Towards Quantification of Network-Based Information Leaks via HTTP

Kevin Borders
Web Tap Security, Inc.
White Lake, MI 48383
kborders@webtapsecurity.com

Atul Prakash
University of Michigan
Ann Arbor, MI 48109
aprakash@eecs.umich.edu

Abstract

As the Internet grows and network bandwidth continues to increase, administrators are faced with the task of keeping confidential information from leaving their networks. Today's network traffic is so voluminous that manual inspection would be unreasonably expensive. In response, researchers have created data loss prevention systems that check outgoing traffic for known confidential information. These systems stop naïve adversaries from leaking data, but are fundamentally unable to identify encrypted or obfuscated information leaks. What remains is a wide open pipe for sending encrypted data to the Internet.

We present an approach for quantifying network-based information leaks. Instead of trying to detect the presence of sensitive data—an impossible task in the general case—our goal is to measure and constrain its maximum volume. We take advantage of the insight that most network traffic is repeated or determined by external information, such as protocol specifications or messages sent by a server. By discounting this data, we can isolate and quantify true information leakage. In this paper, we present leak measurement algorithms for the Hypertext Transfer Protocol (HTTP), the main protocol for web browsing. When applied to real web traffic from different scenarios, the algorithms show a reduction of 94–99.7% over a raw measurement and are able to effectively isolate true information flow.

1. Introduction

Network-based information leaks pose a serious threat to confidentiality. They are the primary means by which hackers extract data from compromised computers. The network can also serve as an avenue for insider leaks, which, according to a 2007 CSI/FBI survey, are the most prevalent security threat for organizations [5]. Because the volume of legitimate network traffic is so large, it is easy for attackers to blend in with normal activity, making leak prevention difficult. In one experiment, a single computer browsing a social networking site for 30 minutes generated over 1.3 MB of legitimate request data—the equivalent of about 195,000 credit card numbers. Manually analyzing network traffic for leaks would be unreasonably expensive and error-prone. Limiting network traffic based on the raw byte count would only help detect large information leaks due to the volume of normal traffic.

In response to the threat of network-based information leaks, researchers have developed data-loss prevention (DLP) systems [6, 8]. DLP systems work by searching through outbound network traffic for known sensitive information, such as credit card and social security numbers. Some even catalog sensitive documents and look for excerpts in outbound traffic. Although they are effective at stopping accidental and plaint-text leaks, DLP systems are fundamentally unable to detect encrypted information flows. They leave an open channel for leaking encrypted data to the Internet.

In this paper, we introduce a new approach for precisely quantifying network-based information leaks. Rather than searching for known sensitive data—an impossible task in the general case—we aim to measure and constrain its maximum volume. We exploit the fact that a large portion of network traffic is repeated or constrained by protocol specifications. By ignoring this fixed data, we can isolate true information flows from a client to the Internet, regardless of encryption or obfuscation. The algorithms presented in this paper yield results that are 94–99.7% smaller than raw request sizes for several common browsing scenarios, and 75–97% smaller than a simple calculation from prior research [1]. They also measure information content irrespective of data hiding techniques. The end result is a small, evasion-proof bandwidth measurement that can precisely quantify and isolate network-based information leaks.

The leak measurement techniques in this paper focus on the Hypertext Transfer Protocol (HTTP), the main protocol for web browsing. They take advantage of HTTP and its interaction with Hypertext Markup Language (HTML) documents to identify information originating from the user. The basic idea is to compute the content of expected HTTP requests using only externally available information, including previous network requests, previous server responses, and protocol specifications. Then, the amount of *unconstrained* outbound bandwidth is equal to the edit distance (edit distance is the size of the edit list required to transform one string into another) between actual and expected requests plus timing information. This unconstrained bandwidth measurement represents the maximum amount of information that could have been leaked by the client, while minimizing the influence of repeated or constrained data.

The reasons that we chose to focus on HTTP in this paper are twofold. First, it is the primary protocol for web browsing. Many networks, particularly those in which confidentiality is a high priority, will only allow outbound HTTP traffic through a proxy server and block other protocols. In this scenario, HTTP would be the only option for an attacker to leak data over the network. Second, a large portion of information in HTTP requests is constrained by protocol specifications. This is not the case for e-mail, where most information is in the e-mail body—an unconstrained field. Although the concepts in this paper would not apply well to all network traffic, we believe that they will yield much lower bandwidth measurements for most protocols, including instant messaging and domain name system (DNS) queries. Applying techniques for unconstrained bandwidth measurement to other protocols is future work.

The remainder of this paper discusses related work in Section 2, a formal problem description in Section 3, our measurement techniques in Section 4, the evaluation in Section 5, and concluding remarks in Section 6.

2. Related Work

Prior research on detecting covert web traffic has looked at measuring information flow via the HTTP protocol [1]. Borders et al. introduce a method for computing bandwidth in outbound HTTP traffic that involves discarding expected header fields. However, they use a stateless approach and therefore are unable to discount information that is repeated or constrained from previous HTTP messages. In our evaluation, we compare the leak measurement techniques presented in this paper with the simple methods used by Web Tap [1] and demonstrate a 75–97% measurement reduction for legitimate traffic.

Research on limiting the capacity of channels for information leakage has traditionally been done assuming that systems deploy mandatory access control (MAC) policies [2] to restrict information flow. However, mandatory access control systems are rarely deployed because of their usability and management overhead, yet organizations still have a strong interest in protecting confidential information.

One popular approach for protecting against network-based information leaks is to limit where hosts with can send data using a content filter, such as Websense [9]. Content filters may help in some cases, but they do not prevent all information leaks. A smart attacker can post sensitive information on *any* website that receives input and displays it to other clients, including useful sites such as www.wikipedia.org. We consider content filters to be complimentary to our measurement methods, as they reduce but do not eliminate information leaks.

Though little work has been done on quantifying network-based information leaks, there has been a great deal of research on methods for leaking data. Prior work on convert network channels includes embedding data in IP fields [3], TCP fields [7], and HTTP protocol headers [4]. The methods presented in this paper aim to quantify the maximum amount of information that an HTTP channel could contain, regardless of the particular data hiding scheme employed.

3. Problem Description

In this paper, we address the problem of quantifying network-based information leaks by isolating information from the client in network traffic. We will refer to information originating from the client as *UI-layer* input. From a formal perspective, the problem can be broken down to quantifying the set $\boldsymbol{U}$ of UI-layer input to a network application given the following information:

- $\boldsymbol{I}$ – The set of previous network inputs to an application.
- $\boldsymbol{O}$ – The set of current and previous network outputs from an application.
- $\boldsymbol{A}$ – The application representation, which is a mapping: $\boldsymbol{U} \times \boldsymbol{I} \rightarrow \boldsymbol{O}$ of UI-layer information

combined with network input to yield network output.

By definition, the set ***I*** cannot contain new information from the client because it is generated by the server. In this paper, the application representation ***A*** is based on protocol specifications, but it could also be derived from program analysis. In either case, it does not contain information from the client. Therefore, the information content of set ***O*** can be reduced to the information in the set ***U***. If the application has been tampered with by malicious software yielding a different representation ***A'***, then the information content of tampered output ***O'*** is equal to the information content of the closest expected output ***O*** plus the edit distance between ***O*** and ***O'***. Input supplied to an application from all sources other than the network is considered part of ***U***. This includes file uploads and system information. Timing information is also part of the set ***U***. Though we do not attempt to quantify timing bandwidth in this paper, we plan to extend the methods presented by Cabuk et al. [3] in the future to measure the bandwidth of active HTTP request timing channels.

4. Measurement Techniques

4.1 HTTP Request Overview

There are two main types of HTTP requests used by web browsers, GET and POST. GET is typically used to obtain resources and POST is used to send data to a server. An example of an HTTP POST request can be seen in Figure 1. This request is comprised of three distinct sections: the request line, headers, and the request body. GET requests are very similar except that they do not have a request body. The request line contains the path of the requested file on the server, and it may also have script parameters. The next part of the HTTP request is the header field section, which consists of "<*field*>: <*value*>" pairs separated by line breaks. Header fields relay information such as the browser version, preferred language, and cookies. Finally, the HTTP request body follows the headers and may consist of arbitrary data. In the example message, the body contains an encoded name and e-mail address that was entered into a form.

4.2 HTTP Header Fields

The first type of HTTP header field that we examine is a fixed header field. Fixed headers should be the same for each request in most cases. Examples include the preferred language and the browser version. We only count the size of these headers for the first request from each client, and count the edit distance from previous requests on subsequent changes. Here, we treat all HTTP headers except for Host, Referer, and Cookie as fixed. Some of these header fields, such as Authorization, may actually contain information from the user. When these fields contain new data, we again count the edit distance with respect to the most recent request.

Next, we look at the Host and Referer header fields. The Host field, along with the request path, specifies the request's uniform resource locator (URL). We only count the size of the Host field if the request URL did not come from a link in another page, which we discuss more in the next section. Similarly, we only count the Referer field's size if does not contain the URL of a previous request.

Finally, we examine the Cookie header field to verify its consistency with expected browser behavior. The Cookie field is supposed to contain key-value pairs from previous server responses. Cookies should never contain UI-layer information from the client. If the Cookie differs from its expected value or we do not have a record from a previous response (this could happen if a mobile computer is brought into an enterprise network), then we count the edit distance between the expected and actual cookie values. At least one known tunneling program, Cooking Channel [4], hides information inside of the Cookie header in violation of standard browser behavior. The techniques presented here correctly measure outbound bandwidth for the Cooking Channel program.

4.3 Standard GET Requests

HTTP GET requests are normally used to retrieve resources from a web server. Each GET request identifies a resource by a URL that is comprised of the server host name, stored in the Hostname header field, and the resource path, stored in the request line. Looking at each HTTP request independently, one cannot determine whether the URL contains UI-layer information or is the result of previous network input (i.e., a link from another page). If we consider the entire browsing session, however, then we can discount request URLs that have been seen in previous server responses, thus significantly improving unconstrained bandwidth measurements.

```
1 POST /download HTTP/1.1
2 Host: www.webtapsecurity.com
2 User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;
  en-US; rv:1.8.1.12) Gecko/20080201 Firefox/2.0.0.12
2 Keep-Alive: 300
2 Connection: keep-alive
2 Referer: http://www.webtapsecurity.com/download.html
2 Content-Type: application/x-www-form-urlencoded
2 Content-Length: 73
3 FirstName=John&LastName=Doe&Email=johndoe%40example.
  com&Submit=Download
```

(a)

```
<html>
<body>
 <form action="/download" method="post">
  <input type="text" name="FirstName">
  <input type="text" name="LastName">
  <input type="text" name="Email">
  <input type="submit" value="Download">
 </form>
</body>
</html>
```

(b)

Figure 1. (a) A sample HTTP POST request for submitting contact information to download a file. Line **1** is the HTTP request line. Lines marked **2** are request headers, and line **3** is the request body. Bytes counted by a simple algorithm are highlighted in gray. UI-layer data is highlighted in black with white text.
(b) A sample HTML document at http://www.webtapsecurity.com/download.html that generated request (a).

The first step in accurately measuring UI-layer information in request URLs is enumerating all of the links on each web page. In this paper, we only use simple HTML parsing to extract link URLs. An example of an HTML link is "`<a href="http://www.example.com/page">Click Here!</a>`" where the URL is "http://www.example.com/page" and the link text is "Click Here!". In the future, we plan to handle Javascript constructs such as such as "`onclick = 'this.document. location = <`*`link`*`>'`". We also plan to employ program analysis techniques or run Javascript through a full-fledged interpreter to handle more advanced Javascript code related to links. Although link extraction is undecidable in general, these approaches are likely to be successful in extracting links for most real cases. Currently, we count URLs that we cannot identify with HTML parsing as UI-layer information.

After the set of links has been determined for each page, we can measure the amount of UI-layer information conveyed by GET requests for those URLs. The first step is dividing links up into two categories: *mandatory* and *voluntary*. A mandatory link is one that should always be loaded. Examples include images, scripts, etc. A voluntary link is selected by the user after the page has been loaded. Loading URLs from mandatory links does not directly leak any information. Omission of some mandatory links may directly leak up to one bit of information per link (one bit for each link, not just omitted links). Reordering mandatory links (they have a predetermined order) may leak up to $\log_2(n!)$ bits where n is the number of mandatory links that were loaded. If we encounter missing or reordered mandatory links, we count $\log_2(m!) + n$ bits where n is the original number of links and m is the number of links that were loaded. For voluntary links, we count $\log_2(n)$ for each request where n is the number of voluntary links on the parent page. We use this figure because selecting 1 of n links leaks up to $\log_2(n)$ bits of information. When the user has multiple pages open, the number of available links may be greater than those on one page. In practice, we count $\log_2(n)$ bits for a link only if it came from the last page that was loaded. Otherwise, we count $\log_2(n) + \log_2(p)$ bits where n is the number of links on the parent page and p is the number of pages in the user's browsing history.

4.4 Form Submission Requests

The primary method for transmitting information to a web server is form submission. Form submission requests send information that the user enters into input controls such as text boxes and radio buttons. They may also include information originating from the server in hidden or read-only fields. Form submissions contain a sequence of delimited <*name*, *value*> pairs, which can be seen in the body of the POST request in Figure 1a. The field names, field ordering, and delimiters between fields can be derived from the page containing the form seen in Figure 1b and thus do not convey UI-layer information. Field values may also be taken from the encapsulating page in some circumstances. Check boxes and radio buttons can transmit up to one bit of information each, even though the value representing "on" is often several bytes. Servers can store client-side state by setting data in "hidden" form fields, which are echoed back by the client upon form submission. Visible form fields may also have large default values,

Scenario	Total Req. Count	Raw (bytes)	Simple (bytes/%)	Precise (bytes/%)	Avg. Precise Size
Web Mail	508	619,661	224,259 / 36.2%	37,218 / 6.01%	73.3 bytes
Sports News	911	1,187,579	199,119 / 16.8%	49,785 / 4.19%	54.6 bytes
News	547	502,208	74,497 / 14.8%	16,582 / 3.30%	30.3 bytes
Shopping	1,530	913,226	156,882 / 17.2%	26,390 / 2.89%	17.2 bytes
Social Networking	1,175	1,404,251	91,270 / 6.5%	15,453 / 1.10%	13.2 bytes
Blog	191	108,565	10,996 / 10.1%	351 / 0.32%	1.8 bytes

Table 1. Bandwidth measurement results for six web browsing scenarios using three different measurement techniques, along with the average bytes/request for the precise technique.

as is the case when editing a blog post or a social networking profile. For fields with default values, we measure the edit distance between the default and submitted values. We measure the full size of any unexpected form submissions or form fields, which may result from active Javascript.

4.5 Custom Web Requests and Edit Distance

Custom network applications, Active Javascript, and malicious software may send arbitrary HTTP requests that do not conform to the behavior of a web browser. This leads us to the problem of measuring UI-layer information ***U*** for an unknown application representation ***A'***. In this case, we reduce the effects of repetition by counting the edit distance between each new request and recent requests to the same server from the same client, rather than counting the entire size of each request. In the future, we plan to explore the use of an incremental compression algorithm on custom web requests to further isolate true outbound information flows. Analyzing active Javascript to derive its application representation may also help to generate more accurate measurements for custom web applications.

5. Evaluation

We evaluated our leak quantification techniques on web traffic from several legitimate web browsing scenarios. The scenarios were 30-minute browsing sessions that included web mail (Yahoo), social networking (Facebook), news (New York Times), sports (ESPN), shopping (Amazon), and personal blog websites. The results are shown in Table 1. The precise measurements show a major reduction compared to the raw byte counts, ranging from 0.3–6.0% of the original values. The precise methods also perform much better than simple bandwidth measurements from prior research [1], demonstrating a reduction to 3–25% of the original values. In the first five scenarios, the precise measurements were still significantly larger than the amount of UI-layer information, which we approximated by recording the number of link traversals and size of form submissions. After examining the web pages in those scenarios, we found this overestimate to be a direct result of Javascript code and Flash objects, particularly from advertisements. For the Web Mail scenario, the median request size was 5 bytes. When compared to the average of 73.3, this indicates that there were a few large requests whose URLs did not appear as links in web pages. We believe that more advanced Javascript and Flash processing will allow us to correctly extract many of these links in the future. Our goal is to approach an optimal case where we correctly read all links in each document. The Blog reading scenario is representative of this best case because no links came from Javascript code. We hope to refine the precise analysis techniques so that the average count is only a few bytes across all browsing scenarios. In the future, we also plan on establishing long-term byte count thresholds similar to those in Web Tap [1] for identifying clients that leak suspiciously large amounts of data.

6. Conclusions and Research Challenges

In this paper, we presented methods for precisely quantifying information leaks in outbound web traffic. These methods exploit protocol knowledge to filter repeated and constrained message fields, thus isolating true information flows from the client. The resulting measurements can help identify leaks from spyware and malicious insiders. We evaluated the precise analysis techniques by applying them to web traffic from several browsing scenarios, including web mail, online shopping, and social networking. They produced request size measurements that were 94–99.7% smaller than raw bandwidth values, demonstrating their ability to filter out constrained information.

The main research challenge we encountered was measuring web requests from pages with active Javascript code or Flash objects. Correctly extracting links from Javascript and Flash is undecidable in general, and may require running scripts in a full-fledged interpreter or performing complex static analysis, even in common cases. Both of these approaches would have a significant impact on performance. Optimizing the precise bandwidth measurement techniques to handle large traffic volumes will be another research challenge. Caching parse results and storing hash values instead of full strings will reduce CPU and memory overhead but hurt accuracy for dynamic content and edited responses. In the future, we plan to quantify these performance tradeoffs and introduce more powerful Javascript analysis techniques.

7. References

[1] K. Borders and A. Prakash. Web Tap: Detecting Covert Web Traffic. In *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS)*, 2004.

[2] S. Brand. DoD 5200.28-STD Department of Defense Trusted Computer System Evaluation Criteria (Orange Book). *National Computer Security Center*, 1985.

[3] S. Cabuk, C. Brodley, and C. Shields. IP Covert Timing Channels: Design and Detection. In *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS)*, 2004.

[4] S. Castro. How to Cook a Covert Channel. *hakin9*, http://www.gray-world.net/projects/ cooking_channels/ hakin9_cooking_channels_en.pdf, 2006.

[5] R. Richardson. CSI Computer Crime and Security Survey. http://i.cmpnet.com/v2.gocsi.com/pdf/ CSISurvey2007.pdf, 2007.

[6] RSA Security, Inc. RSA Data Loss Prevention Suite. *RSA Solution Brief*, http://www.rsa.com/ products/EDS/sb/DLPST_SB_1207-lowres.pdf, 2007.

[7] S. Servetto and M. Vetterli. Communication Using Phantoms: Covert Channels in the Internet. In *Proc. of the IEEE International Symposium on Information Theory*, 2001.

[8] VONTU. Data Loss Prevention, Confidential Data Protection – Protect Your Data Anywhere. http://www.vontu.com, 2008.

[9] Websense, Inc. Web Security, Internet Filtering, and Internet Security Software. http://www.websense.com/global/en/, 2008.