, breaks it into fixed size cells (128 bytes long, at present), and encrypts each cell multiple times – once for each onion router the connection traverses – using the algorithms and keys that were specified in the onion. As a cell of data moves through the anonymous connection, each onion router removes one layer of encryption, so the data emerges as plaintext from the final onion router in the path. The responder proxy regroups the plaintext cells into the data stream originally submitted by the application and forwards it to the destination. For data moving backward, from the recipient to the initiator, this process occurs in the reverse order, with the responder proxy breaking the traffic into cells, and successive onion routers encrypting it using (potentially) different algorithms and keys than the forward path. In this case the initiator's onion proxy decrypts the data multiple times, regroups the plaintext cells, and forwards them to the application.

Normally, either the application that initiates a connection or the destination server will terminate it. Since onion routers may fail, however, any onion router involved in a connection can cause that connection to be terminated. To an application (either at the initiating site or at the destination), such a failure looks the same as if the remote site had simply closed its TCP connection.

Longstanding TCP connections (called 'links' or 'thick pipes') between CORs define the topology of an OR network. Links are negotiated pairwise by CORs in the course of becoming neighbors. All traffic passing over a link is encrypted using stream ciphers negotiated by the pair of onion routers on that link. This cipher is added on top of the onion layers by the COR sending a cell across a link and stripped off again by the receiving COR. Since TCP guarantees sequential delivery, synchronization of the stream ciphers is not an issue. To support a new anonymous connection, an onion proxy creates a random route within the current OR network topology. The (fixed) size of an onion would limit a route to a maximum of 11 nodes in the current implementation. Because connections can be tunneled, however, arbitrarily long routes are possible, even though they will become impractical at some point because of the resulting network latencies.

An eavesdropper or a compromised onion router might try to trace packets based on their content or on the timing of their arrival and departure at a node. All data (onions, content, and network control) is sent through the Onion Routing network in uniform-sized cells (128 bytes). Because it is encrypted (or decrypted) as it traverses each node, a cell changes its appearance (but not its size) completely from input to output. This prevents an eavesdropper or a compromised onion router from following a packet based on its bit pattern as it moves across the Onion Routing network. In addition, all cells arriving at an onion router within a fixed time interval are collected and reordered randomly (i.e., "mixed") before they are sent to their next destinations, in order to prevent an eavesdropper from relating an outbound packet from a router with an earlier inbound one based on timing or sequence of arrival.

If traffic levels are low and requirements for real-time transmission are high, waiting for enough traffic to arrive so that mixing provides good hiding might cause unacceptable transmission delays. In this case, padding (synthetic traffic) can be added to the thick pipes. Conversely, an attacker might try to use a pulse of traffic to track cells flowing through the system. This attack can be made more difficult by imposing limits on the traffic flow over particular links, though this strategy can increase latency.

If a link between two CORs goes down or comes up, that information is propagated among the active CORs and proxies (again using the fixed cell size, and with the same protections as other OR traffic). This information permits proxies to build onions with feasible routes. Since routes are permitted to have loops of length greater than one hop, the number of active nodes does not limit the route length, as long as at least two nodes are active.

An onion router cannot tell the ultimate destination of traffic it forwards to another onion router. The Responder Proxy running on the last onion router in a path, however, can determine where traffic leaving the OR network is bound. Some operators of onion routers may wish to restrict the set of destinations (non onion-router destinations) to which their machines will forward traffic. For example, a commercial onion router might decide that it would forward traffic only to .com sites, or a government onion router might decide only to permit outgoing traffic destined for .gov sites. We call this an "exit policy," and have implemented software so that sites can define and enforce such policies. The onion proxy creating a path through the OR network needs information about exit policies, so that it doesn't create an infeasible route, and the second generation system provides this information. The use of this mechanism could of course warp the traffic flows through the network and might therefore permit some inferences about traffic flow. To counteract the ability of compromised CORs to lie about network topography, public keys, or exit policies, an external audit and verification system for this information has been built into every component. Without both mechanisms, however, we believe far fewer institutions would be willing to operate ORs, and the decreased level of participation could also reduce the effectiveness of our scheme.

The initial OR prototype, since it was not intended for wide deployment, took a number of short cuts. It enforced a fixed length (five hops) for all routes. It did not provide a method for maintaining topology information or communicating topology information among nodes. It did not provide padding or bandwidth limiting facilities. All of these mechanisms are included in the second generation system. To ease its widespread distribution, the second generation system does not include actual cryptographic software. Cryptographic functions are invoked via calls to Crypto APIs, and the operator must provide cryptographic libraries to implement those APIs.

3 Security Goals

Protection of communications against traffic analysis does not require support for anonymous communication. By encrypting data sent over a traffic-analysis-resistant connection, for example, endpoints may identify themselves to one another without revealing the existence of their communication to the rest of the network. However, traffic analysis is a potent tool for revealing parties in conversation, thereby compromising a communication that was intended to be anonymous. Thus, we consider goals for anonymous, as well as private, communication. In fact, the goals for these two cases differ very little; the distinction comes in the specification of the adversary.

There are various basic properties relating initiators, responders, and connections that we wish to protect. Pfitzmann and Waidner[22] have described sender and receiver anonymity as respectively hiding the identity of the sender or receiver of a particular message from an attacker, and unlinkability as a somewhat weaker property, preventing an attacker from linking the physical message sent by the sender with the physical message received by the recipient. In a similar vein, we define:

Sender activity: the mere fact that a sender is sending something.

Receiver activity: the mere fact that a receiver is receiving something.[1]

[1]It may be useful to distinguish principals that actually receive messages from those that are the target (intended receiver) of a message. For example, if a message is public-key encrypted for a principal and broadcast to this principal and 99 others, then barring transmission problems, all 100 received the message. However, only one was the intended destination of the message. In this paper, it is with the intended receiver that we are concerned.

Sender content: that the sender sent a particular content.

Receiver content: that the receiver received a particular content.

These are the basic protections with which we will be concerned. We will also be concerned with more abstract anonymity protection. For example, it may be far more revealing if these are compromised in combination. As one example, consider 50 people sending the message "I love you" to 50 other people, one each. We thus have sender and receiver activity as well as sender and receiver content revealed for all of these messages. However, without source and destination for each of these, we don't know who loves whom.

One of the combined properties that concerns us is:

Source-destination linking that a particular source is sending to a particular destination.[2]

This may or may not involve a particular message or transmission. Building on our previous example, suppose 50 people send 50 messages each to 50 other people (2500 messages total). Then, for any sender and receiver, we can say with certainty that they were linked on exactly one message; although we may not be able to say which one. For purposes of this paper we will be concerned with *connections*, specifically, the anonymity properties of the initiator and responder for a given connection.

4 Network Model

For purposes of this analysis, an Onion Routing network consists of onion proxies (or simply proxies), Core Onion Routers (CORs), links, over which CORs pass fixed length cells, and responder proxies, which reconstruct cells into the application layer data stream.

An attempt to analyze the traffic on a real onion routing network might try to take advantage of topological features, exit policies, outside information about communicants, and other details that we cannot hope to incorporate in a mathematical assessment of onion outing networks generally. We make a number of general and specific assumptions to permit us to proceed with the analysis. We also comment on the validity of these assumptions below.

Assumption 1. The network of onion routers is a clique (fully connected graph).

> Since links are simply TCP/IP connections traversing the Internet, a COR can maintain many such connections with relatively little overhead, and the second generation implementation allows a COR to have on the order of fifty thick pipe links to other CORs. Beyond that size, one is likely to find regions of highly connected nodes with multiple bridges between them. Assumption 1 thus seems reasonable for OR networks of up to 50 CORs.

Assumption 2. Links are all padded or bandwidth-limited to a constant rate.

> This simplification allows us to ignore passive eavesdroppers, since all an eavesdropper will see on any link is a constant flow of fixed length, encrypted cells. In fact, we expect that padding and limiting will be used to smooth rapid (and therefore potentially trackable) changes in link traffic rather than to maintain absolutely fixed traffic flows. Even if fluctuations could be observed, no principal remote from a link can identify his own traffic as it passes across that link, since each link is covered by the stream cipher under a key that the remote principal does not possess.

Assumption 3. The exit policy of any node is unrestricted.

> As noted in section 2, we expect that many CORs will conform to this assumption, but some may not. Restrictive exit policies, to the extent that they vary among CORs, could affect the validity of

[2] In [17], 'unlinkability' is limited to the case where a sender and receiver are both explicitly known to be active and targeted by an adversary; nonetheless they cannot be shown to be communicating with each other. Onion Routing does provide such unlinkability in some configurations, and depending on the adversary, but this is not a general goal for all connections.

Assumption 4, since the exit policy will limit the choice of the final node in a path. However, since in our adversary model the last COR may always be compromised, it makes no difference to the security of a connection given our other assumptions. Also note that this assumption is independent of whether or not the connection of some destinations to the final COR is hidden, e.g., by a firewall.

Assumption 4. For each route through the OR network each hop is chosen at random.

This assumption depends primarily on the route selection algorithm implemented by the onion proxy, and secondarily on conflicts between exit policies and connection requests. In practice, we expect this assumption to be quite good.

Assumption 5. The number of nodes in a route, n, is chosen from $2 \leq n < \infty$ based on repeated flips of a weighted coin.

Note that the expected route length is completely determined by the weighting of the coin. The length is extended by one for each flip until the coin-flip comes up on the terminate-route side—typically the more lightly weighted side. Thus, for example, if the coin is weighted so that the probability of extending the route is .8, then the expected route length is 5. Choosing route length by this means, as opposed to choosing randomly within some range was largely motivated by the Crowds design and the security analysis in [17], of which we will say more below.

Many configurations of Onion Routing components are possible, all yielding different kinds and degrees of assurance. [20] We will limit our analysis to the two configurations that we expect to be both the most common and the most widely used.

In the **remote-COR configuration**, the onion proxy is the only OR system component that runs on a machine trusted by the user. The first COR (and any infunnels) are running on a remote untrusted machine.

In the **local-COR configuration**, all components up to the first COR are running on locally trusted machines. This corresponds to a situation where a COR is running on an enclave firewall, and onions might be built at individual workstations or at the firewall depending on efficiencies, enclave policy, etc. It also corresponds to a situation where an individual with good connections to the Internet is running his own onion router to reduce the amount of information available to untrusted components. The important aspect of this connection is that the system from end application to the first COR is essentially a black box. Perhaps contrary to initial appearance, a PC using dial-up connection to a (trustworthy) ISP might naturally be considered to be in this configuration. This view is appropriate because any attacker residing entirely on the Internet is ordinarily excluded from the telephone dial-up connections running between the customer and the ISP.

We must also make assumptions about the entrance policy of sites. Since entrance policy is controlled by the proxy, it is natural to assume that anyone may connect using any protocol in the remote-COR configuration. In practice, CORs might only accept connections from specific principals (subscribers?); although the COR will be unable to determine, hence control, the application being run. In the local-COR configuration, the entrance policy is effectively to exclude all connections from outside the black box. (However, it still will forward connections from any other COR, and it is assumed to have an open exit policy.) These assumptions are then:

Assumption 6. Every COR is connected to the OR network and the outside via either the remote-COR configuration or the local-COR configuration, but not both.

Assumption 7. The entrance policy for entering the OR network via the remote-COR configuration is unrestricted.

Assumption 8. The entrance policy for entering the OR network via the local-COR configuration is to exclude all but internal connections.

Notice that these policy assumptions also determine the initiator being protected by use of the OR network. For the remote-COR configuration, it is the end application (via its proxy) that is being protected. For the local-COR configuration, it is the local COR that is effectively the initiator being protected. This conforms well with the possibility that a corporate or other enclave may wish to protect not just the activity of individual users of the network but that of the enclave as a whole. Likewise, an individual who is running his own COR would clearly want to protect connections emanating from that COR since he is the only possible initiator of those connections.

5 Adversary Model

One of the main challenges in designing anonymous communications protocols is defining the capabilities of the adversary. Given the tools at our disposal today, the adversary model essentially determines which salient characteristics the system should deploy in order to defeat her.

The basic adversaries we consider are:

Observer: can observe a connection (e.g., a sniffer on an Internet router), but cannot initiate connections.

Disrupter: can delay (indefinitely) or corrupt traffic on a link.

Hostile user: can initiate (destroy) connections with specific routes as well as varying the traffic on the connections it creates.

Compromised COR: can arbitrarily manipulate the connections under its control, as well as creating new connections (that pass through itself).

All feasible adversaries can be composed out of these basic adversaries. This includes combinations such as one or more compromised CORs cooperating with disrupters of links on which those CORs are not adjacent, or such as combinations of hostile outsiders and observers. However, we are able to restrict our analysis of adversaries to just one class, the compromised COR. We now justify this claim.

Especially in light of our assumption that the network forms a clique, a hostile outsider can perform a subset of the actions that a compromised COR can do. Also, while a compromised COR cannot disrupt or observe a link unless it is adjacent to it, any adversary that replaces some or all observers and/or disrupters with a compromised COR adjacent to the relevant link is more powerful than the adversary it replaces. And, in the presence of adequate link padding or bandwidth limiting even collaborating observers can gain no useful information about connections within the network. They may be able to gain information by observing connections to the network (in the remote-COR configuration), but again this is less than what the COR to which such connection is made can learn. Thus, by considering adversaries consisting of collections of compromised CORs we cover the worst case of all combinations of basic adversaries. Our analysis focuses on this most capable adversary, one or more compromised CORs.

The possible distributions of adversaries are

- **single adversary**
- **multiple adversary:** A fixed, randomly distributed subset of CORs is compromised.
- **roving adversary:** A fixed-bound size subset of CORs is compromised at any one time. At specific intervals, other CORs can become compromised or uncompromised.
- **global adversary:** All CORs are compromised.

Onion Routing provides no protection against a global adversary. If all the CORs are compromised, they can know exactly who is talking to whom. The content of what was sent will be revealed as it emerges from the OR network, unless it has been end-to-end encrypted outside the OR network. Even a firewall-to-firewall connection is exposed if, as assumed above, our goal is to hide which local-COR is talking to which local-COR.

6 Security Assessment

As discussed above, there are several possible adversary models. Having specifically ruled out the case of a global adversary, we now focus on the roving adversary model. (The remaining models are subsumed by it.) We begin the security assessment by defining some variables and features of the environment.

Recall that routes are of indeterminate length and that each route is a random walk from the route origin through the network.

We assume a closed system composed of a multitude of users and a set $\mathcal{S}$ of CORs. Let r be the total number of active CORs in the system, and—as mentioned in Section 4—let n be the (variable) length of a specific route $\mathcal{R} = \{R_1, ..., R_n\}$, where each R_j is a COR in the route $\mathcal{R}$. Routes are selected randomly (each route is a random walk from the Route origin through the network) and hops within a route are selected independently (except cycles of length one are forbidden).

Our roving adversary is characterized by c, the maximum number of CORs the adversary is able to corrupt within a fixed time interval (a round). At the end of each round, the adversary can choose to remain in place or shift some of its power to corrupt other CORs. In the latter case, previously-corrupted CORs are assumed to be instantly "healed", i.e., they resume normal, secure operation. $\mathcal{C}_i$ represents the set of CORs controlled by the adversary at round i ($\mathcal{C}_i \subset \mathcal{S}$). We note that this roving adversary model closely follows that found in the literature on proactive cryptography, e.g., [2, 13]. This is a standardly accepted model based on the view that system locations can be compromised periodically, but periodic security checks will detect compromises. Resulting responses as well as periodic system updates, etc. will return compromised components to normal.

At present, most connections through an Onion Routing network are likely to be for Web browsing or email. Given the short duration of typical Web and email connections, a static attack is all that can realistically be mounted; by the time a roving attacker has moved, the typical connection will have closed, leaving no trace amongst honest CORs. (This is in contrast to Crowds, cf. below.) Roving attacks are more likely to be effective against longer telnet or ftp connections.

We first analyze connections initiated from the remote-COR configuration and then connections initiated from the local-COR configuration. Within each case we consider short-lived and long-lived connections.

6.1 Assessment for Remote-COR Configuration

Given a route $\mathcal{R}$ as above, suppose that some but not all of the CORs in the route are compromised. There are three significant cases:

1. $R_1 \in \mathcal{C}_i$
   The first node is compromised. In this case sender activity is established by the adversary. Sender content is not lost since senders always pre-encrypt traffic. The probability of this event, assuming a random route and random node compromises, is $P_1 = c/r$.

2. $R_n \in \mathcal{C}_i$ The last node in the route is compromised. In this case receiver content as well as receiver activity are compromised. Sender content, sender activity, and source-destination linking remain protected. The probability of this event is $P_2 = c/r$.

3. R_1 and $R_n \in \mathcal{C}_i$ Both the first and last node in the path are compromised, so sender/receiver activity and receiver content are compromised. Moreover, the COR end-points are now able to correlate cell totals and compromise source-destination linking. Consequently, sender content can be simply inferred. The probability of this event is $P_3 = c^2/r^2$ (unless $n = 2$, in which case $P_3 = c(c-1)/r^2$ since self-looping is not allowed).

The adversary's goal must be to compromise the endpoints, since he gains little by controlling all intermediate (R_i for $1 < i < n$) CORs of a given route $\mathcal{R} = \{R_1, ..., R_n\}$. In the case of short-lived connections, the roving adversary has, in effect, only one round in which to compromise the connection, and succeeds in compromising all properties of a connection with probability c^2/r^2 (or $c(c-1)/r^2$).

We now consider the roving adversary against a long-lived connection.

	$R_1 \in \mathcal{C}_i$	$R_n \in \mathcal{C}_i$	R_1 and $R_n \in \mathcal{C}_i$
sender activity	Yes	No	Yes
receiver activity	No	Yes	Yes
sender content	No	No	Yes (inferred)
receiver content	No	Yes	Yes
source-destination linking	No	No	Yes
probability	c/r	c/r	c^2/r^2

Table 1: Properties of Attack Scenarios.

At route setup time, the probability that at least one COR on the route of length n is in $\mathcal{C}_1$ is given by:

$$1 - P(\mathcal{R} \cap \mathcal{C}_1 \ = \ \emptyset) \ = \ 1 - \frac{(r-c)^n}{r^n}$$

We now make (a perhaps optimistic) assumption that, if none of the CORs on the route are compromised at route setup time, then the adversary will not attempt the attack. In any case, for such an adversary the expected number of rounds must be at least one more than for an adversary that starts with at least one compromised COR on the route. Given at least one compromised COR on the route, how many rounds does it take for the adversary to achieve source-destination linking?

In general, the attack starts when one of the subverted CORs receives a route setup request. She then proceeds to attempt the discovery of the route's true endpoints.

In either case, at round 1, the adversary can establish:

$$R_s \text{ where } s = min(j \in [1..n] \text{ and } R_j \in \mathcal{R} \cap \mathcal{C}_1)$$

as well as:

$$R_e \text{ where } e = max(j \in [1..n] \text{ and } R_j \in \mathcal{R} \cap \mathcal{C}_1)$$

While the actual indices e and s are not known to the adversary, she can identify R_s and R_e by timing the propagation of route setup. Moreover, the adversary can trivially test if $R_e = R_1$ or $R_s = R_n$.

The adversary's subsequent optimal strategy is illustrated in Figure 2. As shown in the pseudocode, the game played by the adversary is two-pronged:

1. In each round she moves one hop closer towards route endpoints (moving to the preceding hop of R_s and next hop of R_e.)
2. She also randomly picks a set of (at least $c-2$) routers to subvert from among the uncorrupted set (which is constantly updated). When one of the end-points is reached, the adversary can concentrate on the other, thus having $c-1$ routers to corrupt (at random) at each round.

In the worst case, it takes the adversary $MAX(s, n-e)$ rounds to reach both endpoints. More generally, the greatest value of $MAX(s, n-e)$ is n. (For example, a single roving adversary always takes exactly n rounds to compromise source and destination.)

An interesting open issue is the expected number of rounds a ($c \geq 2$)-adversary needs in order to reach the route endpoints provided that she starts out with at least one compromised COR on the route. In lieu of an analytic solution, it might be interesting to run this for sample sets of test configurations of nodes and adversary nodes for various possible (sets) of connections. We leave this for future work.

6.2 Assessment For Local-COR Configuration

The local-COR configuration is distinguished from the remote-COR configuration by the fact that the first COR in the connection is assumed to be immune from compromise. In the remote-COR configuration, compromising the first COR is the only way to compromise sender properties or source-destination linking.

```
/*  assume at least one router initially compromised */
/*  assume t>=2 */
/*  HEALTHY_ROUTERS is the set of hereto uncorrupted routers */
/*  remove_from_set() returns set minus element to be removed; */
/*     if an element not in set, return set unchanged */
/*  compute_max_span() returns R_s and R_e, the compromised routers
*      farthest apart on the route;
R_1_found = R_n_found = false;
available_random = c-2;
HEALTHY_ROUTERS = ALL_ROUTERS;
while ( (! R_1_found) && (! R_n_found))
{
    /* identify first, last subverted routers */
    compute_max_span(R_s,R_e);
    /* note that it's possible that R_s=R_e */
    if ( R_s==R_1 )
    {   R_1_found = true;
        available_random ++;
    }
    if ( R_e==R_n )
    {   R_n_found = true;
        available_random ++;
    }
    R_s = prev_hop (R_s);
    R_e = next_hop (R_e);
    subvert (R_s);
    remove_from_set(HEALTHY_ROUTERS, R_s);
    subvert (R_e);
    remove_from_set(HEALTHY_ROUTERS, R_e);
    /* subvert a set of random routers */
    for (i=0; i<available_random; i++)
    {   j = random_router_index(HEALTHY_ROUTERS);
        subvert (R_j);
        remove_from_set(HEALTHY_ROUTERS, R_j);
    }
}
```

Figure 2: Pseudo-code for the adversary's game.

In the local-COR configuration, the only way that source-destination linking or any sender properties can be compromised is if the adversary can somehow infer that the first COR is in fact first. There is no way for this to happen within our threat model unless all other CORs are compromised. (If all CORs connected to the local-COR were compromised they could infer that this was the first COR, but since we have assumed a clique of CORs, this would imply that the local-COR is the only uncompromised COR in the onion routing network.)

There is a way that the first COR could be identified as such that is outside of our described threat model: if the second COR is compromised, and *if* it is possible to predict that some data cells will produce an immediate response from the initiator, then the second COR may be able to infer that the first COR is first by the response time. This possibility is less remote if the last COR is also compromised and we assume that the data sent over it is not end-to-end encrypted for the responder. We will return to this discussion below.

7 Related Work

Basic comparison of Onion Routing to broadly related anonymity mechanisms, such as remailers [11, 5] and ISDN-Mixes [14] can be found in [16]. Also mentioned there are such complementary connection-based mechanisms as LPWA [7] and the Anonymizer [1]. These are both very effective at anonymizing the data stream in different ways, but they both pass all traffic directly from the initiator via a single filtering point to the responder. There is thus minimal protection for the anonymity of the connection itself, which is our primary focus. We therefore restrict our comments to related work that is directed to wide-spread Internet communication either below the application layer or specifically for some form of connection based traffic. For this reason, direct comparisons to local anonymity systems such as TG and SS [12] are omitted due to their small deployable size and tight timing constraints.

We know of only one other published proposal for application-independent, traffic-analysis-resistant Internet communication, viz: that of [6], wherein a system is described that effectively builds an onion for every IP packet. There is thus no connection, either in the sense of a TCP/IP socket, or more significantly, in the sense of a path of nodes that can perform fast (symmetric) encryption on passing traffic. In Onion Routing, computationally expensive public-key cryptography is used only during connection setup. Using an onion for every IP packet makes for real-time capabilities significantly slower than those of Onion Routing and thus less applicable to such things as telnet connections or even Web traffic—if loading pages is expected to be anywhere close to current speeds. On the other hand, for applications that do not have these requirements, such a design may offer better security since there is no recurrent path for packets in an (application) connection.

A commercial system that appears to be quite similar to Onion Routing is being built by Zero Knowledge Systems (www.freedom.net). Like Onion Routing, it establishes a connection in the form of a path of routers that have been keyed for subsequent data passing; however, its underlying transmission is based on UDP rather than TCP/IP.

Given the above, our remaining comparative comments will be with respect to Crowds. We begin with a brief comparative description. The first thing to note is that Crowds is designed exclusively for Web traffic. Interestingly, though Crowds is only to be used for (short-lived) Web connections, it make use of longstanding cryptographic paths. Once a path is established from an initiator through a Crowd, all subsequent HTTP connections are passed through that path. (The tail of a path is randomly regenerated only beyond a break point and only when broken. Whole paths are regenerated only when new members are brought into the Crowd.) This means that a path initiator is less likely to be identified than would be the case if a new path were built for each connection [17]. This is especially important because, unlike in Onion Routing, a compromised node knows content and destination of all connections passing through it (see below). This means that adversaries have a better means to build likely profiles of repeated connections by the same initiator. The static paths of Crowds makes such profile information less useful by making it harder to know which Crowds member is profiled. If a new path were built for each connection, compromised nodes would have a better chance of intersecting predecessors with the same profile and thus identifying the initiator. On the other hand, known (rather than inferred) path profiles in that case would be much less complete, i.e., forward anonymity (in the sense of [21]) is worse for static paths. Put another way, in the remote-COR configuration, assuming a fixed distributed adversary, the likelihood that some connection one makes will have a compromised first and last node increases over the number of connections made (if the first COR is chosen different each time). However, the compromise pertains only to the current connection. If paths are static, compromise is ongoing.

Encrypted traffic looks the same as it passes along a path through a Crowd. And, the decryption key is available to all path participants. Thus, any compromised node on the path compromises both receiver activity and receiver content. Also, link padding is not a system option. As a result, a local eavesdropper (observing links from the initiator) and any one compromised member of a path completely compromise all properties mentioned in section 3. A local eavesdropper in the remote-COR configuration of Onion Routing compromises sender activity; however, unless the last COR in the connection is also compromised, nothing else is revealed. On the other hand, in this configuration, the first untrusted component of the system is able to compromise sender activity. And, a local eavesdropper together with a compromised last COR compromise all properties in section 3. For Crowds without a local eavesdropper, the first untrusted component of the system (i.e., the next node on the path) cannot identify the initiator with certainty, in fact with any more likelihood than that dictated by the probability of forwarding (i.e., the probability of extending the path vs. connecting to the responder).

In the local-COR configuration Onion Routing provides similar protections to Crowds of sender activity, source-destination linking, and sender content, and much better protection against receiver activity or receiver content, with the adversary model we have set out above and without a local eavesdropper. If we add to the adversary model, things get more complicated.

As noted in section 6.2, if an adversary that has compromised the second COR in a route can predict which data will prompt an immediate response from the initiator (e.g., if the final COR is also compromised), then she may be able to time responses to determine that there can be at most one COR prior in the route. This sort of problem was recognized early on in the design of Crowds. It is a more serious problem for Crowds because the second node alone can read requests and responses, making it easy to find the timing-relevant data. In Crowd, if URLs are included in data coming from a responder that will prompt a subsequent request from the initiator, nodes later on the path will themselves parse the HTML and make these requests back in the direction of the responder, thus eliminating the timing distinction between the first node and any others.

Timing information is thus obscured by having the nodes on the path actively processing and filtering data as well as managing the connection. This is important because it means that all nodes must be able to read all traffic so that the data stream must be anonymous. Therefore, unlike Onion Routing, Crowds inherently cannot be used in circumstances where one would like to identify (and possibly authenticate) oneself to the far end but would like to hide from others with whom one is communicating. More importantly, as ever more functionality is added to Web browsers, and since nodes on a Crowds route must be able to read and alter all traffic on a connection, a single compromised node can embed requests, either for identifying information or to open connections directly back to it—bypassing the crowd and identifying the originator. Obvious means of attack such as by use of cookies, or by Java, Javascript, etc. are easily shut off or filtered (along with their provided functionality). However, other more subtle mechanisms are also available (cf. www.onion-router.net/Tests.html for a list of test sites, in particular www.onion-router.net/dynamic/snoop), and more are becoming available all the time. The upshot of all this is that, for Crowds, the anonymity of the connection is always at best as good as the latest installed filtering code for anonymity of the data stream. For Onion Routing, these two functions—anonymity of the connection and anonymity of the data stream—are separate, and a new means to identify the initiator via the data stream only affects the anonymity of the data stream. Even then, it is only employable at the far end of a connection, rather than by any node on an anonymized connection.

The last point of comparison we discuss is performance. Direct comparison in practice is nearly impossible. Each onion routing connection requires several public-key decryptions. So, with respect to cryptographic overhead Crowds has much better performance potential. On the other hand, within an onion routing network, connections are expected to be longstanding high-capacity channels between dedicated machines, possibly with cryptographic coprocessors. The major performance limitation is often likely to be the end user's Internet connection. But, for Crowds the end users are the network. Thus, Crowds members with slow or intermittent connections will affect the performance of everyone in their crowd. One can limit Crowds participation to those with longstanding, high-speed (say T1 or better) Internet connections. But, this will seriously limit the population for whom it is feasible. Depending on the user population, network configuration, and the components that make up the network, one is likely to find very different performance numbers in each of the systems.

8 Conclusions and Future Work

We have presented some of the features of the current Onion Routing design and analyzed its resistance to worst-case adversaries. This design generally resists traffic analysis more effectively than any other published and deployed mechanisms for Internet communication.

We note some ways that the design might be changed to improve security: Adding a time delay to traffic at the proxy could complicate timing attacks against the local-COR configuration to determine the first COR. (Similarly, if the last COR is local to the responder, in the sense of this paper, then it would be possible to add a time delay at the responder proxy.) Of course, this is only necessary when the goal is actually to protect the local COR, for example to protect the activity of an enclave or if the COR is run by one or a few individuals who are the only ones accessing/exiting the onion routing network through that COR. Suppose a typical customer-ISP configuration, in which the initiator is someone connecting through dial-up to an ISP running an onion router. As noted above in Section 4, this could be viewed as a local-COR configuration. But, in this case, it is the anonymity of the individual source rather than the COR that matters. Thus, no delay is necessary. (One could address a semi-trusted local-COR by building onions at the workstation for a COR, e.g., at an ISP or an enclave firewall. Such options are discussed in [20].)

Finally, if partial-route padding is used on individual connections, besides link padding, then compromise by even internal attackers is complicated. For example, a local eavesdropper or compromised first COR (in the remote-COR configuration) would not be able to easily cooperate with a compromised last COR to break source-destination linking. In fact, the second generation design has been made consistent with the possibility that onion proxies can choose to do this via in-channel signaling to intermediate CORs if they so desire. Also, long-lived application connections could be hopped between shorter-lived Onion Routing connections using specialized proxies. This would both frustrate a roving attacker, and make such connections look more like short-lived connections even to network insiders. We have discussed some of the features such proxies might have, but such proxies have not yet been designed.

Acknowledgments

This work supported by ONR and DARPA. Jim Proto and Jeremy Barrett have done much of the coding of the second generation system, and Mark Levin did much of its design specification. Lora Kassab has investigated route selection algorithms, and her work has influenced the above discussion. Thanks to Mike Reiter and Avi Rubin for helpful discussions about the goals of Crowds. Thanks to Ira Moskowitz, Cathy Meadows, and LiWu Chang for other helpful discussions.

References

[1] The Anonymizer. `http://www.anonymizer.com`

[2] R. Canetti and A. Herzberg. "Maintaining Security in the Presence of Transient Faults", *Advances in Cryptology—CRYPTO'94*, LNCS vol. 839, Springer-Verlag, 1994, pp. 425–438.

[3] D. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", *Communications of the ACM*, vol. 24, no. 2, Feb. 1981, pages 84-88.

[4] D. Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability", *Journal of Cryptology*, vol. 1, no. 1, 1988, pages 65-75.

[5] L. Cottrell. *Mixmaster and Remailer Attacks,* http://obscura.obscura.com/~loki/remailer/remailer-essay.html

[6] A. Fasbender, D. Kesdogan, O. Kubitz. "Variable and Scalable Security: Protection of Location Information in Mobile IP", 46^{th} *IEEE Vehicular Technology Society Conference*, Atlanta, March 1996.

[7] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer. "How to Make Personalized Web Browsing Simple, Secure, and Anonymous", in *Financial Cryptography: FC '97, Proceedings*, R. Hirschfeld (ed.), Springer-Verlag, LNCS vol. 1318, pp. 17–31, 1998.

[8] I. Goldberg and D. Wagner. "TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web", *First Monday*, vol. 3 no. 4, April 1998.

[9] D. Goldschlag, M. Reed, P. Syverson. "Hiding Routing Information", in *Information Hiding*, R. Anderson, ed., LNCS vol. 1174, Springer-Verlag, 1996, pp. 137–150.

[10] D. Goldschlag, M. Reed, P. Syverson. "Onion Routing for Anonymous and Private Internet Connections," *Communications of the ACM*, vol. 42, num. 2, February 1999.

[11] C. Gülcü and G. Tsudik. "Mixing Email with *Babel*", in *1996 Symposium on Network and Distributed System Security*, San Diego, February 1996.

[12] D. Martin Jr., "Local Anonymity in the Internet", Ph.D. Dissertation, Boston University, 1999.

[13] R. Ostrovsky and M. Yung. "How to Withstand Mobile Virus Attacks", in *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing (PODC '91)*, ACM Press, 1991, pp. 51–59.

[14] A. Pfitzmann, B. Pfitzmann, and M. Waidner. "ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead", *GI/ITG Conference: Communication in Distributed Systems*, Mannheim Feb, 1991, Informatik-Fachberichte 267, Springer-Verlag, Heidelberg 1991, pp. 451-463.

[15] M. Reed, P. Syverson, and D. Goldschlag. "Protocols using Anonymous Connections: Mobile Applications", in *Security Protocols: Fifth International Workshop*, B. Christianson, B. Crispo, M. Lomas, and M. Roe, eds., LNCS vol. 1361, Springer-Verlag, 1997, pp. 13–23.

[16] M. Reed, P. Syverson, and D. Goldschlag. "Anonymous Connections and Onion Routing", *IEEE Journal on Selected Areas in Communications*, vol. 16 no. 4, May 1998, pp. 482–494. (A preliminary version of this paper appeared in [19].)

[17] M. Reiter and A. Rubin. "Crowds: Anonymity for Web Transactions", *ACM Transactions on Information System Security*, vol. 1, no. 1, November 1998, pp. 66–92. (A preliminary version of this paper appeared in [18].)

[18] M. Reiter and A. Rubin. *Crowds: Anonymity for Web Transactions,* DIMACS Technical Reports 97-15, April 1997 (Revised August 1997).

[19] P. Syverson, D. Goldschlag, and M. Reed. "Anonymous Connections and Onion Routing", in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, IEEE CS Press, May 1997, pp. 44–54.

[20] P. Syverson, M. Reed, and D. Goldschlag. "Onion Routing Access Configurations", in *DISCEX 2000: Proceedings of the DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, IEEE CS Press, January 2000, pp. 34–40.

[21] P. Syverson, S. Stubblebine, and D. Goldschlag, "Unlinkable Serial Transactions", in *Financial Cryptography: FC '97, Proceedings*, R. Hirschfeld (ed.), Springer-Verlag, LNCS vol. 1318, pp. 39–55, 1998.

[22] A. Pfitzmann and M. Waidner, "Networks without User Observability", *Computers & Security*, vol. 6 (1987), pp. 158–166.