Anonymous Connections and Onion Routing

Michael G. Reed, *Member, IEEE*, Paul F. Syverson, and David M. Goldschlag

***Abstract*—Onion routing is an infrastructure for private communication over a public network. It provides anonymous connections that are strongly resistant to both eavesdropping and traffic analysis. Onion routing's anonymous connections are bidirectional, near real-time, and can be used anywhere a socket connection can be used. Any identifying information must be in the data stream carried over an anonymous connection. An onion is a data structure that is treated as the destination address by onion routers; thus, it is used to establish an anonymous connection. Onions themselves appear different to each onion router as well as to network observers. The same goes for data carried over the connections they establish. Proxy-aware applications, such as web browsers and e-mail clients, require no modification to use onion routing, and do so through a series of proxies. A prototype onion routing network is running between our lab and other sites. This paper describes anonymous connections and their implementation using onion routing. This paper also describes several application proxies for onion routing, as well as configurations of onion routing networks.**

***Index Terms*—Anonymity, communications, Internet, privacy, security, traffic analysis.**

I. Introduction

IS INTERNET communication private? Most security concerns focus on preventing eavesdropping,[1] i.e., outsiders listening in on electronic conversations. But encrypted messages can still be tracked, revealing who is talking to whom. This tracking is called traffic analysis and may reveal sensitive information. For example, the existence of inter-company collaboration may be confidential. Similarly, e-mail users may not wish to reveal who they are communicating with to the rest of the world. In certain cases anonymity may be desirable, for example, anonymous e-cash is not very anonymous if delivered with a return address. Web-based shopping or browsing of public databases should not require revealing one's identity.

This paper describes how a freely available system, *onion routing*, can be used to protect a variety of Internet services against both eavesdropping and traffic analysis attacks from both the network and outside observers. This paper includes a specification sufficient to guide both re-implementations and new applications of onion routing. We also discuss configurations of onion routing networks and applications of onion routing, including virtual private networks (VPN's), Web browsing, e-mail, remote login, and electronic cash.[2]

One purpose of traffic analysis is to reveal who is talking to whom. The *anonymous connections* described here are designed to be resistant to traffic analysis, i.e., to make it difficult for observers to learn identifying information from the connection (e.g., by reading packet headers, tracking encrypted payloads, etc.). Any identifying information must be passed as data through the anonymous connections. Our implementation of anonymous connections, onion routing, provides protection against eavesdropping as a side effect. Onion routing provides bidirectional and near real-time communication similar to TCP/IP socket connections or ATM AAL5 [6]. The anonymous connections can substitute for sockets in a wide variety of unmodified Internet applications by means of proxies. Data may also be passed through a privacy filter before being sent over an anonymous connection. This removes identifying information from the data stream, to make communication anonymous too.

Although onion routing may be used for anonymous communication, it differs from anonymous remailers [7], [15] in two ways: communication is real-time and bidirectional, and the anonymous connections are application independent. Onion routing's anonymous connections can support anonymous mail as well as other applications. For example, onion routing may be used for anonymous Web browsing. A user may wish to browse public Web sites without revealing his identity to those Web sites. That requires removing information that identifies him from his requests to Web servers and removing information from the connection itself that may identify him. Hence, anonymous Web browsing uses anonymized communication over anonymous connections. The Anonymizer [1] only anonymizes the data stream, not the connection itself. So it does not prevent traffic analysis attacks like tracking data as it moves through the network.

This paper is organized in the following way: Section II presents an overview of onion routing. Section III presents empirical data about our prototype. Section IV defines our threat model. Section V describes onion routing and the application specific proxies in more detail. Section VI describes the implementation choices that were made for security reasons. Section VII describes how onion routing may be used in a wide variety of Internet applications. Section VIII contrasts onion routing with related work, and Section IX presents concluding remarks.

Manuscript received March, 1997; revised September, 1997. This work was supported by the Office of Naval Research and by the Defense Advanced Research Projects Agency.

M. G. Reed and P. F. Syverson are with the Naval Research Laboratory, Center For High Assurance Computer Systems, Washington, D.C. 20375-5337 USA (e-mail: reed@itd.nrl.navy.mil; syverson@itd.nrl.navy.mil).

D. M. Goldschlag is with Divx, Herndon, VA 20170 USA (e-mail: david.goldschlag@divx.com).

Publisher Item Identifier S 0733-8716(98)01110-X.

[1] Internet Engineering Task Force. HTTP://www. ietf. org.

[2] Preliminary versions of portions of this paper have appeared in [23], [13], [19], and [14].

II. Onion Routing Overview

In onion routing, instead of making socket connections directly to a responding machine, initiating applications make connections through a sequence of machines called *onion routers*. The *onion routing network* allows the connection between the *initiator* and *responder* to remain anonymous. Anonymous connections hide who is connected to whom, and for what purpose, from both outside eavesdroppers and compromised onion routers. If the initiator also wants to remain anonymous to the responder, then all identifying information must be removed from the data stream before being sent over the anonymous connection.

Onion routers in the network are connected by longstanding (permanent) socket connections. Anonymous connections through the network are multiplexed over the longstanding connections. For any anonymous connection, the sequence of onion routers in a route is strictly defined at connection setup. However, each onion router can only identify the previous and next hop along a route. Data passed along the anonymous connection appear different at each onion router, so data cannot be tracked en route, and compromised onion routers cannot cooperate by correlating the data stream each sees. We will also see that they cannot make use of replayed onions or replayed data.

A. Operational Overview

The onion routing network is accessed via a series of *proxies*. An initiating application makes a socket connection to an *application proxy*. This proxy massages connection message format (and later data) to a generic form that can be passed through the onion routing network. It then connects to an *onion proxy*, which defines a route through the onion routing network by constructing a layered data structure called an *onion*. The onion is passed to the *entry funnel*, that occupies one of the longstanding connections to an onion router and multiplexes connections to the onion routing network at that onion router. That onion router will be the one for whom the outermost layer of the onion is intended. Each layer of the onion defines the next hop in a route. An onion router that receives an onion peels off its layer, identifies the next hop, and sends the embedded onion to that onion router. The last onion router forward data to an *exit funnel*, whose job is to pass data between the onion routing network and the responder.

In addition to carrying next-hop information, each onion layer contains key seed material from which keys are generated for crypting[3] data sent forward or backward along the anonymous connection. (We define *forward* to be the direction in which the onion travels and *backward* as the opposite direction.)

Once the anonymous connection is established, it can carry data. Before sending data over an anonymous connection, the onion proxy adds a layer of encryption for each onion router in the route. As data move through the anonymous connection, each onion router removes one layer of encryption, so it arrives at the responder as plaintext. This layering occurs in the reverse order for data moving back to the initiator. Therefore data that have passed backward through the anonymous connection must be repeatedly post-crypted to obtain the plaintext.

By layering cryptographic operations in this way, we gain an advantage over link encryption. As data move through the network it appears different to each onion router. Therefore, an anonymous connection is as strong as its strongest link, and even one honest node is enough to maintain the privacy of the route. In link encrypted systems, compromised nodes can trivially cooperate to uncover route information.

Onion routers keep track of received onions until they expire. Replayed or expired onions are not forwarded, so they cannot be used to uncover route information, either by outsiders or compromised onion routers. Note that clock skew between onion routers can only cause an onion router to reject a fresh onion or to keep track of processed onions longer than necessary. Also, since data are encrypted using stream ciphers, replayed data will look different each time it passes through a properly operating onion router.

Although we call this system onion routing, the routing that occurs here does so at the application layer of the protocol stack and not at the IP layer. More specifically, we rely upon IP routing to route data passed through the longstanding socket connections. An anonymous connection is comprised of portions of several linked longstanding multiplexed socket connections. Therefore, although the series of onion routers in an anonymous connection is fixed for the lifetime of that anonymous connection, the route that data actually travels between individual onion routers is determined by the underlying IP network. Thus, onion routing may be compared to loose source routing.

Onion routing depends upon connection-based services that deliver data uncorrupted and in order. This simplifies the specification of the system. TCP socket connections, which are layered on top of a connectionless service like IP, provide these guarantees. Similarly, onion routing could easily be layered on top of other connection based services, like ATM AAL5.

Our current prototype of onion routing considers the network topology to be static and does not have mechanisms to automatically distribute or update public keys or network topology. These issues, though important, are not the key parts of onion routing and will be addressed in a later prototype.

B. Configurations

As mentioned above, neighboring onion routers are neighbors in virtue of having longstanding socket connections between them, and the network as a whole is accessed from the outside through a series of proxies. By adjusting where those proxies reside it is possible to vary which elements of the system are trusted by users and in what way. (For some configurations it may be efficient to combine proxies that reside in the same place, thus they may be only conceptually distinct.)

1) Firewall Configuration: In the *firewall configuration*, an onion router sits on the firewall of a sensitive site. This onion firewall and the external network. Connections from machines behind the firewall to the onion router are protected by other means (e.g., physical security). To complicate tracking of traffic originating or terminating within the sensitive site, this onion router should also route data between other onion routers. This configuration might represent the system interface from a typical corporate or government site. Here the application proxies (together with any privacy filters) and the onion proxies would typically live at the firewall as well. (Typically, there might only be one onion proxy.)

There are three important features of this basic configuration.

- Connections between machines behind onion routers are protected against both eavesdropping and traffic analysis. Since the data stream never appears in the clear on the public network, this data may carry identifying information but communication is still private. (This feature is used in Section VII-A.)
- The onion router at the originally protected site knows both the source and destination of a connection. This protects the anonymity of connections from observers outside the firewall but also simplifies enforcement of and monitoring for compliance with corporate or governmental usage policy.
- The use of anonymous connections between two sensitive sites that both control onion routers effectively hides their communication from outsiders. However, if the responder is not in a sensitive site (e.g., the responder is some arbitrary Web server), the data stream from the sensitive initiator must also be anonymized. If the connection between the exit funnel and the responding server is unencrypted, the data stream might otherwise identify the initiator. For example, an attacker could simply listen in on the connections to a Web server and identify initiators of any connection to it.

2) Remote Proxy Configuration: What happens if an initiator does not control an onion router? If the initiator can make encrypted connections to some remote onion router, then he can function as if he is in the firewall configuration just described, except that both observers and the network can tell when he makes connections to the onion router. However, if the initiator trusts the onion router to build onions, his association with the anonymous connection from that onion router to the responder is hidden from observers and the network. In a similar way, an encrypted connection from an exit funnel to a responder hides the association of the responder with the anonymous connection.

Therefore, if an initiator makes an anonymous connection to some responder, and layers end-to-end encryption over that anonymous connection, the initiator and responder can identify themselves to one another, yet hide their communication from the rest of the world.

Notice, however, that the initiator trusts the remote onion router to conceal that the initiator wants to communicate with the responder and to build an anonymous connection through other onion routers. The next section describes how to shift

3) The Customer-ISP Configuration: Suppose, for example, an Internet Services Provider (ISP) runs a funnel that accepts connections from onion proxies running on subscribers' machines. In this configuration, users generate onions specifying a path through the ISP to the destination. Although the ISP would know who initiates the connection, the ISP would not know with whom the customer is communicating, nor would it be able to see data content. So the customer need not trust the ISP to maintain her privacy. Furthermore, the ISP becomes a *common carrier* that carries data for its customers. This may relieve the ISP of responsibility for both whom users are communicating with and the content of those conversations. The ISP may or may not be running an onion router as well. If he is running an onion router, then it is more difficult to identify connections that terminate with his customers; however, he is serving as a routing point for other traffic. On the other hand, if he simply runs a funnel to an onion router elsewhere, it will be possible to identify connections terminating with him, but his overall traffic load will be less. Which of these would be the case for a given ISP would probably depend on a variety of service, cost, and pricing considerations. Note that in this configuration, the entry funnel must have an established longstanding connection to an onion router just like any neighboring onion router (cf. Section V-F for a description of how these are established). But, in most other cases, where the funnel resides on the same machine as the onion router, establishing an encrypted longstanding connection should not be necessary because the funnel can be directly incorporated into the onion router.

III. Empirical Data

We invite readers to experiment with our prototype of onion routing by using it to anonymously surf the Web, send anonymous e-mail, and do remote logins. For instructions, please see `http://www.onion.router.net/` .

One should be aware that accessing a remote onion router does not completely preserve anonymity because the connection between a remote machine and the first onion router is not protected. If that connection were protected, one would be in the remote proxy configuration, but there would still be no reason to trust the remote onion router. If one had a secured connection to an onion router one trusted, our onion router could be used as one of several intermediate routers to further complicate traffic analysis.

We have recently set up a 13-node distributed network of government, academic, and private sites. However, at press time we have not yet gathered performance data for this network. The data we present are for a network running on a single machine. In our experimental onion routing network, five onion routers run on a single Sun Ultra 2 2170. This machine has two 167-MHz processors and 256 MB of memory. Anonymous connections are routed through a random sequence of five onion routers. Connection setup time should be comparable to a more distributed topology. Data latency, however, is more difficult to judge. Clearly, data will travel faster over socket connections between onion routers on the machines. However, on a single machine the removal or addition of layers of encryption is not pipelined, so data latency may be worse.

Onion routing's overhead is mainly due to public key cryptography and is incurred while setting up an anonymous connection. On our Ultra 2, running a fast implementation of RSA [2], a single public key decryption of a 1024-b plaintext block using a 1024-b private key and a 1024-b modulus, takes 90 ms. Encryption is much faster, because the public keys are only 16 b long. (This is why RSA signature verification is cheaper than signing.) So, the public key cryptographic overhead for routes spanning five onion routers is just under 0.5 s. This overhead can be further reduced, either with specialized hardware, or even simply on different hardware (a 200-MHz Pentium would be almost twice as fast).

In practice, our connection setup overhead does not appear to add intolerably to the overhead of typical socket connections. Still, it can be further reduced. There is no reason that the same anonymous connection could not be used to carry the traffic for several "real" socket connections, either sequentially or multiplexed. In fact, the specification for HTTP 1.1 defines pipelined connections to amortize the cost of socket setup, and pipelined connections would also transparently amortize the increased cost of anonymous connection setup. We are currently updating our Web proxy to be HTTP 1.1 compliant.

IV. Threat Model

This section outlines our threat model. It does not intend to quantify the cost of attacks, but to define possible attacks. Future work will quantify the threat. First, some vocabulary. A session is the data carried over a single anonymous connection. Data are carried in fixed length cells. Because these cells are multiply encrypted and change as they move through an anonymous connection, tracking cells is equivalent to tracking markers that indicate when cells begin. In a marker attack, the attacker identifies the set of outbound connections that some distinguished marker may have been forwarded upon. By intersecting these sets for a series of distinguished markers belonging to the same session, an attacker may determine, or at least narrow, the set of possible next hops. In a timing attack, the attacker records a timing signature for a session that correlates data rate over time. A session may have a very similar timing signature wherever it is measured over a route, so cooperating attackers may determine if they carry a particular session.

We assume that the network is subject to both passive and active attacks. Traffic may be monitored and modified by both external observers and internal network elements, including compromised onion routers. Attackers may cooperate and share information and inferences. We assume roving attackers that can monitor part, but not all, of the network at a time.

Our goal is to prevent traffic analysis, not traffic confirmation. If an attacker wants to confirm that two endpoints often communicate, and he observes that they each connect to an anonymous connection at roughly the same time, more often than is statistically expected, it is reasonable to infer that the

is infeasible if endpoints live in protected networks behind trusted onion routers on firewalls.

If the onion routing infrastructure is uniformly busy, then passive external attacks are ineffective. Specifically, neither the marker nor timing attacks are feasible, since external observers cannot assign markers to sessions. Active attacks are possible, because reducing the load on the system makes the network easier to analyze (and makes the system not uniformly busy).

Passive internal attacks require at least two compromised onion routers. Since onion routers can assign markers to a session, both the marker and timing attacks are possible. Specifically, timing signatures can be broadcast, and other compromised onion routers can attempt to find connections with matching timing signatures.

Another attack that is only feasible as an internal attack is the volume attack. Compromised onion routers can keep track of the number of cells that have passed over any given anonymous connection. They can then simply broadcast totals to other compromised onion routers. Cell totals that are close to the same amount at the same time at different onion routers are likely to belong to the same anonymous connection.[4]

Active internal attacks amplify these risks, since individual onion routers can selectively limit traffic on particular connections. An onion router, for example, could force a particular timing signature on a connection and advertise that signature.

V. Onion Routing Specifics

A. *Onion Routing Proxies*

A proxy is a transparent service between two applications that would usually make a direct socket connection to each other but cannot. For example, a firewall might prevent direct socket connections between internal and external machines. A proxy running on the firewall may enable such connections. Proxy-aware applications are becoming quite common.

Our goal has been to design an architecture for private communication that would interface with *unmodified* applications, so we chose to use proxies as the interface between applications and onion routing's anonymous connections. For applications that are designed to be proxy aware (e.g., WWW browsers), we simply design appropriate interface proxies. Surprisingly, for certain applications that are not proxy aware (e.g., RLOGIN), we have also been able to design interface proxies.

Because it is necessary to bridge between applications and the onion routing network, proxies must understand both application protocols and onion routing protocols. Therefore, we modularize the design into components: the application proxy, the onion proxy, and the entry funnel. The application proxy bridges between a socket connection from an application and a socket connection to the onion proxy. It is the obligation of the application proxy to massage the data stream so the onion proxy, the entry funnel, and the exit funnel can be application independent. Specifically, the application proxy must prepend to the data stream a *standard structure* that identifies the ultimate destination by either hostname/port or IP address/port. Additionally, it must process a 1-byte return code from the exit funnel and either continue if no error is reported or report the onion routing error code in some application specific meaningful way. The application proxy may also contain an optional privacy filter for sanitizing the data stream.

Upon receiving a new request, the onion proxy builds an onion defining the route of an anonymous connection. (It may use the destination address in the prepended structure to help define the route.) It then passes the onion to the funnel, and repeatedly precrypts the standard structure. Finally, it passes the precrypted standard structure through the anonymous connection to the exit funnel, thus specifying the ultimate destination. From this point on, the onion proxy blindly relays data back and forth between the application proxy and the onion routing network (and thus the exit funnel at the other end of the anonymous connection). Of course, it must apply the appropriate keystreams to incoming and outgoing data when blindly relaying data.

The entry funnel multiplexes connections from onion proxies to the onion routing network. For the services we have considered to date, a nearly generic exit funnel is adequate. Its function is to demultiplex connections from the last onion router to the outside. When it reads a data stream from the terminating onion router, the first datum received will be the standard structure specifying the ultimate destination. The exit funnel makes a socket connection to that IP address/port, reports a one-byte status message back to the onion routing network (and thus back to the onion proxy which in turn forward it back to the application proxy), and subsequently moves data between the onion routing network and the new socket. (For certain services, like RLOGIN, the exit funnel also infers that the new socket must originate from a trusted port.) Entry and exit funnels are not application specific but must understand the onion routing protocol, that defines how multiplexed connections are handled.

As an example, consider the application proxy for HTTP. The user configures his browser to use the onion routing proxy. His browser may send the proxy a request like `GET http://www.onion-router.net/index.html HTTP1.0` followed by optional fields.

The application proxy is listening for new requests. Once it obtains the `GET` request, it creates the standard structure and sends it (along a new socket connection) to the onion proxy, to inform the onion proxy of the service and destination of the anonymous connection. The application proxy then modifies the `GET` request to `GET/index.html HTTP/1.0` and sends it directly (through the anonymous connection) to the HTTP server `www.onion-router.net`, followed by the optional fields. Notice that the server name and `http://` are eliminated from the `GET` request because the connection is made directly to the HTTP server.

The application proxy essentially makes a connection to `www.onion-router.net`, and issues a request as if it were a client. Once this request is transmitted to the server, all proxies blindly forward data in both directions between the

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |    Protocol   |  Retry Count  |  Addr Format  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 1. The standard structure.

For the anonymizing onion routing HTTP proxy, the application proxy proceeds as outlined above with one change: It is now necessary to sanitize the optional fields that follow the `GET` command because they may contain identity information. Furthermore, the data stream during a connection must be monitored to sanitize additional headers that might occur during the connection. For our current anonymizing HTTP proxy, operations that store cookies on the user's browser (to track a user, for example) are removed. This reduces function, so applications that depend upon cookies (like online shopping baskets) may not work properly.

B. *Implementation*

This section presents the interface specification between the components in an onion routing system. To provide some structure to this specification, we will discuss components in the order that data would move from an initiating client to a responding server.

There are four phases in an onion routing system: network setup, that establishes the longstanding connections between onion routers; connection setup, which establishes anonymous connections through the onion router network; data movement over an anonymous connection; and the destruction and cleanup of anonymous connections. We will commingle the discussion of these below.

C. *Application Proxy*

The interface between an application and the application proxy is application specific. The interface between the application proxy and the onion proxy is defined as follows: For each new proxy request, the application proxy first determines if it will handle or deny the request. If rejected, it reports an application-specific error message and then closes the socket and waits for the next request. If accepted, it creates a socket to the onion proxy's well-known port. The application proxy then sends a standard structure to the onion proxy of the form as shown in Fig. 1.

Version is currently defined to be 1. *Protocol* is either 1 for RLOGIN, 2 for HTTP, or 3 for SMTP. *Retry Count* specifies how many times the exit funnel should attempt to retry connecting to the ultimate destination. Finally, the *Addr Format* field specifies the form of the ultimate destination address: 1 for a NULL terminated ASCII string with the hostname or IP address (in ASCII form) immediately followed by another NULL terminated ASCII string with the destination port number, and all others currently undefined. The ultimate destination address is sent after this standard structure, and the application proxy waits for a one byte error code before

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.