USENIX Association

Proceedings of the 11th USENIX Security Symposium

San Francisco, California, USA
August 5-9, 2002

© 2002 by The USENIX Association All Rights Reserved For more information about the USENIX Association:
Phone: 1 510 528 8649 FAX: 1 510 548 5738 Email: office@usenix.org WWW: http://www.usenix.org
Rights to individual papers remain with the author or the author's employer.
Permission is granted for noncommercial reproduction of the work for educational or research purposes.
This copyright notice must be included in the reproduced paper. USENIX acknowledges all trademarks herein.

Infranet: Circumventing Web Censorship and Surveillance

Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, David Karger
MIT Laboratory for Computer Science
{feamster, mbalazin, gch, hari, karger}@lcs.mit.edu
http://nms.lcs.mit.edu/projects/infranet

Abstract

An increasing number of countries and companies routinely block or monitor access to parts of the Internet. To counteract these measures, we propose Infranet, *a system that enables clients to surreptitiously retrieve sensitive content via cooperating Web servers distributed across the global Internet. These Infranet servers provide clients access to censored sites while continuing to host normal uncensored content. Infranet uses a tunnel protocol that provides a covert communication channel between its clients and servers, modulated over standard HTTP transactions that resemble innocuous Web browsing. In the upstream direction, Infranet clients send covert messages to Infranet servers by associating meaning to the* sequence *of HTTP requests being made. In the downstream direction, Infranet servers return content by hiding censored data in uncensored images using steganographic techniques. We describe the design, a prototype implementation, security properties, and performance of Infranet. Our security analysis shows that Infranet can successfully circumvent several sophisticated censoring techniques.*

1 Introduction

The World Wide Web is a prime facilitator of free speech; many people rely on it to voice their views and to gain access to information that traditional publishing venues may be loath to publish. However, over the past few years, many countries, political regimes, and corporations have attempted to monitor and often restrict access to portions of the Web by clients who use networks they control. Many of these attempts have been successful, and the use of the Web as a free-flowing medium for information exchange is being severely compromised.

Several countries filter Internet content at their borders, fearful of alternate political views or external influences. For example, China forbids access to many news sites that have been critical of the country's domestic policies. Saudi Arabia is currently soliciting content filter vendors to help block access to sites that the government deems inappropriate for political or religious reasons [10]. Germany censors all Nazi-related material. Australia's laws ban pornography. In addition, Internet censorship repeatedly threatens to cross political boundaries. For example, the U.S. Supreme Court recently rejected France's request to censor Nazi-related material on Yahoo's site [12]. Censorship and surveillance also extend into free enterprise, with several companies in the U.S. reportedly blocking access to sites that are not related to conducting business. In addition to blocking sites, many companies routinely monitor their employees' Web surfing habits.

This paper focuses on the challenging technical problems of circumventing Web censorship and largely ignores the many related political, legal, and policy issues. In particular, we investigate how to leverage Web communication with accessible servers in order to surreptitiously retrieve censored content, while simultaneously maintaining plausible deniability against receiving that content. To this end, we develop a *covert communication tunnel* that securely hides the exchange of censored content in normal, innocuous Web transactions.

Our system, called *Infranet*, consists of *requesters* and *responders* communicating over this covert tunnel. A requester, running on a user's computer, first uses the tunnel to request censored content. Upon receiving the request, the responder, a standard public Web server running Infranet software, retrieves the sought content from the Web and returns it to the requester via the tunnel.[1]

The covert tunnel protocol between an Infranet requester and responder must be difficult to detect and block. More specifically, a censor should not be able to detect that a Web server is an Infranet responder or that a client is an In-

[1] We use the terms "requester" and "responder" rather than the more traditional "client" and "server" to avoid confusion with Web clients ("browsers") and Web servers. We also considered a number of terms like "proxy", "gateway", "front-end", etc., but rejected them for similar reasons.

franet requester. Nothing in their HTTP transactions ought to arouse suspicion.

The Infranet tunnel protocol uses novel techniques for covert upstream communication. It modulates covert messages on standard HTTP requests for uncensored content using a confidentially negotiated function which maps URLs to message fragments that compose requests for censored content. For downstream communication, the tunnel protocol leverages existing data hiding techniques, such as steganography. While steganography provides little defense against certain attacks, we are confident that the ideas we present can be used in conjunction with other data hiding techniques.

The main challenge in the design of the tunnel protocol is ensuring covertness while providing a level of performance suitable for interactive browsing. Furthermore, the tunnel protocol must defend against a censor capable of passive attacks based on logging all transactions and packets, active attacks that modify messages or transactions, and impersonation attacks where the adversary pretends to be a legitimate Infranet requester or responder. Our security analysis indicates that Infranet can successfully circumvent several sophisticated censoring techniques, including various active and passive attacks. Our system handles almost all of these threats while achieving reasonable performance. This is achieved by taking advantage of the asymmetric bandwidth requirements of Web transactions, which require significantly less upstream bandwidth than downstream bandwidth.

To assess the feasibility of our design, we implemented an Infranet prototype and conducted a series of tests using client-side Web traces to evaluate the performance of our system. Our experimental evaluation shows that Infranet provides acceptable bandwidth for covert Web browsing. Our range-mapping algorithm for upstream communication allows a requester to innocuously transmit a hidden request in a number of visible HTTP requests that is proportional to the binary entropy of the hidden request distribution. For two typical Web sites running Infranet responders, we find that a requester using range-mapping can modulate 50% of all requests for hidden content in 6 visible HTTP requests or fewer and 90% of all hidden requests in 10 visible HTTP requests or fewer. Using typical Web images, our implementation of downstream hiding transmits approximately 1 kB of hidden data per visible HTTP response.

2 Related Work

Many existing systems seek to circumvent censorship and surveillance of Internet traffic. `Anonymizer.com` provides anonymous Web sessions by requiring users to make Web requests through a proxy that anonymizes user-specific information, such as the user's IP address [2]. The company also provides a product that encrypts HTTP requests to protect user privacy; Zero Knowledge provides a similar product [24]. Squid is a caching Web proxy that can be used as an anonymizing proxy [21]. The primary shortcoming of these schemes is that a well-known proxy is subject to being blocked by a censor. Additionally, the use of an encrypted tunnel between a user and the anonymizing proxy (e.g., port forwarding over `ssh`) engenders suspicion.

Because censoring organizations are actively discovering and blocking anonymizing proxies, SafeWeb has proposed a product called Triangle Boy, a peer-to-peer application that volunteers run on their personal machines and that forwards clients' Web requests to SafeWeb's anonymizing proxy [19, 27]. SafeWeb recently formed an alliance with the Voice of America [28], whose mission is to enable Chinese Internet users to gain access to censored sites. However, Triangle Boy has several drawbacks. First, the encrypted connection to a machine running Triangle Boy is suspicious and can be trivially blocked since SSL handshaking is unencrypted. Second, SafeWeb's dependence on an encrypted channel for confidentiality makes it susceptible to traffic analysis, since Web site fingerprinting can expose the Web sites that a user requests, even if the request itself is encrypted [7]. Third, SafeWeb is vulnerable to several attacks that allow an adversary to discover the identity of a SafeWeb user, as well as every Web site visited by that user [11]. Peekabooty also attempts to circumvent censoring firewalls by sending SSL-encrypted requests for censored content to a third party, but its reliance on SSL also makes it susceptible to traffic analysis and blocking attacks [26].

Various systems have attempted to protect anonymity for users who publish and retrieve censored content. In Crowds, users join a large, geographically diverse group whose members cooperate in issuing requests, thus making it difficult to associate requests with the originating user [18]. Onion routing also separates requests from the users who make them [25]. Publius [30], Tangler [29], and Free Haven [4] focus on protecting the anonymity of publishers of censored content and the content itself. Freenet provides anonymous content storage and retrieval [3].

Infranet aims to overcome censorship and surveillance, but also provides plausible deniability for users. In addition to establishing a *secure channel* between users and Infranet responders, our system creates a *covert channel* within HTTP, i.e., a communication channel that transmits information in a manner not envisioned by the original design of HTTP [9]. In contrast with techniques that attempt to overcome censorship using a confidential channel (e.g., using SSL, which is trivial to detect and block) [19, 23, 24, 26], our approach is significantly harder to detect or block. To be effective against blocking, a scheme for circumventing censorship must be covert as well as secure.

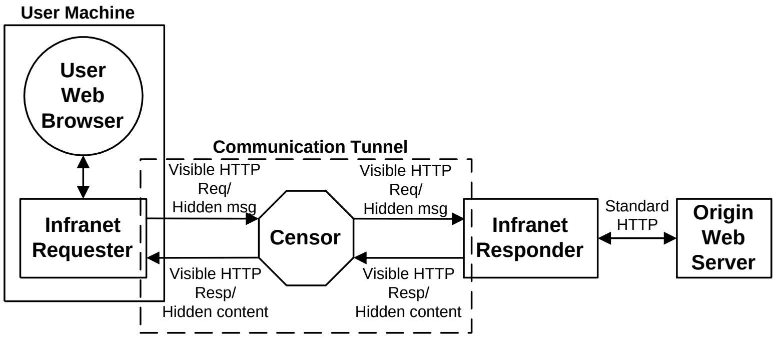

User Machine
User Web Browser
Infranet Requester
Communication Tunnel
Visible HTTP Req/ Hidden msg
Censor
Visible HTTP Req/ Hidden msg
Infranet Responder
Standard HTTP
Origin Web Server
Visible HTTP Resp/ Hidden content
Visible HTTP Resp/ Hidden content

Figure 1. Infranet system architecture.

3 System Architecture

This section presents Infranet's design considerations and system architecture and gives an overview of the system's communication protocols.

3.1 Terminology

Figure 1 shows the system architecture of Infranet and introduces relevant terminology. Users surf Web content as usual via a Web browser. To retrieve censored content, the browser uses a software entity that runs on the same host, called the *Infranet requester*, as its local proxy. The Infranet requester knows about one or more *Infranet responders*, which are Web servers in the global Internet that implement additional Infranet functionality. The idea is for the Web browser to request censored content via the Infranet requester, which in turn sends a message to an Infranet responder. The responder retrieves this content from the appropriate origin Web server and returns it to the requester, which delivers the requested content to the browser. The requester and responder communicate with each other using a covert *tunnel*. Technically, Infranet involves three distinct functions—issuing a hidden request, decoding the hidden request, and serving the requested content. We first describe a system whereby the responder performs the latter two functions. We describe a design enhancement in Section 8 whereby an untrusted *forwarder* can forward hidden requests and serve hidden content, thereby making it more difficult for a censor to block access to the system.

The *censor* shown in Figure 1 might have a wide range of capabilities. At a minimum, the censor can block specific IP addresses (e.g., of censored sites and suspected Infranet responders). More broadly, the censor might have the capability to analyze logs of all observed Web traffic, or even to modify the traffic itself.

The long-term success of Infranet depends on the widespread deployment of Infranet responders in the Internet. One way of achieving this might be to bundle responder software with standard Web server software (e.g., Apache). Hopefully, a significant number of people will run Infranet responders due to altruism or because they believe in free speech.

3.2 Design Goals

We designed Infranet to meet a number of goals. Ordered by priority, the goals are:

1. *Deniability for any Infranet requester.* It should be computationally intractable to confirm that any individual is intentionally downloading information via Infranet, or to determine what that information might be.

2. *Statistical deniability for the requester.* Even if it is impossible to confirm that a client is using Infranet, an adversary might notice statistical anomalies in browsing patterns that suggest a client is using Infranet. Ideally, an Infranet user's browsing patterns should be statistically indistinguishable from those of normal Web users.

3. *Responder covertness.* Since an adversary will likely block all known Infranet responders, it must be difficult to detect that a Web server is running Infranet simply by watching its behavior. Of course, any requester using the server will know that the server is an Infranet responder; however, this knowledge should only arise from possession of a secret that remains unavailable to the censor. If the censor chooses not to block access to the responder but rather to watch clients connecting to it for suspicious activities, deniability should not be compromised. The responder must assume that *all clients are Infranet requesters*. This ensures that Infranet requesters cannot be distinguished from innocent users based on the responder's behavior.

4. *Communication robustness.* The Infranet channel should be robust in the presence of censorship activities designed to interfere with Infranet communication. Note that it is impossible to be infinitely robust, because a censor who blocks all Internet access will successfully prevent Infranet communication. Thus, we assume the censor permits some communication with non-censored sites.

Any technique that prevents a site from being used as an Infranet responder should make that site fundamentally unusable by non-Infranet clients. As an example of a scheme that is *not* robust, consider using SSL as our Infranet channel. While this provides full requester and responder deniability and covertness (since many Web servers run SSL for innocent reasons), it is quite plausible for a censor to block *all* SSL access to the Internet, since vast amounts of information remain accessible through non-encrypted connections. Thus, a censor can block SSL-Infranet without completely restricting Internet access.

In a similar vein, if the censor has concluded that a particular site is an Infranet responder, we should ensure that their only option for blocking Infranet access is to block *all* access to the suspected site. Hopefully, this will make the censor more reluctant to block sites, which will allow more Infranet responders to remain accessible.

5. *Performance.* We seek to maximize the performance of Infranet communication, subject to our other objectives.

3.3 Overview

A requester must be able to both join and use Infranet without arousing suspicion. To join Infranet, a user must obtain the Infranet requester software, plus the IP address and public key of at least one Infranet responder. Users must be able to obtain Infranet requester software without revealing that they are doing so. Information about Infranet responders must be available to legitimate users, but should not fall into the hands of an adversary who could then configure a simple IP-based filtering proxy. One way to distribute software is out-of-band via a CD-ROM or floppy disk. Users can share copies of the software and learn about Infranet responders directly from one another.

The design and implementation of a good tunnel protocol between an Infranet requester and responder is the critical determinant of Infranet's viability. The rest of this section gives an overview of the protocol, and Section 4 describes the protocol in detail.

We define the tunnel protocol between the requester and responder in terms of three abstraction layers:

1. *Message exchange.* This layer of abstraction specifies high-level notions of information that requester and responder communicate to each other.
2. *Symbol construction.* Any communication system must specify an underlying alphabet of symbols that are transmitted. This layer of abstraction specifies the alphabets for both directions of communication. The primary design constraint is covertness.
3. *Modulation.* The lowest layer of abstraction specifies the mapping between symbols in the alphabet and message fragments. The main design goal is reasonable communication bandwidth, without compromising covertness.

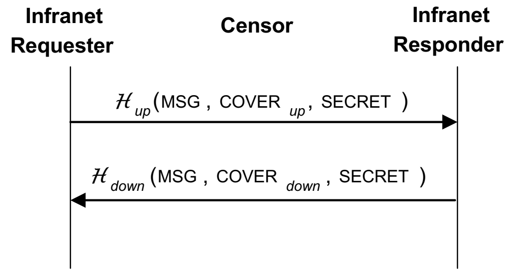

Infranet Requester
Censor
Infranet Responder
$\mathcal{H}_{up}$(MSG , COVER $_{up}$, SECRET)
$\mathcal{H}_{down}$(MSG , COVER $_{down}$, SECRET)

Figure 2. Top-most layer of abstraction in communication tunnel.

The top-most layer of abstraction is the exchange of messages between the requester and responder. The requester sends requests for content to the responder, hidden in visible HTTP traffic. The responder answers with requested content, hidden in visible HTTP responses, after obtaining it from the origin server. As shown in Figure 2, messages are hidden with a *hiding function* $\mathcal{H}(\text{MSG}, \text{COVER}, \text{SECRET})$, where MSG is the message to be hidden, COVER is the visible traffic medium in which MSG is hidden, and SECRET ensures that only the requester and responder can reveal MSG.

Designing the hiding function $\mathcal{H}$ involves defining a set of *symbols* that map onto message fragments. The set of all symbols used to transmit message fragments is called an *alphabet*. Since an ordered sequence of fragments forms a message, an ordered sequence of symbols, along with the hiding function, also represent a message. Both upstream and downstream communication require a set of symbols for transmitting messages.

The lowest abstraction layer is *modulation*, which specifies the mapping between message fragments and symbols. We discuss several ways to modulate messages in the upstream and downstream directions in Sections 4.2 and 4.3.

3.3.1 Upstream Communication

In our design, the cover medium for upstream communication, COVER_{up}, is sequences of HTTP requests (note that *which* link is selected on the page contains information and can therefore be used for communication).

The alphabet is the set of URLs on the responder's Web site. Other possible alphabets exist, such as various fields in the HTTP and TCP headers. We choose to use the set of URLs as our alphabet because it is more difficult for the transmission of messages to be detected, it is immune to malicious field modifications by a censor, and it provides reasonable bandwidth. Careful metering of the order and timing of the cover HTTP communication makes it difficult for the censor to distinguish Infranet-related traffic from regular Web browsing. Upstream modulation corresponds

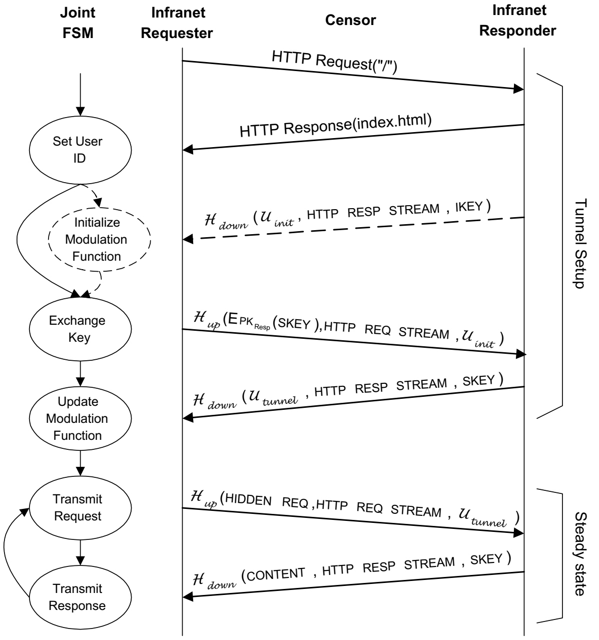

Joint FSM
Infranet Requester
Censor
Infranet Responder
HTTP Request("/")
HTTP Response(index.html)
Set User ID
Initialize Modulation Function
$\mathcal{H}_{down}(\mathcal{U}_{init}$, HTTP RESP STREAM , IKEY)
Exchange Key
$\mathcal{H}_{up}$(E$_{PK_{Resp}}$(SKEY),HTTP REQ STREAM ,$\mathcal{U}_{init}$)
Update Modulation Function
$\mathcal{H}_{down}(\mathcal{U}_{tunnel}$, HTTP RESP STREAM , SKEY)
Tunnel Setup
Transmit Request
$\mathcal{H}_{up}$(HIDDEN REQ ,HTTP REQ STREAM , $\mathcal{U}_{tunnel}$)
Transmit Response
$\mathcal{H}_{down}$ (CONTENT , HTTP RESP STREAM , SKEY)
Steady state

Figure 3. Messages exchanged during the tunnel setup and steady state communication phases. Message exchanges are driven by a common state machine shown on the left. In the optional *Initialize Modulation Function* state, the responder sends an initial modulation function $\mathcal{U}_{init}$ to the requester.

to mapping a sequence of one or more URL retrievals, visible to a censor, to a surreptitious request for a censored Web object.

3.3.2 Downstream Communication

The cover medium in the downstream direction, COVER$_{down}$, is provided by JPEG images (within HTTP response streams). The responder uses the high frequency components of images as its alphabet for sending messages to the requester. This technique provides good bandwidth and hiding properties. Downstream modulation consists of mapping a sequence of high-frequency image components to the censored web object, such as HTML or MIME-encoded content.

4 Tunnel Protocol

The Infranet tunnel protocol is divided into three main components: tunnel setup, upstream communication, and downstream communication. Tunnel setup allows both parties to agree on communication parameters. Upstream communication consists of message transmissions from requester to responder. Downstream communication consists of message transmissions in the opposite direction.

Both the requester and responder operate according to the finite state machine shown in the left column of Figure 3. The first four states constitute *tunnel setup*. The last two compose *steady state* communication, where the requester transmits hidden URLs and the responder answers with the corresponding content.

We now explore various design alternatives and describe the mechanisms used for each part of the protocol.

4.1 Tunnel Setup

An Infranet requester and responder establish a tunnel by agreeing on parameters to the hiding functions $\mathcal{H}_{up}$ and $\mathcal{H}_{down}$. The requester and responder exchange these parameters securely, thereby ensuring confidentiality during future message exchanges.

Figure 3 shows the messages involved in establishing the tunnel. Communication with an Infranet responder begins with a request for an HTML page served by the responder. This first request initiates the following tunnel setup protocol:

1. *Set User ID*

   The requester sends an implicit HELLO message to the responder by requesting an HTML document, such as `index.html`.

   To identify subsequent message transmissions from the requester, the responder creates a unique user ID for the requester. This user ID could be explicitly set via a Web cookie. However, for greater defense against tampering, the user ID should be set implicitly. As explained later, the responder modifies the visible URLs on its Web site for each requester. Such modification is sufficient to identify requesters based on which URLs are requested.

2. *Exchange Key*

   To ensure confidentiality, the requester uses a responder-specific modulation function $\mathcal{U}_{init}$ to send a shared secret, SKEY, encrypted with the public key of the responder.

   The responder recovers SKEY using its private key.

3. *Update Modulation Function*

   The responder first selects a requester-specific modulation function $\mathcal{U}_{tunnel}$. Next, the responder hides the function in an HTTP response stream with the shared secret SKEY.

   The requester recovers $\mathcal{U}_{tunnel}$ from the HTTP response stream using SKEY.

Thus, the tunnel setup consists of the exchange of two secrets: a secret key SKEY, and a secret modulation function $\mathcal{U}_{tunnel}$. SKEY ensures that only the requester is capable of decoding the messages hidden in HTTP response streams. $\mathcal{U}_{tunnel}$ allows the requester to hide messages in HTTP request streams. The secrecy of $\mathcal{U}_{tunnel}$ provides confidentiality for upstream messages by ensuring that it is hard for a censor to uncover the surreptitious requests, even if a requester is discovered.

In order for the requester to initiate the transmission of SKEY, encrypted with the Infranet responder's public key, the requester must have a way of sending a message to the responder. The transmission is done using an initial modulation function, $\mathcal{U}_{init}$. This initial function may be a well-known function. Alternatively, the responder may send an initial modulation function, $\mathcal{U}_{init}$ to the requester. To protect responder covertness, this initial function should be hidden using a responder-specific key IKEY. The requester may learn IKEY with the IP address and public key of the responder. This method, which requires the additional *Initialize Modulation Function* state, has the advantage of allowing responders to periodically change modulation schemes, but suffers the disadvantage of requiring more HTTP message exchanges to establish a tunnel.

With the tunnel established, the requester and responder enter the *Transmit Request* state. In this state, the requester uses $\mathcal{U}_{tunnel}$ to hide a request for content in a series of HTTP requests sent to the Infranet responder. When the covert request completes, the requester and responder enter the *Transmit Response* state, at which point the responder fetches the requested content and hides it in an HTTP response stream using SKEY. When the transmission is complete, the requester and responder both re-enter the *Transmit Request* state.

4.2 Upstream Communication

At the most fundamental level, a requester sends a message upstream by sending the responder a visible HTTP request that contains additional hidden information. Figure 4 shows the decomposition of the upstream hiding function $\mathcal{H}_{up}(\text{MSG}, \text{HTTP REQUEST STREAM}, \mathcal{U}_x)$, where MSG is the transmitted information (e.g., request for hidden content), HTTP REQUEST STREAM is the cover medium, and $\mathcal{U}_x$ is a modulation function that hides the message in a *visible HTTP request stream*. The specific mapping from message fragments to visible HTTP requests depends on the parameter x.

To send a hidden message, a requester divides it into multiple fragments, each of which translates to a visible HTTP request. The responder applies $\mathcal{U}_x^{-1}$ to the requester's HTTP requests to extract the message fragments and reassembles them to recover the hidden message.

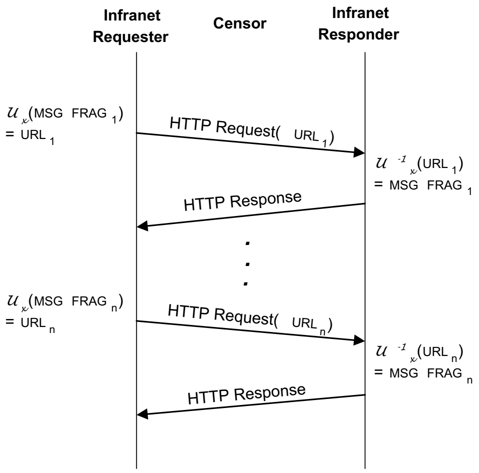

Infranet Requester
Censor
Infranet Responder
$\mathcal{U}_x$(MSG FRAG $_1$) = URL $_1$
HTTP Request(URL $_1$)
$\mathcal{U}^{-1}_x$(URL $_1$) = MSG FRAG $_1$
HTTP Response
$\mathcal{U}_x$(MSG FRAG $_n$) = URL $_n$
HTTP Request(URL $_n$)
$\mathcal{U}^{-1}_x$(URL $_n$) = MSG FRAG $_n$
HTTP Response

Figure 4. Sequence of HTTP requests and responses involved in a single upstream message transmission. Both parties must know the secret modulation function $\mathcal{U}_x$.

There are many possible choices for the upstream modulation function $\mathcal{U}_x$. Each option for $\mathcal{U}_x$ presents a different design tradeoff between covertness and upstream bandwidth. There are many modulation functions that provide deniability for an Infranet requester—certain types of basic mapping schemes, when implemented correctly, can do so. We describe two such examples in Sections 4.2.1 and 4.2.2. To provide statistical deniability, however, requests should follow typical browsing patterns more closely. To achieve this, we propose the range-mapping scheme in Section 4.2.3.

4.2.1 Implicit Mapping

One of the simplest $\mathcal{U}_x$ modulates each bit of a hidden message as a separate HTTP request. While this approach provides extremely limited bandwidth, it offers a high level of covertness—on any given page the requester may click on any one of half of the links to specify the next fragment. For example, one can specify that any even-numbered link on the page corresponds to a 0, while any odd-numbered link corresponds to a 1. A generalization of this scheme uses the function $R \bmod n$, where R is specified by the Rth link on the last requested page and n is at most equal to the total number of links on that page. This mechanism may be less covert, but sends $\lg(n)$ bits of information per visible HTTP request.

4.2.2 Dictionary-based Schemes

An Infranet responder can send the requester a static or dynamic *codebook* that maps visible HTTP requests to message fragments. While a static mapping between visible HTTP requests and URLs is simple to implement, the resulting visible HTTP request streams may result in strange browsing behavior. To create a dynamic mapping, the responder uses images embedded in each requested page to send updates to the modulation function as the upstream transmission progresses. The responder may also use its log of hidden requests to provide most probable completions to an ongoing message transmission. Transmission of a b-bit hidden message using a dictionary-based scheme, where each dictionary entry contains M bits requires b/M visible HTTP requests.

The structure of the dictionary determines both the covertness and bandwidth of the modulated request. Therefore, the dictionary might be represented as a directed graph, based on the structure of the Infranet responder's Web site. To preserve confidentiality in the event that a communication tunnel is revealed, the dictionary should be known only to the requester and the responder.

4.2.3 Range-mapping

The requester and responder communicate via a channel with far greater bandwidth from the responder to the requester than vice versa. Because the responder serves many Infranet users' requests for hidden content, it can maintain the frequency distribution of hidden messages, $\mathcal{C}$. A requester wants to send a message, URL, from the distribution $\mathcal{C}$. This communication model is essentially the asymmetric communication model presented by Adler and Maggs [1].

We leverage their work to produce an iterative modulation function based on *range-mapping* of the distribution of lexicographically ordered URLs, $\mathcal{C}$. In each round, the responder sends the requester a set $\mathcal{S}$ of tuples (s_i, r_i), where s_i is a string in the domain of $\mathcal{C}$ and r_i is the visible HTTP request that communicates s_i. s_i is called a *split-string*. It specifies the *range* of strings in $\mathcal{C}$ that are lexicographically smaller than itself and lexicographically larger than the preceding split-string in $\mathcal{S}$. The client determines which lexicographic interval contains the hidden message and responds with the split-string that identifies that interval.

While the focus of the Adler-Maggs protocol is to enable communication over an asymmetric channel, we are also concerned with maintaining covertness and statistical deniability. In particular, we aim to ensure that at each step, the probability that an Infranet requester selects a particular link is equal to the probability that an innocent browser selects that link. We therefore include link traversal probability information as a parameter to the algorithm for choosing split-strings. We extract these probabilities from the server's access log.

```
PROCEDURE MODULATE(URL, $\mathcal{S}$)
    // Select the smallest split-string from $\mathcal{S}_s$
    // lexicographically larger than URL
    $\mathrm{string_{max}} \leftarrow \{u \mid u \in \mathcal{S}_s$ and $u > url$
                  and $\nexists v \in \mathcal{S}$ s.t. $url < v < u\}$
    // Request the page corresponding to the selected string
    $r \leftarrow \{\mathcal{S}_r[i] \mid \mathcal{S}_s[i] = \mathrm{string_{max}}\}$
    send $r$
```

Figure 5. Pseudocode for a modulation function using range-mapping.

Prior to communicating with the requester, the responder computes the following information offline: $\mathcal{C}$, the cumulative frequency distribution for hidden requests; and $\mathcal{P}$, the link-traversal probabilities. Specifically, $\mathcal{P}$ is the set of probabilities $p_{ij} = P(\text{next request is for page } p_j | \text{current page is } p_i)$ for all pages p_i and p_j in R, the set of all pages on the responder's Web site.

In each round, the set $\mathcal{S}$ contains k tuples, where k is the number of pages r for which $P(r|r_{current}) > 0$, i.e., the number of possibilities for the requester's next visible HTTP request, given the current page $r_{current}$. These tuples specify k consecutive probability intervals within $\mathcal{C}$. The size of each probability interval Δv_i is proportional to the conditional probability $P(r|r_{current})$. By assigning probability intervals according to the next-hop probabilities for HTTP requests on a Web site, range-mapping provides statistical deniability for the requester by making it more likely that the requester will take a path through the site that would be taken by an innocuous Web client.[2] The sum of all k intervals is equal to δ, the size of the probability interval for the previous iteration, or to 1 for the first iteration.

Pseudocode for the modulation function is shown in Figure 5. In each iteration, the requester receives $\mathcal{S}$ and selects the split string s that specifies the range in which its message URL lies. It then sends the corresponding visible HTTP request r.

Figure 6 shows pseudocode for the demodulation function. The responder interprets the request $r_{current}$ as a range specification, bounded above by the corresponding split-string $s_{current}$ and below by split-string in $\mathcal{S}$ which precedes $s_{current}$ lexicographically. Given this new range, the responder updates the bounds for the range, $\mathrm{string_{min}}$ and $\mathrm{string_{max}}$ (lines 12-13), and generates a new split-string set $\mathcal{S}$ for that range (as shown in lines 5-9 and Figure 7).

A requester can use this scheme to modulate the hidden message URL even if it is not in the domain of $\mathcal{C}$. The re-

[2] We present the range-mapping model based on one-hop conditional probabilities. It should be noted that although this approach provides the appropriate distribution on link probabilities at each step, it is not guaranteed to properly distribute more complex quantities such as the probability of an entire sequence of link choices.

```
PROCEDURE DEMODULATE($\mathcal{P}, \mathcal{C}, r_{current}, \text{string}_{\min}, \text{string}_{\max}$)
    // When the range reaches zero, return the string found
    if ($\text{string}_{\min} = \text{string}_{\max}$)
        return $\text{string}_{\min}$
    else
        // Compute total range for this iteration
        $\delta \leftarrow \mathcal{C}(\text{string}_{\max}) - \mathcal{C}(\text{string}_{\min})$
        // Initialize lower bound of first sub-interval
        $v_{\min} \leftarrow \mathcal{C}(\text{string}_{\min})$
        // For all openly served pages $r$ in $R$, where
        // $R$ is the set of all pages on the Web site
        $\forall r \in R$
            // Set the upper bound of the sub-interval proportional
            // to the probability of requesting page $r$
            $v \leftarrow \delta \cdot \mathcal{P}(r|r_{current}) + v_{\min}$
            // Extract the string at the boundary of the sub-interval
            // This is the split-string for the sub-interval
            $s \leftarrow \mathcal{C}^{-1}(v)$
            // Save the pair formed by split-string $s$ and page $r$
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{(s, r)\}$
            Prepare to compute subsequent sub-interval
            $v_{\min} \leftarrow v$
        // Send the set of all pairs to the requester
        send $S$
        // Receive new HTTP request
        receive $r_{current}$
        // Update interval given the selected split-string
        $\text{string}_{\max} \leftarrow \{\mathcal{S}_s[i] \mid \mathcal{S}_r[i] = r_{current}\}$
        $\text{string}_{\min} \leftarrow \begin{cases} \{\mathcal{S}_s[i] \mid \mathcal{S}_s[i+1] = \text{string}_{\max}\} \text{ if } i \neq 0 \\ 0 \text{ otherwise} \end{cases}$
        // Further reduce interval
        return DEMODULATE($\mathcal{P}, C, r_{current}, \text{string}_{\min}, \text{string}_{\max}$)
```

Figure 6. Pseudocode for a demodulation function using range-mapping.

quester and responder can perform range-mapping until the range becomes two consecutive URLs in $\mathcal{C}$. The prefix that these two URLs share becomes the prefix p of the hidden message. At this point, the requester and responder may continue the range-mapping algorithm over the set of all strings that have p as a prefix.

Since the requester's message may be of arbitrary length, there must exist an explicit way to stop the search. One solution is to add a tuple (ϵ, r_{end}) to $\mathcal{S}$, where ϵ indicates that the requester is finished sending the request. When all split-strings share a common prefix equal to the hidden message, the requester transmits r_{end}.

Range-mapping is similar to arithmetic coding, which divides the size of each interval in the space of binary strings according to the probability of each symbol being transmitted. The binary entropy, $H(\mathcal{C})$, is the expected number of bits required to modulate a message in $\mathcal{C}$. Arithmetic coding of a binary string requires $H(\mathcal{C}) + 2$ transmissions, assuming 1 bit per symbol [31]. In our model, each page has k links. Therefore, each visible HTTP request transmits $\lg(k)$ bits, and the expected number of requests required to modulate a hidden request is $\left\lceil \frac{H(\mathcal{C})}{\lg k} \right\rceil + 2$.

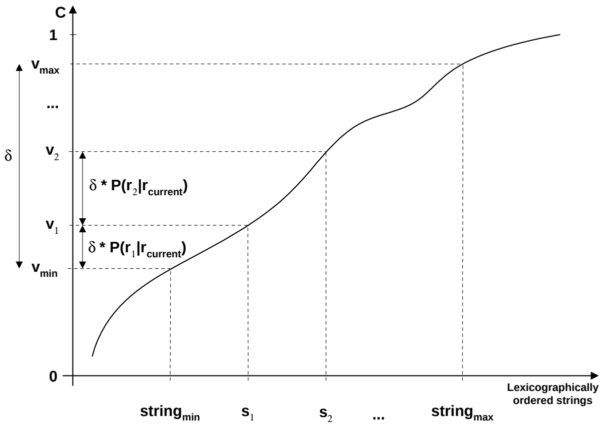

C
1
0
v_{max}
...
v_2
v_1
v_{min}
δ
δ * P(r_2|$r_{current}$)
δ * P(r_1|$r_{current}$)
Lexicographically ordered strings
$string_{min}$
s_1
s_2
...
$string_{max}$

Figure 7. One iteration of the range-mapping demodulation function. The initial interval $[\mathcal{C}(\text{string}_{\min}), \mathcal{C}(\text{string}_{\max})]$ is divided into k sub-intervals according to probabilities of requesting any page on the responder's Web site given the current page $r_{current}$. The responder generates the split-strings s_1 through s_k that correspond to sub-interval boundaries and returns them to the requester.

4.3 Downstream Communication

Figure 8 shows the decomposition of the downstream hiding function $\mathcal{H}_{down}$. The requester receives a hidden message from the responder by making a series of HTTP requests for images. The responder applies $\mathcal{D}$ to the requested images and sends the resulting images to the requester. The requester can then apply the inverse modulation function $\mathcal{D}^{-1}$ to recover the hidden message fragment. To ensure innocuous browsing patterns, the requester should request an HTML page and subsequently request the embedded images from that page (as opposed to making HTTP requests for images out of the blue).

For the modulation function $\mathcal{D}$, we use the `outguess` utility [16], which modifies the high frequency components of an image according to both the message being transmitted and a secret key. Modulation takes place in two stages—finding redundant bits in the image (i.e., the least significant bits of DCT coefficients in the case of JPEG), and embedding the message in some subset of these redundant bits. The first stage is straightforward. The second stage uses

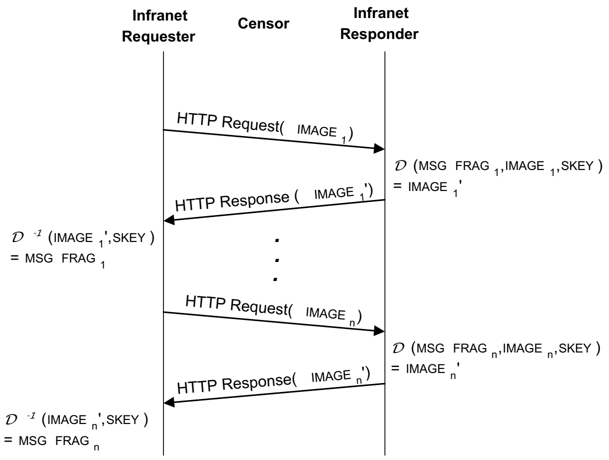


Figure 8. Downstream communication also consists of a sequence of HTTP requests and responses. The responder hides messages that it sends to the requester in the HTTP responses. One efficient downstream communication mechanism uses steganography to hide downstream messages in requested images.

the shared secret SKEY as a seed to a pseudorandom number generator that determines which subset of these bits will contain the message. Therefore, without knowing the secret key, an adversary cannot determine which bits hold information. Previous work describes this process in greater detail [17].

Steganography is designed to hide a message in a cover image, where the adversary does not have access to the original cover and thus cannot detect the presence of a hidden message. However, because a Web server typically serves the same content to many different users (and even to the same user multiple times), an adversary can detect the use of steganography simply by noticing that the same requested URL corresponds to different content every time it is requested. One solution to this problem is to require that the responder never serve the same filename twice for files that embed hidden information. For this reason, a webcam serves as an excellent mode for transmitting hidden messages downstream, because the filenames and images that a webcam serves regularly change by small amounts. We discuss this problem in more detail in Section 5. To protect Infranet requesters, Infranet responders embed content in *every* image, regardless of whether the Web client is an Infranet requester.

There are many other possible modulation functions for hiding downstream messages. One possibility is to embed messages in HTTP response headers or HTML pages. However, this does not provide the downstream bandwidth that is necessary to deliver messages to the requester in a reasonable amount of time. Another alternative is to embed message fragments in images using hidden watermarks. Both watermarking and steganography conceal hidden content; additionally, watermarks are robust to modification by an adversary. Past work has investigated watermarking techniques for compressed video [6]. A feasible downstream modulation function might use downloaded or streaming audio or video clips to hide messages.

Note that our downstream modulation scheme does not fundamentally depend on the use of steganography. In fact, it may make more sense to use a data hiding technique that an adversary cannot modify or remove without affecting the visibly returned content. For example, if a responder uses *low-order bits* of the brightness values of an image to embed data, the censor will have more difficulty removing the covert data without affecting the visible characteristics of the requested image. Since we assume that the censor does not want to affect the experience of normal users, this type of downstream communication might be more appropriate.

5 Security Analysis

In this section, we discuss Infranet's ability to handle a variety of attacks from a determined adversarial censor. We are concerned with maintaining deniability and covertness in the face of these attacks, i.e., making it hard for the censor to detect requesters and responders. In addition, Infranet should provide confidentiality, so that even if a censor discovers an Infranet requester or responder, it cannot recover any of the messages exchanged.

The adversary has access to all traffic passing between its network and the global Internet, especially all visible HTTP requests and responses. Furthermore, the adversary can actively modify any traffic that passes through it, as long as these modifications do not affect the correctness of the HTTP transactions themselves.

5.1 Discovery Attacks

A censor might attempt to discover Infranet requesters or responders by joining Infranet as a requester or a responder. To join Infranet as a requester, a participant must discover the IP address and public key of a responder. Once the client joins, all information exchanged with a responder is specific to that requester. Thus, by joining the network as a requester, the censor gains no additional information other than that which must already have been obtained out-of-band.

Alternatively, a censor might set up an Infranet responder in the hope that some unlucky requesters might contact it. By determining which Web clients' visible HTTP requests demodulate to sensible Infranet messages, a cen-

	HTTP Requests	**HTTP Responses**
No Target	Suspicious HTTP request headers	Suspicious response headers or content
One Target	HTTP request patterns	Content patterns (e.g., same URL, different image)
Multiple Targets	Link requests across Infranet requesters	Common patterns in HTTP responses (e.g., for commonly requested forbidden URLs)

Table 1. A taxonomy of passive attacks on Infranet. If the censor has the ability to target suspected users, attacks involve more sophisticated analysis of visible HTTP traffic.

sor can distinguish innocent Web clients from Infranet requesters. Currently, we rely on each requester *trusting* the legitimacy of any responder it contacts. Section 8 describes a possible defense against this attack by allowing for untrusted forwarders.

A censor might mount a passive attack in an attempt to discover an Infranet communication tunnel. Because this type of attack often requires careful traffic analysis, passive attacks on Infranet are much more difficult to mount than active attacks based on filtering or tampering with visible HTTP traffic. The types of attacks that an adversary can perform depend on the amount of state it has, as well as whether or not it is targeting one or more users.

Table 1 shows a taxonomy of passive attacks that a censor can perform on Infranet. Potential attacks become more serious as the adversary targets more users. Weak attacks involve detecting anomalies in HTTP headers or content. Stronger attacks require more complex analysis, such as correlation of users' browsing patterns.

If a censor observes all traffic that passes through it without targeting users, it could attempt to uncover an Infranet tunnel by detecting suspicious HTTP request and response headers, such as a request header with a strange `Date` value or garbage in the response header. Infranet defends against these attacks by avoiding suspicious modifications to the HTTP headers and by hiding downstream content with steganography. Additionally, by requiring that Infranet responders always serve unique URLs when content changes, Infranet guards against a discovery attack on a responder, whereby a censor notices that slightly different content is being served from the same URL each time it is requested.

A censor who targets a suspected Infranet requester can mount stronger attacks. A censor can observe a Web user's browsing patterns and determine whether these patterns look suspicious. Since the modulation function determines the browsing pattern, a function that selects subsequent requests based on the structure of the responder's Web site might help, but does not always reflect actual user behavior. Some pages might be rarely requested, while others might always be requested in sequence. Thus, it is best to base modulation functions on information from real access logs.

While generating visible HTTP requests automatically requires the least work on the part of the user, this is not the only alternative. A particularly cautious user might fear that any request sequence generated by the system is likely to look "strange" and thus arouse suspicion. To overcome this, the system could give the user a certain degree of control over which visible requests are transmitted. One example is for the requester to confirm each URL with the user before sending it. If the user finds the URL strange, he can force the requester to send a different URL that communicates the same message fragment. These *overrides* introduce noise into the requester's sequence. However, the requester can encode the message with an error correcting code that allows for such noise.

Another solution would be to ensure that multiple URLs map to each message fragment the requester wants to send to give the user a choice of which specific visible URL to request. We conjecture that with sufficient redundancy, a user will frequently be able to find a plausible URL that sends the desired message fragment.

By targeting multiple users, a censor may learn about many Infranet users as a result of discovering one Infranet user. Alternatively, a censor could become an Infranet requester and compare its behavior against other suspected users. We defend against these types of attacks by using requester-specific shared secrets.

5.2 Disruptive Attacks

Because all traffic between an Infranet requester and responder passes through the censor, the censor can disrupt Infranet tunnels by performing active attacks on HTTP traffic, such as filtering, transaction tampering, and session tampering.

5.2.1 Filtering

A censor may block access to various parts of the Internet based on IP address or prefix block, DNS name, or port number. Additionally, censors can block access to content by filtering out Web pages that contain certain keywords. For instance, Saudi Arabia is reportedly trying to acquire such filtering software [10].

Infranet's success against filtering attacks depends on the pervasiveness of Infranet responders throughout the Web. Because Infranet responders are discovered out-of-band, a censor cannot rapidly learn about Infranet responders by crawling the Web with an automated script. While a censor could conceivably learn about responders out-of-band and systematically block access to these machines, the out-of-band mechanism makes it more difficult for a censor to block access to *all* Infranet responders. The wider the deployment of responders on Web servers around the world, the more likely it is that Infranet will succeed.

Note that because the adversary may filter traffic based on content and port number, it is relatively easy for the adversary to block SSL by filtering SSL handshake messages. Thus, Infranet provides far better defense against filtering than a system that simply relies on SSL.

5.2.2 Transaction Tampering

A censor may attempt to disrupt Infranet tunnel communication by modifying HTTP requests and responses in ways that do not affect HTTP protocol conformance. For example, the adversary may change fields in HTTP request or response headers (e.g., changing the value of the `Date:` field), reorder fields within headers, or even remove or add fields. Infranet is resistant to these attacks because the tunnel protocol does not rely on modifications to the HTTP header.

As described in Section 4.1, the Infranet requester must present a unique user identifier with each HTTP request in order to be recognized across multiple HTTP transactions. A requester could send its user ID in a Web cookie with each HTTP request. However, if the censor removes cookies, we suggest maintaining client state by embedding the user ID (or some token that is a derivative of the user ID) in each URL requested by the client. Of course, to preserve the requester's deniability, the responder must rewrite all embedded links to include this client token.

The censor may modify the returned content itself. For example, it might insert or remove embedded links on a requested Web page or flip bits in requested images. Link insertion and deletion does not affect tunnel communications that use a codebook because the client sends messages upstream according to this codebook. Attacks on image content could disrupt the correct Infranet communication. Traditional robust watermarking techniques defend against such attacks. Infranet detects and blocks such disruptions by embedding the name of the served URL in each response.

5.2.3 Session Tampering

An adversary might attempt to disrupt tunnel communication by interfering with *sequences* of HTTP requests. A censor could serve a requester's visible HTTP request from its own cache rather than forwarding this request to the Infranet responder. To prevent such an attack, the Infranet requester must ensure that its HTTP requests are *never* served from a cache. One way to do this is to always request unique URLs. We consider this requirement fairly reasonable: many sites that serve dynamic content (e.g., CGI-based pages, webcams, etc.) constantly change their URLs. Another option is to use the `Pragma: no-cache` directive, although a censoring proxy will likely ignore this.

Alternatively, a censor might insert, remove, or reorder HTTP requests and responses. If a censor alters HTTP request patterns, the Infranet responder might see errors in the received message. However, Infranet responders include the name of the served URL in each response stream, thus enabling the requester to detect session corruption and restart the transmission. In the case of range-mapping, upstream transmission errors will be reflected in the split-strings returned by the responder. The requester can also defend against these attacks with error correction techniques that recover from the insertion, deletion, and transposition of bits [20].

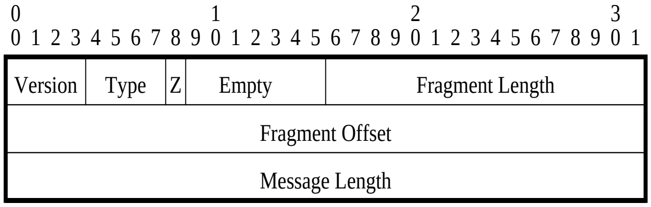
6 Implementation

Our implementation of Infranet consists of two components: the Infranet requester and the Infranet responder. The requester functions as a Web proxy and is responsible for modulating a Web browser's request for hidden content as a sequence of visible HTTP requests to the Infranet responder. The responder functions as a Web server extension and is responsible for demodulating the requester's messages and delivering requested content. The requester and responder utilize a common library, `libinfranet`, that implements common functionality, such as modulation, hiding, and cryptography. In this section, we discuss our implementation of the Infranet requester and responder, as well as the common functionality implemented in `libinfranet`.

6.1 Requester

We implemented the Infranet requester as an asynchronous Web proxy in about 2,000 lines of C++. We used `libtcl` for the asynchronous event-driven functionality. The requester sends visible HTTP requests to an Infranet responder, based on an initial URL at the responder, the responder's public key, and optionally an initial key IKEY.

The requester queues HTTP requests from the user's browser and modulates them sequentially. If the requester



<table>
<tr><td colspan="4">0 1 2 3</td><td colspan="4">4 5 6 7</td><td>8</td><td colspan="7">9 0 1 2 3 4 5</td><td colspan="16">6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td></tr>
<tr><td colspan="4">Version</td><td colspan="4">Type</td><td>Z</td><td colspan="7">Empty</td><td colspan="16">Fragment Length</td></tr>
<tr><td colspan="32">Fragment Offset</td></tr>
<tr><td colspan="32">Message Length</td></tr>
</table>

Figure 9. Responder header format.

knows about more than one responder, it can service requests in parallel by using multiple responders.

In our implementation, visible HTTP requests from the requester are generated entirely automatically. As discussed in Section 5.1, we could also allow the user to participate in link selection to provide increased covertness.

6.2 Responder

We implemented the Infranet responder as an Apache module, `mod_infranet` in about 2,000 lines of C++, which we integrated with Apache 1.3.22 running on Linux 2.4.2. The Apache request cycle consists of many phases, including URI translation, content-handling, and authorization [22]. Our `mod_infranet` module augments the *content handler* phase of the Apache request loop. The responder processes requests as it normally would but also interprets them as modulated hidden messages.

An Infranet responder must maintain state for a requester across multiple HTTP transactions. The current implementation of the responder uses Web cookies to maintain client state because this mechanism was simple to implement; in the future, we plan to implement the URL rewriting mechanism outlined in Section 5 because of its stronger defense against passive discovery attacks. The responder uses the REQUESTERTOKEN cookie, which contains a unique identifier, to associate HTTP requests to a particular requester. For each requester, the responder maintains per-requester state, including which FSM state the requester is in, the modulation function the requester is using, the shared secret SKEY, and message fragments for pending messages.

Figure 9 shows the header that the responder prepends to each message fragment. *Version* is a 4-bit field that specifies which version of the Infranet tunnel protocol the responder is running. *Type* specifies the type of message that the payload corresponds to (e.g., modulation function update, requested hidden content, etc.). *Z* is a 1-bit field that indicates whether the requested content in the payload is compressed with `gzip` (this is the case for HTML files but not images). *Fragment Length* refers to the length of the message fragment in the payload in bytes, and *Fragment Offset* specifies the offset in bytes where this fragment should be placed for reassembly of the message. *Message Length* specifies the total length of the message in bytes. Because upstream bandwidth is scarce and transmitting a header might create recognizable modulation patterns, the requester does not prepend a header to its messages.

6.3 Steganography and Compression

Upon receiving a request, the responder determines whether it can embed hidden content in the response. Currently, `mod_infranet` only embeds data in JPEG images. If the responder determines that it is capable of hiding information in the requested data, it uses `outguess` to embed the hidden information using SKEY. To reduce the amount of data that the responder must send to the requester, the responder compresses HTML files with `gzip` [5] before embedding them into images.

6.4 Cryptography

The Infranet requester generates the 160-bit shared secret SKEY using `/dev/random`. SKEY is encrypted using the RSA public key encryption implementation in the OpenSSL library [15]. This ciphertext is 128 bytes, which imposes a large communication overhead. However, because the ciphertext is a function of the length of the requester's public key, it is difficult to make this ciphertext shorter. One option to ameliorate this would be to use an implementation of elliptic curve cryptography [13].

7 Performance Evaluation

In this section, we examine Infranet's performance. We evaluate the overhead of the tunnel setup operation and the performance of upstream and downstream communication. Finally, we estimate the overhead imposed by `mod_infranet` on normal Web server operations.

All of our performance tests were run using Apache 1.3.22 with `mod_infranet` on a 1.8 GHz Pentium 4 with 1 GB of RAM. For all performance tests, we ran an Infranet requester and a Perl script that emulates a user's browser from the same machine.

7.1 Tunnel Setup

Tunnel setup consists of two operations: upstream transmission of SKEY encrypted with the responder's public-key, and downstream transmission of $\mathcal{U}_{tunnel}$. In our implementation, SKEY is 160 bits long and the corresponding ciphertext is 128 bytes, proportional to the length of the responder's public key. Transmission of $\mathcal{U}_{tunnel}$ is equivalent to a single document transmission. The transmission of an initial modulation function, if one is used, requires one additional downstream transmission.

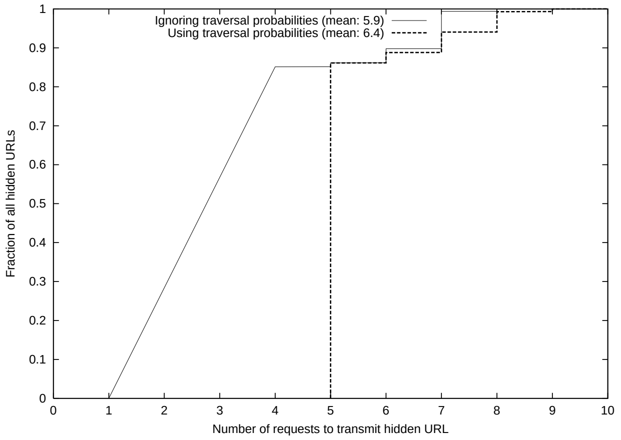


Figure 10. Number of visible HTTP requests required to modulate hidden URLs with and without link traversal probabilities for range-mapping, assuming 8 links per page ($k = 8$). The expected number of visible HTTP requests required to modulate a message is independent of whether the link traversal probabilities are used.

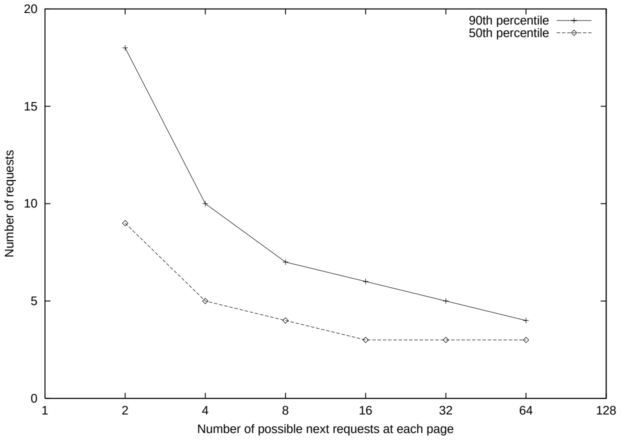


Figure 11. The median and 90th percentile of number of requests to modulate a message is very small even for small numbers of outgoing links (k) on each page.

7.2 Upstream Communication

An important measure of upstream communication performance is how many HTTP requests are required to modulate a typical message. We focus our evaluation on the range-mapping scheme described in Section 4.2.

We evaluated the performance of range-mapping using a Web proxy trace containing 174,100 unique URLs from the Palo Alto IRCache [8] proxy on January 27, 2002.[3] When we weight URLs according to popularity, the most popular 10% of URLs account for roughly 90% of the visible HTTP traffic. This is significant, since modulating the most popular 10% of URLs requires a small number of visible HTTP requests.

A requester can achieve statistical deniability by patterning sequences of HTTP requests after those of innocuous Web clients. As described in Section 4.2.3, this is done by assigning split-string ranges in $\mathcal{C}$ according to the pairwise link traversal probabilities $\mathcal{P}$. To evaluate the effect of using the distribution $\mathcal{P}$, we modulated 1,740 requests from $\mathcal{C}$, both using and ignoring $\mathcal{P}$, assuming 8 outgoing links per page.

Figure 10 shows that assigning ranges based on link traversal probabilities does not affect the expected number of visible HTTP requests required to modulate a hidden request. This follows directly from properties of arithmetic codes [31]. In both cases, over half of hidden messages required 4 visible HTTP requests, and no more than 10 requests were needed for any message. Therefore, using traversal probabilities to determine the size of ranges in $\mathcal{C}$ provides statistical deniability without hurting performance.

Setting link traversal probabilities to $1/k$, we evaluated the effect of the number of links on a page, k, on upstream communication performance. Figure 11 shows that 90% of messages from $\mathcal{C}$ can be modulated in 10 visible HTTP requests or fewer, even for k as small as 4.

In the trace we used for our experiments, the binary entropy of the frequency distribution of requested URLs is 16.5 bits. Therefore, the expected number of requests required to transmit a URL from $\mathcal{C}$ is $\left\lceil \frac{16.5}{\lg k} \right\rceil + 2$. The empirical results shown in Figure 11 agree with this analytical result.

To evaluate the performance of range-mapping on real sites, we modulated hidden requests while varying k and p according to the browsing patterns observed on a real Web site. We analyzed the Web access logs for `nms.lcs.mit.edu` and `pdos.lcs.mit.edu` to generate values for k and p for each page on these sites. We then observed the number of visible HTTP requests required to transmit 1,740 messages from $\mathcal{C}$. Results from this experiment are shown in the table below. The number of requests required to modulate a hidden message is slightly higher than in Figure 11, since k varies for a real Web site.

SITE	k_{avg}	MEDIAN	90%
`nms.lcs.mit.edu`	8	6	10
`pdos.lcs.mit.edu`	12	4	6

[3] These traces were made available by National Science Foundation grants NCR-9616602 and NCR-9521745, and the National Laboratory for Applied Network Research.

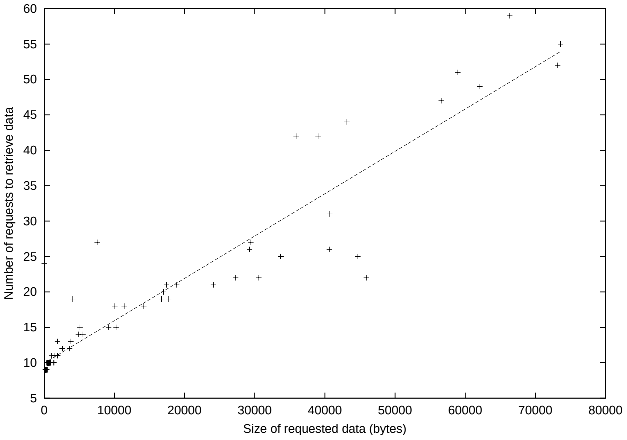


Figure 12. Number of requests required to retrieve data of various sizes. Each visible HTTP request contains approximately 1 kB of a hidden message.

7.3 Downstream Communication

Figure 12 shows the number of HTTP requests that an Infranet requester must make to retrieve hidden messages of various sizes. Each visible HTTP response contains approximately 1 kB of a hidden message. The amount of data that can be embedded in one visible HTTP response depends on two factors—the compression ratio of the message and the amount of data that can be steganographically embedded into a single image. Thus, depending on the requested document and the images used to embed the hidden response, the number of visible HTTP requests required to send a given amount of hidden data may vary.

We microbenchmarked the main operations involved in content preparation. First, we ran a microbenchmark of `outguess` in an attempt to determine the rate at which it can embed data into images. Our measurements indicate that `outguess` embeds data into an image at a rate of 20 kB/sec, and that the time that `outguess` takes to embed data is proportional to the size of the cover image.

We also ran microbenchmarks on `gzip` to determine its computational requirements, as well as the compression ratios it achieves on typical HTML files. We fetched the `index.html` page from 100 popular Web sites that we selected from Nielsen Netratings [14] and IRCache [8]; the median file size for these files was 10 kB. `gzip` compressed these HTML files to as much as 12% of their original size; in the worst case, `gzip` compressed the HTML file to 90% of its original size. In all cases, compression of an HTML file never took more than 20 milliseconds.

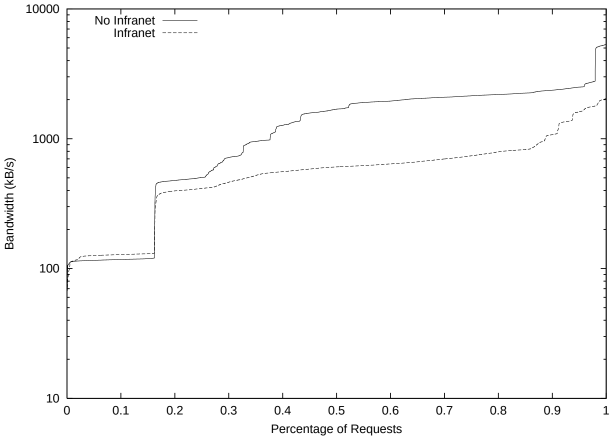


Figure 13. The overhead of running Infranet on Web server performance. For disk bound workloads, an Infranet responder performs comparably to an unmodified Apache server.

7.4 Server Overhead

To ensure plausible deniability for Infranet requesters, Infranet responders *always* embed random or requested data in the content they serve. Because the responder must make no distinction between Infranet requesters and normal Web clients, an Infranet-enabled Web server incurs additional overhead in serving data to *all* clients.

Therefore, to determine the performance implications of running an Infranet responder, we submitted an identical sequence of requests to an Infranet-enabled Apache server and an ordinary Apache server. The request trace contains a sequence of visible HTTP requests that were generated by using an Infranet requester to modulate the requests in the set of 100 popular Web sites.

Figure 13 shows the additional overhead of running Infranet. Because the server must embed every image with data, regardless of whether it is serving an Infranet request or not, running Infranet incurs a noticeable performance penalty. 16% of all requests served on an Infranet responder achieved 300 kB/s or less, and 89% of requests were transferred at 1 MB/s or less. In contrast, 62% of requests on a normal Apache server were delivered at 1 MB/s or greater. Nevertheless, for disk bound workloads, an Infranet responder performs comparably to a regular Apache server. Bandwidth achieved by Infranet never drops below 32% of that achieved by an Apache server running without Infranet, and is within 90% of Apache without Infranet for 25% of requests. Our current implementation has not been optimized, and we believe we can reduce this overhead. One way to do so might be to pre-fetch or cache commonly requested censored content. The overhead of running `outguess` could

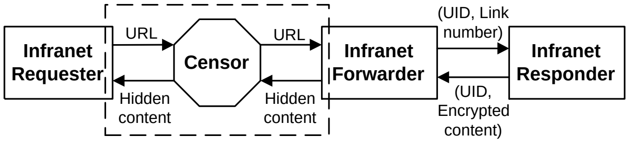


Figure 14. An improved architecture separates the forwarding and decoding of hidden messages in both directions. This allows a potentially untrusted forwarder to service requests and serve hidden content. The UID serves to demultiplex requesters.

be reduced by pre-computing the least-significant bits of the DCT for each image on the site.

8 Enhancements

To this point, we have assumed that Infranet requester software can be distributed on a physical medium, such as a CD-ROM. However, this distribution mechanism is slow, provides evidence for culpability, and is much easier for oppressive governments to control than electronic distribution. Thus, a future enhancement would allow clients to download the Infranet requester software over Infranet itself, essentially *bootstrapping* the Infranet requester.

The architecture we presented in Section 3 does not protect against an impersonation attack whereby a censor establishes an Infranet responder and discovers requesters by identifying the Web clients that send meaningful Infranet requests. Figure 14 shows an improved system architecture, where a requester forwards visible HTTP requests to a trusted responder through a potentially untrusted *forwarder*, such that only the responder can recover the hidden request. Responders fetch and encrypt requested content and return it to the requester through a forwarder, which hides the encrypted content in images. This scheme provides several improvements. First, blocking access to Infranet becomes more difficult, because a requester can contact any forwarder, trusted or untrusted, to gain access. Second, the censor can become a forwarder, but it is much more difficult for the censor to become a trusted responder.

In the current tunnel communication protocol, an Infranet requester and responder take turns transmitting messages. It is conceivable that a scheme could be devised whereby the HTTP requests used to fetch the requested content could also be used to transmit the next hidden message, thus *interleaving* the retrieval of hidden information with the transmission of the next hidden message.

Since there are many conceivable ways of performing modulation and hiding, it is likely that there will be many different versions of the tunnel communication protocol. Tunnel setup should be amended to include version agreement, such that if some particular aspect of the tunnel protocol in a given version is found to be insecure, the requester and responder can easily adapt to run a different version.

9 Conclusion

Infranet enables users to circumvent Web censorship and surveillance by establishing covert channels with accessible Web servers. Infranet requesters compose secret messages using sequences of requests that are hard for a censor to detect, and Infranet responders covertly embed data in the openly returned content. The resulting traffic resembles the traffic generated by normal browsing. Hence, Infranet provides both access to sensitive content and plausible deniability for users.

Infranet uses a tunnel protocol that provides a covert communication channel between requesters and responders, modulated over standard HTTP transactions. In the upstream direction, Infranet requesters send covert messages to Infranet responders by associating additional semantics to the HTTP requests being made. In the downstream direction, Infranet responders return content by hiding censored data in uncensored images using steganographic techniques. While downstream confidentiality is achieved using a session key, upstream confidentiality is achieved by confidentially exchanging a modulation function.

Our upstream and downstream protocols solve two independent problems, and each can be tackled separately. Although our protocol is optimized for downloading Web pages, it actually provides a channel for arbitrary two-way communication; for example, Infranet could be used to carry out a remote login session.

Our security analysis showed that Infranet can successfully circumvent several sophisticated censoring techniques, ranging from active attacks to passive attacks to impersonation. Our performance analysis showed that Infranet provides acceptable bandwidth for covert Web browsing. The range-mapping algorithm for upstream communication allows the requester to innocuously transmit a hidden request in a number of visible HTTP requests that is proportional to the binary entropy of the hidden request distribution. We believe that the widespread deployment of Infranet responders bundled with Web server software has the potential to overcome various increasingly prevalent forms of censorship and surveillance on the Web.

Acknowledgments

We thank Sameer Ajmani and Dave Andersen for several helpful discussions, and Frank Dabek, Kevin Fu, Kyle Jamieson, Jaeyeon Jung, David Martin, and Robert Morris for useful comments on drafts of this paper.

References

[1] M. Adler and B. Maggs. Protocols for asymmetric communication channels. In *Proceedings of 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, Palo Alto, CA, 1998.

[2] Anonymizer. `http://www.anonymizer.com/`.

[3] I. Clarke, O. Sandbert, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[4] R. Dingledine, M. Freedman, and D. Molnar. The Free Haven Project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[5] gzip. `http://www.gzip.org/`.

[6] F. Hartung and B. Girod. Digital watermarking of raw and compressed video. In *Proc. European EOS/SPIE Symposium on Advanced Imaging and Network Technologies*, pages 205–213, Berlin, Germany, October 1996.

[7] A. Hintz. Fingerprinting websites using traffic analysis. In *Workshop on Privacy Enhancing Technologies*, San Francisco, CA, April 2002.

[8] IRCache. `http://www.ircache.net/`.

[9] B.W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[10] J. Lee. Companies compete to provide Saudi Internet veil, November 19, 2001. `http://www.nytimes.com/2001/11/19/technology/19SAUD.html`.

[11] D. Martin and A. Schulman. Deanonymizing users of the SafeWeb anonymizing service. In *Proc. 11th USENIX Security Symposium*, San Francisco, CA, August 2002.

[12] P. Meller. Europe moving toward ban on Internet hate speech, November 10, 2001. `http://www.nytimes.com/2001/11/10/technology/10CYBE.html`.

[13] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, MA, 1993.

[14] Nielsen Netratings' Top 25. `http://pm.netratings.com/nnpm/owa/NRpublicreports.toppropertiesweekly`, November 25, 2001.

[15] OpenSSL. `http://www.openssl.org/`.

[16] Outguess. `http://www.outguess.org/`.

[17] N. Provos. Defending against statistical steganalysis. In *Proc. 10th USENIX Security Symposium*, Washington, D.C., August 2001.

[18] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1:66–92, November 1998. `http://www.research.att.com/projects/crowds/`.

[19] SafeWeb. `http://www.safeweb.com/`.

[20] L. J. Schulman and D. Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. In *Symposium on Discrete Algorithms*, pages 669–674, 1997.

[21] Squid Web Proxy Cache. `http://www.squid-cache.org/`.

[22] L. Stein et al. *Writing Apache Modules with Perl and C: The Apache API and mod_perl*. O'Reilly and Associates, Sebastopol, CA, March 1999.

[23] Stunnel—universal SSL wrapper. `http://www.stunnel.org/`.

[24] Zero-Knowledge Systems. Freedom WebSecure. `http://www.freedom.net/products/websecure/`.

[25] P. Syverson, M. Reed, and D. Goldschlag. Onion routing access configurations. In *DARPA Information Survivability Conference and Exposition*, pages 34–40, Hilton Head Island, SC, January 2000. `http://www.onion-router.net`.

[26] The Cult of the Dead Cow (cDc). Peekabooty. `http://www-peek-a-booty.org`.

[27] Triangle Boy. `http://fugu.safeweb.com/sjws/solutions/triangle_boy.html`.

[28] Voice of America. `http://www.voa.gov/`.

[29] M. Waldman and D. Mazières. Tangler: A censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, November 2001.

[30] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, Denver, CO, August 2000.

[31] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, February 1987.