Traffic Analysis of SSL Encrypted Web Browsing

Heyning Cheng and Ron Avnur
(heyning@bmrc.berkeley.edu, ronathan@cs.berkeley.edu)

Abstract

The SSL protocol, an application-layer mechanism widely used for encrypted Web browsing, was not designed to address traffic analysis attacks. We investigate the threat to privacy posed by such attacks and consider possible defenses. We implement a prototype of a traffic analysis attack and employ it to identify the pages visited by users browsing a Web site. Numerical models and simulations are used to predict the effectiveness of traffic analysis on various sites, as well as the efficacy of several possible defenses. Our results show that an attack using simple techniques can identify the pages visited with very high accuracy, and suggest that defenses exist which may provide some degree of privacy protection in many cases.

1. Introduction

The World Wide Web is quickly becoming everyone's first choice for information retrieval. From reading the local newspaper to checking stock quotes, surfing the Web has become a vital tool. Unfortunately, it can also expose private information to malicious users. With easy-to-obtain traffic-watching applications, a hacker can view the data between a server and an unsuspecting browser. For example, the hacker could obtain information about which stock the user is interested in.

By popular demand, Web servers and browsers provide a protocol stack for protecting the privacy of traffic data, known as HTTP Secure, or HTTPS. Application data is encrypted by the Secure Socket Layer (SSL) before being handed down to the TCP/IP protocol layer. For each secure session, the browser and server follow a handshaking protocol to agree on the encryption and decryption method as well as the shared keys. Data streams are encrypted in both directions.

Unfortunately, the new encryption-based protocol has given users a false impression of the confidentiality of Web surfing. [8] raises the issue of traffic analysis. Despite the encryption of application data, we will show that it is quite simple to identify exactly which page a user downloads by inspecting the TCP/IP packet headers, which are not encrypted because of obvious routing needs. (This is NOT a paper about cracking the SSL encryption algorithm, which we assume is perfect.) Packet headers identify the destination and source addresses and the size of the payload data, along with other information. The address and packet size information is usually sufficient to identify which page a target user downloads. Moreover, the applications needed to perform this analysis are readily available on the Web: a packet-sniffing application and a Web crawler are all that is needed to set up shop.

The highly random size distribution of files commonly found on Web sites makes traffic analysis easy. HTML files can be of arbitrary length; thus it is very unlikely that two randomly selected files will have exactly the same size. In fact, the size of a given file is often unique among all such files at a particular site, so that an adversary who knows the size of a page downloaded to a browser can easily identify the page being browsed. Figure 1 contains a histogram of the sizes of HTML files for a site containing approximately 500 pages. Only 10% of the pages have non-unique sizes; in all but one of those cases, there are only two pages with the same size, while in the last case there are three. Inline images follow a similarly random size distribution: such images are almost always stored and transferred in compressed form, and the compressed size of an image depends on its actual content as well as its spatial dimensions. As we will explain later, the sizes of HTML and image files transferred can easily be determined by watching encrypted traffic. Other clues are also available for traffic analysis.

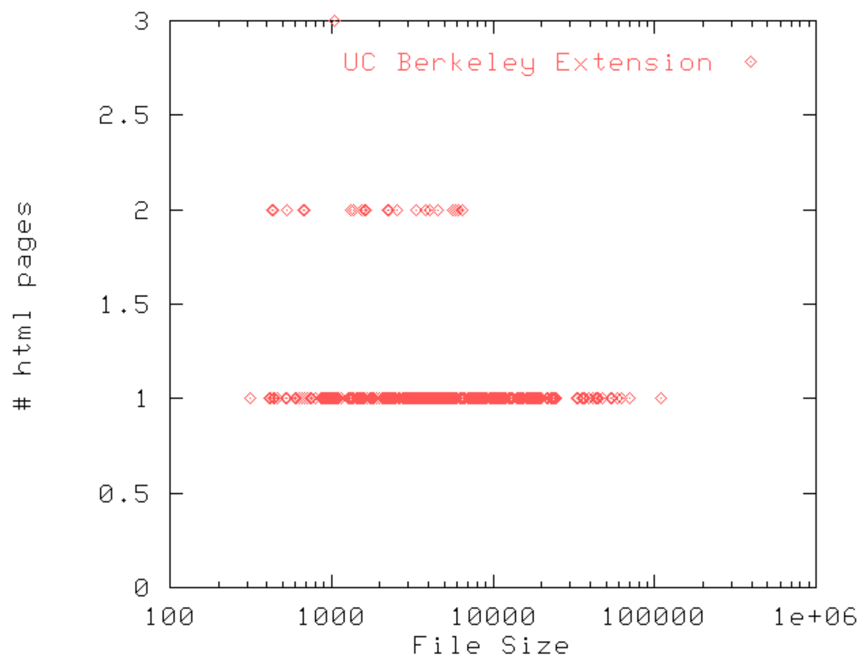

Figure 1. A histogram of HTML page sizes of all pages served by U.C. Berkeley Extension Web Site [7]. Note that pages very rarely share the same size – even then, the number of same-sized pages is small.

The rest of this paper is structured as follows. In Section 2, we discuss basic techniques for mounting a traffic analysis attack. Section 3 explores possible defense strategies and counterattacks. Section 4 describes our implementation. Section 5 presents empirical and theoretical results. Section 6 concludes.

2. Traffic Analysis Techniques

This section is organized as follows: Section 2.1 outlines the relevant protocol issues; Section 2.2 describes techniques for extracting useful data from sniffed network traffic; and Section 2.3 discusses methods for identifying Web pages using the extracted data.

2.1. Protocol Issues

A detailed understanding of the protocols used for Web browsing is necessary to mount an effective traffic analysis attack. HTTP follows a simple procedure for downloading Web pages. First, the client browser sends a request for a page. The server then delivers a stream of IP packets containing the HTML code for the page. This code contains references to other embedded objects, such as images, which the browser must fetch from the server. After receiving and parsing the HTML, the browser issues requests for all of the embedded objects. If there are multiple embedded objects, several objects are requested concurrently. These embedded images also arrive as streams of IP packets.

A browser-server interaction may look like this:

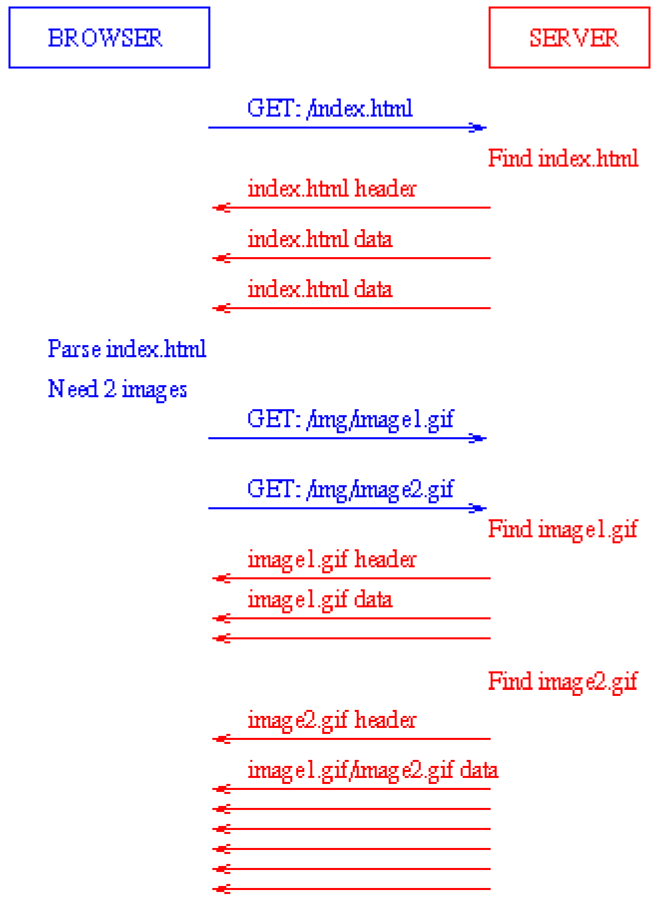

Figure 2. Example session between a browser and server. Note: each arrow represnts one IP packet. ACKs not shown for brevity.

Only the HTML page is sent during the interval between the first request made and the remaining requests. Responses coming after the second request all correspond to the transfer of embedded objects. The protocol makes it simple to differentiate between HTML data and object data transfer. In fact, even more size information is available. A browser limits the number of concurrent outstanding requests it permits. In order to differentiate between packets that arrive for any one of the expected objects, a transaction identifier is associated with each request and reply. It is therefore possible to extract the size of each object downloaded: it is the amount of data received with a specific transaction identifier between the original request for an object and a subsequent request for another object using the same transaction identifier or a timeout on the transaction.

One variable which may complicate traffic analysis is the user browsing rate. A user may select a link, enter a new URL, or stop data transfer before all of the files for a Web page have been downloaded. He/she may also configure the browser not to request certain types of embedded objects. Thus the total size and number of images sent over the network may not reflect the actual properties of the page. In this discussion, we limit embedded objects to images only and assume that they are fully downloaded. In addition, we assume that each page requested is completely downloaded before the next request is issued. Without these assumptions, the attack can only identify pages by their HTML code size. However, we will later show that this is not a serious limitation.

2.2. Extracting Information from Sniffed Traffic

Based on the information available in the TCP/IP packet headers, an attacker must determine the sizes of the HTML data and embedded objects for each page. A packet-sniffing tool is required to extract the packet header information from Web traffic involving the target browser. If the attacker has full access to a machine on the same network as the target, the *tcpdump* tool can be used. Below is an example of the packet-sniffing output for the same scenario described in Figure 2:

```
herland.CS.Berkeley.EDU.1460 > amber.Berkeley.EDU.4243: 260
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1460: 174
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1460: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1460: 1422
herland.CS.Berkeley.EDU.1463 > amber.Berkeley.EDU.4243: 269
herland.CS.Berkeley.EDU.1462 > amber.Berkeley.EDU.4243: 269
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 174
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 175
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1463: 1099
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1460
amber.Berkeley.EDU.4243 > herland.CS.Berkeley.EDU.1462: 1450
```

Figure 3. Sample output from a *packet sniffer*. The output displays which machines communicated and how much data was transferred.

This output format displays a packet transfer on each line of text. The source and destination of each packet are ordered from left to right, with the port used on that machine following the machine name. The size of each packet is listed at the end of the line; note that the maximum IP packet size in this scenario is 1460 Bytes.

In Figure 3, we can see that *herland* makes three requests. Between the first and second requests it receives some data from *amber*. The first packet received (174 bytes) is the HTML header packet. The next two packets received on that port consist of the HTML data that was requested. The second and third requests can not be made until *herland* receives the entire HTML file and parses it. After parsing the HTML, *herland* issues requests for the two images embedded in the page. Since the delivery of each image does not depend on the delivery of the other, *herland* issues both requests concurrently. Notice that *amber* transmits data for the two images concurrently as well. We identify ports 1463 and 1462 on *herland* as the destinations for image1.gif and image2.gif respectively. The first packet received on each port consists of header information. Since header packets are smaller than the maximum IP packet size, they are easy to distinguish from the first data packet. The remaining packets consist of image data. Finally, we should notice that the last packet delivered for each image is smaller than the maximum size; thus it is easy to locate the end of the transfer. Based on this analysis, we can conclude that *herland* received an 1882 byte HTML file from *amber*, along with two embedded images of 8,935 and 8,750 bytes respectively. By browsing all pages available on *amber,* the adversary can identify all possible pages that match the data size characteristics extracted from the sniffed traffic.

To reduce browsing latency and network traffic, most browsers maintain a cache of recently browsed files on local memory and disk. Requests are not issued for files residing in the cache. In practice, no useful assumptions can be made about the contents of the cache when a browsing session is first sniffed; any subset of the HTML and object components of a given page may be cached. In general, we can not guarantee that the first file requested for a page is the HTML code. However, as we explained in Section 2.1, HTML files cannot be transmitted concurrently with other files and thus remain identifiable within the traffic. On the other hand, the total quantity of embedded object traffic may not reflect the total size of all embedded objects on the page. Therefore when caching is turned on, our policy is to identify pages solely based on their HTML size. This strategy should work well except in the infrequent cases where the HTML is cached and only one image needs to be retrieved, in which case the image may be mistaken for HTML code.

2.3. Page Identification

The next task is to identify the pages visited using the information extracted from the sniffed traffic. We assume that the relevant statistics for each page in the site of interest are stored in a database (see Section 4). The simplest identification scheme is to query the database for all pages which exactly match the HTML size and total object size for the given sample, and choose randomly from the resulting set. If the Web site is not unusually large, often there is only one exact match. Since HTML size and object size are largely independent, the key *<HTML size, object size>* defines each page in a two-dimensional feature space, further reducing the likelihood of a false match.

Some queries will return more than one match. However, the nature of Web browsing tends to place additional constraints on the set of likely matches for a given query. Web surfers browsing a site almost never visit a sequence of randomly selected pages; they navigate mostly by following links, sometimes using Back and Forward browser buttons and bookmarks. Assuming that our database contains the link structure of the Web site, we could use the preceding and subsequent data samples as contextual information to help us identify the correct page among a set of candidates. In general, the traffic analysis problem can be modeled as a hidden Markov model, in which the actual pages visited correspond to hidden states, the hyperlinks correspond to state transitions, and the data samples correspond to outputs. The problem is to determine the most likely sequence of pages (states) given a sequence of samples (outputs); this is commonly solved using the Viterbi algorithm for computing the most likely explanation [4].

In reality, Web surfers often follow at most several links before backtracking, leaving the Web site, or navigating by other means. Thus we suspect that using a general hidden Markov model formulation would be impractical; the most useful contextual information is likely to be contained in the previous sample and the very next sample. Accordingly, we devised a link analysis algorithm which uses a window of three samples. The algorithm is based on the assumption that each link on a page is followed with equal probability; while this is a very inaccurate premise, it will have to suffice in the absence of detailed knowledge of browsing patterns for the particular site. We shall denote the previous, current, and next samples as S_{-1}, S_0, and S_1. By querying the database on *<HTML size, object size>* for S_{-1}, S_0, and S_1, we can determine the set of candidate pages C_{-1}, C_0, and C_1 for the previous, current, and next sample respectively. Assume that the previous page is randomly selected from C_{-1}. We then assign a score to each page p in C_0 by computing the probability that by randomly following links, we would first visit p followed by any of the pages in C_1. The page receiving the highest score among all pages in C_0 is selected as our guess. Note that $Score(p) = InScore(p) * OutScore(p)$, where $InScore(p)$ is the probability that p is visited immediately following a page randomly selected from C_{-1}, and $OutScore(p)$ is the probability that p is immediately followed by any of the pages in C_1. If the surfer navigates without following a link, there may not be any link from the actual previous page visited to the actual current page, or from the actual current page to the actual next page. In this case, there is no page p in C_0 for which both $InScore(p)>0$ and $OutScore(p)>0$; thus we select the page p in C_0 that has the greatest value of either $InScore(p)$ or $OutScore(p)$.

There are several additional techniques worth mentioning that we did not implement in our protoype. Detailed object information might be used to improve the identification algorithm. In this scenario, an exact match exists between the data and a page only if there is a one-to-one correspondence that matches each object size extracted from the sniffed traffic with an embedded object of the same size. This is more likely to be useful if the site contains a large number of different images and if there is wide variation in the sets of images that are embedded in different pages. Hyperlinks that point to pages on other Web sites can also provide useful clues, even if the attacker does not know anything about the internal structure of those other sites. Since TCP/IP header information is not encrypted, a sniffer operating near the client can identify the Web site visited by the client after leaving the target site; this would suggest that the last page visited on the target site contains a link to that external site. The techniques employed in our prototype can also be used for dynamic Web pages which have a fixed size. Dynamic pages with several possible sizes may be treated as a set of separate pages, one for each size. However, some dynamic pages may generate content of arbitrary size, in which case traffic analysis based on size information is impossible.

When browser caching is enabled, consecutive data samples may not correspond to consecutively visited pages; this potentially reduces the utility of the link structure algorithm. As we explained in Section 2.2, we cannot generally derive useful information about the total object size or number of objects from the sniffed traffic (though it may be possible to query the database for pages containing some embedded objects whose sizes match the sizes of any objects that were transferred). Thus we use the link structure algorithm described above with only one modification: the database is queried on HTML size only, and object information is ignored.

3. Defenses

There exists a wide range of potential defenses against traffic analysis attacks. We identify three general classes of defenses: modification of the actual protocols; restructuring of Web sites; and intermediate proxy servers which interpose between clients and servers. It is important to evaluate each defense strategy with respect to their effectiveness under various conditions, impact on quality of service, vulnerability to counterattack, and ease of adoption. Furthermore, defenses almost always impose some degree of overhead in terms of computing resources, network bandwidth, or latency. Therefore, the user should have the option of choosing among different levels of defenses (or no traffic analysis defense, for that matter), depending on the desired tradeoff between security and performance.

3.1 Protocol Modifications

One defense strategy is to modify the security protocols so that the identity of the Web pages visited cannot be easily determined from the sniffed network traffic. We will first need to consider architectural issues. While SSL is currently used mostly for confidential Web browsing, it is designed to be usable with different application-layer protocols. Since many defenses against traffic analysis are necessarily application-specific, adding such defenses to the SSL protocol may reduce its future flexibility. Thus, we believe that traffic analysis defenses for Web browsing should be encapsulated in a separate security protocol layer to mediate between the HTTP and SSL layers. This would in principle make it possible to add additional defenses in the future without altering existing protocol standards, or to modify HTTP without making changes to SSL. Implementation of protocol-level defenses will require revisions to commercial Web browsers. Since Netscape and Microsoft together account for almost all of the browser market, this will only require modifying a small number of products and could be accomplished in a reasonably short period of time. We describe several specific protocol-level defenses below.

Random padding. One simple defense is to pad each file with a random number of bytes. To maximize randomness, the number of padding bytes should follow a uniform probability distribution from zero to some maximum value. With random padding, our traffic analysis attack would have to query the database for a range of page sizes, increasing the likelihood of multiple matches. Since link structure supplies additional identifying clues, random padding is likely to be effective only if most HTML size queries match a substantial number of documents. This approach is most effective for Web sites having a large number of pages of similar size, and least useful for concealing the identity of unusually large documents. Since random padding imposes bandwidth overhead proportional to the amount of padding per page, padding every page to at least the size of the largest document is generally prohibitive. SSL currently supports random padding only for the block cipher modes [8].

Constant size packets. Alternatively, we could impose a maximum IP packet size and pad all HTML files and objects so that only full size packets are sent. Thus the number of bytes sent for a Web page is always a multiple of the maximum packet size, as is the case with block ciphers. As in the case of random padding, this increases the probability that a page size query will return multiple matches. The bandwidth overhead incurred is roughly proportional to the packet size. Like random padding, using constant size packets is most effective when there are many pages with similar sizes, but does not hide particularly large pages. However, unlike random padding, the relationship between the Web page sent and the number of bytes transferred is deterministic. Since the degree of protection is a function of the number of matches per query rather than the amount of padding, extra bytes may be used inefficiently if a Web page contains a large number of objects which all have to be padded to full packets. The distribution of HTML sizes for the Unex Web site after padding to 1460-byte packets is shown in Figure 4. Constant size packets may also be combined with random padding, though this hybrid approach is unlikely to provide better protection than random padding alone for the same amount of overhead.

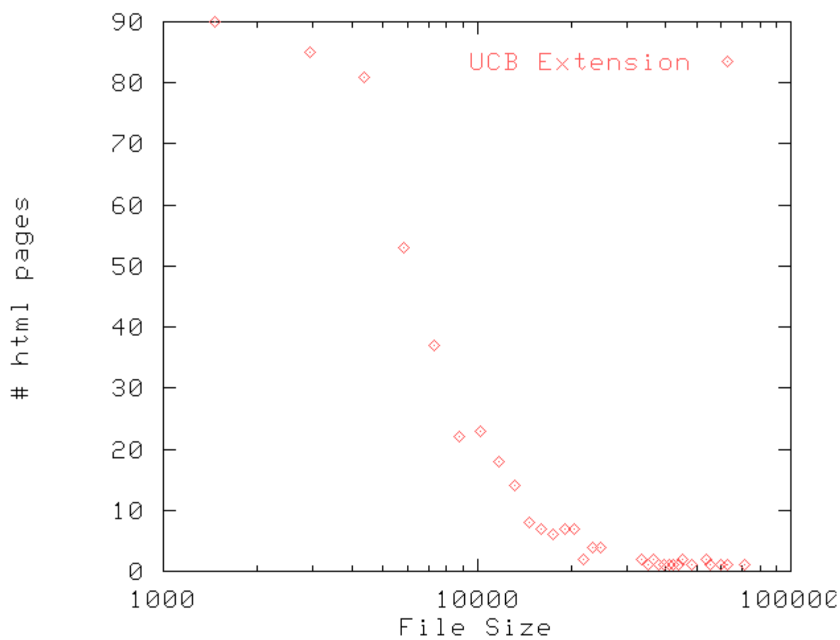

Figure 4. A histogram of HTML page sizes of all pages served by U.C. Berkeley Extension Web Site [7] where each file is padded to the nearest 1460 Byte boundary. Note that as page sizes grow, the number of duplicates considerably drops off.

Background traffic. The security protocol could also inject spurious packets to confuse traffic analysis attacks. While sending a continuous stream of background traffic would be prohibitively wasteful of bandwidth, selective addition of spurious traffic may provide some protection with reasonable bandwidth overhead. For instance, the judicious insertion of extra header packets and partially full packets from the server, along with extra requests from the client, might prevent attackers from separating packets containing HTML data from those containing object data. Thus sniffers would have access to the approximate total size of each uncached Web page visited, but would not be able to determine the HTML size, total object size, or number of objects sent. Since browser caches are almost always used, it is often impossible to identify a page based on the total number of bytes sent without knowing which files are currently cached. For this strategy to be effective, however, the size and timing of extra packets must be sufficiently random that the spurious packets cannot be reliably identified using statistical techniques. These measures also cannot be expected to conceal the identity of a particular Web page that is much larger (in terms of total size) that any of the other pages at the site. It may be possible to send a large page as several smaller installments, separated by random delays of several seconds or longer, in order to simulate visits to multiple distinct Web pages. However, this may be highly disruptive to the viewer.

3.2. Web site Modifications

An alternative approach is to modify the structure of Web sites to make traffic analysis more difficult. For example, a large Web page could be broken into several smaller pages connected by hyperlinks. While this may inconvenience viewers and requires the discretion of the Web site designer, it may be the most effective defense for very large, "pathological" pages. It is also easy to pad the size of a page without changing its appearance by inserting extra spaces into HTML tags, including lengthy metatags, embedding blank images of the background color, or recoding compressed images at higher resolution. It would not be difficult to write a program to make these changes automatically. If limited protocol-level defenses such as random padding are available, it is not necessary for all pages at a site to be of the same size, as long as for any given Web page there are many other pages of similar size. A key disadvantage of this approach is that it would require modifications to be made at thousands of separate Web sites, with significant human effort needed in some cases. Moreover, most Web site designers are not familiar with the problem of traffic analysis of encrypted communications and are unlikely to recognize the need for site changes in the near future. For these reasons, we do not realistically expect site-based defenses to be implemented, except possibly at commercial sites whose business interests rely on strong assurances of privacy.

3.3 Web Proxies

Another defensive option is for the client and the server to communicate indirectly through an intermediate proxy. The client sends requests to the proxy, which forwards them on to the server; the server sends data to the proxy, which in turn sends it to the client. Thus, an eavesdropper cannot identify the Web site browsed by a particular client by sniffing either the client-proxy connection or the proxy-server connection. This approach can provide much broader privacy protection than any of the other defenses since Web surfers often have reasons to conceal the identity of the sites they visit as well as the actual pages browsed. The Anonymizer [1] is one example of a Web proxy that provides an anonymous browsing service. Traffic analysis protection is also implicitly provided by other proxies such as TranSend [6], which is designed to transcode images and HTML documents to smaller representations for clients with low-bandwidth network connections. While the main disadvantage of using proxies is increased latency, there is evidence that a well-engineered proxy server can easily meet the realtime demands of Web surfing; as a case in point, browsing with TranSend actually reduces latency for many types of clients.

Proxy defenses are vulnerable to a potential counterattack: an attacker who simultaneously sniffs both the client-proxy and proxy-server connections can use size and timing information to determine correspondences between the packets seen by the two sniffers, and thus identify the Web site browsed by the client. Proxies that transcode data in predictable ways are still susceptible to this attack; a proxy that sends only constant-size packets and services multiple clients simultaneously is less vulnerable. This is a technically difficult attack to use against a targeted client, as it typically requires the ability to sniff multiple network links which may be geographically distant. Anonymous email services exist which provide far more sophisticated privacy features, such as chaining of multiple remailers and the use of random delays. The realtime requirements of Web browsing generally preclude the use of these techniques; however a chain of several proxies may provide a reasonable tradeoff for especially privacy-conscious users. In general, we believe that proxy defenses provide sufficient protection against traffic analysis for the majority of SSL users; they are also the only defenses currently available. Widespread adoption of proxy services would require maintaining a large number of servers with substantial computing resources and resolving the associated economic issues.

4. Implementation

We implemented an prototype attack in order to test out traffic analysis methods and defenses. We used two off-the-shelf applications to build the prototype: *tcpdump* as a packet sniffer, and *webcopy* to crawl Web pages. What follows is a description of how to mount an attack using our prototype:

The *webcopy* spider [9] is used to produce a local mirror copy of an external Web site. The local copy is traversed in order to generate a database of all the files belonging to the site, along with their sizes. For each HTML page, a crude parser is used to extract hyperlinks and image references, which are resolved to refer to other records in the database. Thus, the database entry for each HTML file includes a list of references to other HTML files and a list of references to object files. All other Web pages features, including dynamic pages and Java applets, are ignored. Hyperlinks to other web sites are excluded. The *tcpdump* utility is used to sniff IP packets passing through the client computer. Inbound and outbound packets using the HTTP port are parsed to extract the HTML size, total object size, and total number of objects for each Web page visited. This data is used to query the database compiled for the site. If multiple matches are returned, the link structure algorithm described in Section 2.3 is used to resolve ambiguities.

Our experience indicates that the main components of the traffic analysis attack are amenable to a relatively simple implementation: we wrote approximately 2000 lines of C code and did not leverage any existing software other than *webcopy* and *tcpdump*. Both the HTML parser and the network traffic parser are incomplete due to time constraints. We believe that we were able to perform useful study on many of the core issues concerning the traffic analysis problem despite these limitations. As the result will show, even with our limited parsing we produced a highly successful attack.

5. Results

We conducted a series of theoretical and empirical studies to determine the effectiveness of our traffic analysis attack. We selected several remote Web sites as testbeds for our experiments; we used ordinary unencrypted Web sites since they are more abundant than encrypted sites and exhibit nearly identical traffic patterns. HTTPS and HTTP exhibit identical traffic patterns; the only difference is that HTTPS files are a constant-byte amount larger than HTTP files when transferred. The search for suitable testbeds was complicated by the fact that most Web sites block access by mirroring Web spiders to all or part of their content. We looked for Web sites that include at least several hundred Web pages, are fully accessible to Web spiders, contain a mix of HTML and images, and consist mostly of static content. For the purpose of evaluating our link structure algorithm, sites having a reasonably rich link structure and few if any unresolvable hyperlinks were also preferred. The Unex Web site [7] served as our primary testbed for both simulated and actual studies.

5.1 Numerical Simulations

We first tested our attack prototype on data generated by stochastic model of human browsing behavior. Since browsing patterns vary widely depending on the user and on the site being browsed, the simulations are only intended to provide general insight about traffic analysis methods and defenses. Our model makes the following simplifying assumptions: 1) the user navigates mostly by following hyperlinks but sometimes use other mechanisms such as browser buttons or bookmarks; 2) all hyperlinks on a given page are followed with equal probability; and 3) pages visited via buttons and bookmarks are randomly selected. Since most Web sites are organized hierarchically, the visitation frequency distribution is very uneven; pages close to the root of the hierarchy tend to be visited much more often than pages at the leaves.

The simulation model is implemented using the following algorithm. A page is randomly selected from the database compiled for the target Web site. The next page is then selected by following a randomly chosen hyperlink. If two consecutive pages have been arrived at via hyperlinks, or if the current page contains no outgoing links, the next page is selected at random. If a loop is traversed by following the hyperlinks, a new page is randomly selected with probability 0.8; this is to prevent the simulation from repeatedly alternating between two pages. For each simulation, a sequence of 10,000 pages is generated and the corresponding HTML size and total object size data is used as the input to the attack. Our model assumes that sniffed network traffic can be perfectly parsed to extract the relevant information, which a fully implemented sniffer and analyzer can.

We ran simulations to predict the accuracy of our attack for various Web sites, as well as the effect of randomly padding the sizes of pages. To simulate random padding, the HTML size and total object size (if there are embedded objects) are each padded by a uniformly distributed random number between 0 and some maximum number of bytes. The initial tests were run without using the link structure algorithm and with the browser cache disabled. Table 1 shows accuracy results for the Enough [2], Unex, and Spectator [5] sites, with various amounts of padding. When no padding is used, the HTML size and total object size correctly identifies virtually all pages at all three sites, though accuracy is somewhat lower for the larger sites, presumably because of duplicate page sizes. We can see that padding is quite effective at preventing accurate identification if used in sufficient amounts. As expected, padding is much more effective as a defense for the larger sites; this is particularly evident for the Spectator site, which contains many pages of similar sizes which embed the same images.

Website	**Site size (pages)**	**Accuracy**					
		N=0	**N=100**	**N=300**	**N=1000**	**N=3000**	**N=10000**
Enough	63	100.0	94.9	89.3	78.2	61.1	37.2
Unex	489	96.7	75.1	62.4	45.9	29.7	14.8
Spectator	926	93.4	59.2	44.1	29.0	11.9	2.9

Table 1. Traffic analysis without using the link structure algorithm and with caching disabled. Page sizes are randomly padded by 0 to N bytes.

Max Pad Bytes	**Accuracy**	**Overhead**
0	99.0%	0.0%
100	92.6%	0.4%
300	87.0%	1.1%
1000	75.3%	3.6%
3000	59.2%	11.3%
10000	37.7%	36.1%

Table 2. Traffic analysis for the Unex website with random padding. The link algorithm is used and the browser cache is disabled.

In all of the remaining simulations we used the Unex site and enabled the link algorithm. Table 2 shows the effect of random padding on accuracy and quantifies the additional network traffic generated. The use of link analysis has the effect of making the attack much more resistant to padding, increasing by an order of magnitude the amount of padding needed to provide the same amount of protection. Thus, relying on random padding alone to defend against traffic analysis imposes a heavy burden on the networks.

Packet Size	**Accuracy**	**Overhead**
(Variable)	99.0	0.0%
100	92.7%	0.6%
500	82.0%	2.8%
1460	71.7%	8.9%
4096	54.9%	25.2%

Table 3. Traffic analysis for the Unex website using constant sized packets (cache disabled).

We also simulated the use of constant size packets. This was done by padding the sizes of the HTML file and each individual embedded object to the nearest multiple of the packet size. Table 3 presents accuracy and bandwidth overhead results for several different packet sizes. Compared to random padding, a defense based on constant size packets is slightly less effective for a given level of overhead. As we explained in Section 3.1, this is probably because the amount of data transferred for a file is a deterministic function of its original size.

Finally, we devised a new data generating model that takes into account the use of a local cache. We use the same assumptions about browsing practices as before. The effect of the cache is to alter the frequency distribution of the pages requested, since pages close to the root of the site's hierarchy are often cached and need not be requested frequently. The page sequence is generated according to the following algorithm. We assume that each browsing session visits 50 pages, after which a new session begins with an empty cache. As an approximation, we assume that pages that were not previously visited during the current session are never present in the cache; pages visited once before are cached with probability 0.7; and pages visited at least twice before are cached with probability 0.9. Traffic analysis is performed without using object size information.

Max Pad Bytes	Accuracy
0	98.8%
100	81.6%
300	62.7%
1000	36.1%
3000	19.4%
10000	10.9%*

Table 4. Traffic analysis for the Unex website with random padding and caching enabled. Overhead is similar to that of Table 2. * Used sample size of 1000

The results are shown in Table 4. When no padding is used, almost all the pages are correctly identified. However, random padding is much more effective against traffic analysis when caching is used, due to the loss of object size information. For example, padding HTML size by a no more than 3000 bytes would conceal more than 80% of the requested pages while imposing a modest bandwidth overhead of about 10% (from Table 2; we assume that caching has no appreciable effect on overhead). This does not mean that padding alone suffices as an adequate and bandwidth-efficient defense for all Web sites, or for all user needs. Even with relatively ample padding, some pages are still likely to be identifiable, either because they are unusually large or because of their position within the link structure. It may be more effective to set the padding parameters dynamically so that maximum amount of padding for each file is proportional to its size, though we do not expect this to stop all leakage of identifying information. The fact remains that large files are hard to hide amongst many small files.

5.2 User Testing

We also conducted user studies to test the effectiveness of our attack in a realistic setting. Several subjects browsed the Unex Web site using Netscape Navigator while network traffic at their computer was monitored. Subjects were instructed to browse for 5-10 minutes, pausing for at least two seconds on each page they visit (for prototype implementation simplification only). The URL of each page requested by the client browser was recorded to verify the pages guessed by the traffic analysis system. Occasionally the subjects visited pages that either were not retrieved by the Web spider or consisted of data types which were not parsed; such pages were not represented in our Web site database and thus were excluded from consideration. In a small fraction of cases, the network traffic parser erred in extracting the HTML size and total object size; we made minor modifications to the query algorithm to detect and correct for some of the common mistakes. We ran separate trials with browser caching enabled and disabled; for each trial, we ran traffic analysis both with and without the link structure algorithm. For the trials where caching was enabled, the size of the memory and disk caches ranged from 100-300 KB apiece. No defenses were used. The results, summarized in Table 5 below, show that our traffic analysis attack is very effective under all the conditions we tested. Most importantly, the high accuracy achieved without the use of the link algorithm and with caching enabled confirms that most of the documents visited were identifiable based on their HTML size alone. The results do not disprove the power of the link algorithm, but merely indicate that it was not necessary under these conditions; in such cases, the primary effect of link analysis is to strengthen traffic analysis attacks and increase their resistance to padding defenses.

	Correct	**Total**	**Accuracy**
Cache disabled, with link alg	88	92	96%
Cache disabled, w/o link alg	88	92	96%
Cache enabled, with link alg	67	71	94%
Cache enabled, w/o link alg	67	71	94%

Table 5. Results of user browsing tests on the Unex Web site.

6. Discussion and Conclusions

Traffic analysis constitutes a real threat to the privacy of SSL-encrypted Web browsing. We have implemented and tested a successful traffic analysis attack, and our results demonstrate that Web pages browsed using SSL can be identified with very high accuracy using very simple statistical techniques. While our investigation has clarified some of the fundamental issues relating to this attack, further work is needed to evaluate the threat of traffic analysis for different types of Web content, and in particular for interactive personal and financial transactions where privacy is especially crucial.

We also identified several defenses that provide some degree of protection from traffic analysis. For the most part, these fall into the rough categories of protocol-based and proxy-based approaches. Proxy services for anonymous Web browsing already exist and should provide adequate protection against all but the most powerful eavedroppers. However, since most users of SSL encryption are unaware of the issue of traffic analysis, we believe that protocol-based defenses are still necessary and should be incorporated into commercial Web browsers. Theoretical studies suggest that random padding could provide reasonable privacy protection in many cases if properly used. However, the effectiveness of random padding depends greatly on the quantity of padding as well as the size and structure of the particular Web site being browsed; moreover, the required amount of padding is also highly site-specific. While we agree that SSL should be amended to support random padding for all cipher modes as suggested in [8], we do not believe that random padding alone is sufficient to protect against traffic analysis in the general case. Further study is needed to determine whether it is feasible to devise a more robust protocol-level defense, and if so to provide insight on how such a defense might best be constructed.

References

1. Anonymiser, Inc., http://www.anonymizer.com/
2. Enough is Enough Home Page, http://www.enough.org
3. Ian Goldberg, David Wagner, and Eric Brewer, “Privacy-enhancing technologies for the Internet”, IEEE COMPCON ’97, February 1997.
4. L. Rabiner and B. Juang, “An introduction to hidden Markov models,” IEEE ASSP Magazine, 1986.
5. American Spectator Web Site, http://www.spectator.org/.
6. TranSend, http://transend.cs.berkeley.edu/
7. U.C. Berkeley Extension, http://www.unex.berkeley.edu:4243/
8. David Wagner and Bruce Schneier, “Analysis of the SSL 3.0 protocol”, 2nd USENIX Workshop on Electronic Commerce, November 1996.
9. Webcopy, ftp://ftp.inf.utfsm.cl/pub/utfsm/perl/webcopy.tgz