Web Tap: Detecting Covert Web Traffic

Kevin Borders, Atul Prakash
University of Michigan
Department of Electrical Engineering and Computer Science
Ann Arbor, MI 48109-2122
{kborders, aprakash}@umich.edu

ABSTRACT

As network security is a growing concern, system administrators lock down their networks by closing inbound ports and only allowing outbound communication over selected protocols such as HTTP. Hackers, in turn, are forced to find ways to communicate with compromised workstations by tunneling through web requests. While several tools attempt to analyze inbound traffic for denial-of-service and other attacks on web servers, Web Tap's focus is on detecting attempts to send significant amounts of information *out* via HTTP tunnels to rogue Web servers from within an otherwise firewalled network. A related goal of Web Tap is to help detect spyware programs, which often send out personal data to servers using HTTP transactions and may open up security holes in the network. Based on the analysis of HTTP traffic over a training period, we designed filters to help detect anomalies in outbound HTTP traffic using metrics such as request regularity, bandwidth usage, inter-request delay time, and transaction size. Subsequently, Web Tap was evaluated on several available HTTP covert tunneling programs as well as a test backdoor program, which creates a remote shell from outside the network to a protected machine using only outbound HTTP transactions. Web Tap's filters detected all the tunneling programs tested after modest use. Web Tap also analyzed the activity of approximately thirty faculty and students who agreed to use it as a proxy server over a 40 day period. It successfully detected a significant number of spyware and aware programs. This paper presents the design of Web Tap, results from its evaluation, as well as potential limits to Web Tap's capabilities.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive Software – *backdoors, spyware*; C.2.3 [**Network Operations**]: Network monitoring

General Terms

Security, Measurement, Design, Algorithms

Keywords

Covert channels, intrusion detection, anomaly detection, HTTP tunnels, spyware detection.

CCS'04, October 25-29, 2004, Washington, DC, USA.

1. INTRODUCTION

Network security has been an increasing concern for network administrators and executives alike. Consequently, Firewalls and proxy servers have become prevalent among high-security networks (and even private homes). Many networks require all traffic to the internet to go through an HTTP proxy server or mail server, allowing no direct access to the internal network. This makes the job of a hacker much more difficult than before, where direct access to network machines was available.

When a hacker attacks a network with no direct access to the internet, the first step is getting a user to access a malicious file or web site. This can be done effectively by e-mailing a Trojan horse program or a link to a page which exploits the browser [7]. Once the machine is compromised, the next step is to establish a path of communication. Traditionally, this would be done by installing a backdoor program such as BackOrifice [6]. The problem with using such programs on firewalled networks is that they listen for an incoming connection on a specific port. All incoming traffic, however, is blocked. This means that the only way to communicate with a compromised machine is to have it make a *callback* (outbound connection). Often, the only two ways out of the network are through a mail server or through a proxy server. Since e-mail is often more closely logged and filtered, the hacker may find outbound HTTP transactions to be the best avenue for communication with a compromised workstation.

Spyware is also a huge problem for both system administrators and users alike [4]. Besides annoying users by popping up advertisements, spyware can leak information about a user's behavior or even send data on the machine to outside servers. Spyware programs can also degrade system performance and take valuable time and effort to remove. In addition to these lesser threats, Security holes have been found in Gator and eZula (two popular spyware programs) that would allow a hacker to execute arbitrary code on a target machine [13, 30].

Web Tap is a network-level anomaly detection system that takes advantage of legitimate web request patterns to detect covert communication, backdoors, and spyware activity that is tunneled through outbound HTTP connections. We note that, unlike the previous work on securing web servers (e.g., [22]), Web Tap's focus is on analyzing *outbound* HTTP traffic from protected network machines to outside web servers, rather than guarding web servers against hostile attacks. The goal is to make it more difficult for hackers or malicious users to run Trojan and HTTP tunnel programs within an organization that leak information to the outside. Web Tap is designed for deployment at an organization's HTTP proxy server (either passively or actively) to help detect anomalies in outbound traffic.

To evaluate Web Tap, we used it to look at web traffic from 30 clients over a 40-day period as well as traffic from known

HTTP tunneling programs. We were successful at detecting different types of spyware and adware as well as many data mining and advertisement servers. During the 40 days of observation, Web Tap generated alerts for adware clients such as Weatherbug, Kazaa, Lycos search bar, Google search bar, and Gator. It also was able to find programs which may be unwanted in a workplace environment such as iTunes, BitTorrent, and AIM Express. In addition to non-browser clients, Web Tap detected data mining and advertisement sites such as coremetrics.com, ru4.com abetterinternet.com, and doubleclick.net. Two of the three known HTTP tunneling programs tested, Wsh [12] and Firepass [11], immediately caused Web Tap to raise an alert. The third HTTP tunnel, Hopster [18], was detected an hour and twenty minutes after it began running. A custom HTTP tunnel that we designed, which does a better job of mimicking legitimate browser requests, was detected within seven hours. When the backdoor was actively used to transfer files, it was detected almost immediately.

The rest of the paper is presented as follows. Section 2 discusses related work. Section 3 gives a threat model. Section 4 presents the design of filtering methods, based on measurements during the one week training phase. Section 5 provides an evaluation of Web Tap for an extended period after the evaluation phase. Section 6 talks about vulnerabilities in Web Tap filters. Section 7 outlines future work and Section 8 concludes.

2. RELATED WORK

Signature analysis is a commonly used technique to look for Trojan programs and to do intrusion detection. For example, Snort [31] is configured with over 2500 signature rules to detect scans and attacks. Several commercial programs detect and remove spyware from computers by using the same principle and looking for spyware program signatures [1, 32, 33]. One limitation of signature analysis techniques is that new attacks are developed frequently that existing signatures may fail to detect. For that reason, signature analysis techniques should be complemented with anomaly detection techniques. Web Tap is able to detect new spyware and HTTP tunneling programs because it relies on anomaly detection, rather than signature analysis. Additionally, Web Tap runs at the network level, not the host level, and thus is more easily deployed for an organization that uses an HTTP proxy server for all its outbound web traffic.

Tracking sequences of events using Markov chains or other models has been used for host and network intrusion detection [9, 15, 16, 23, 36]. This approach is very effective for many situations such as analysis of system call traces [15] to detect tampering of applications on a system after. Anomaly detection has also been used to detect network attacks [32, 36] and attacks on web servers [22].

In [22], the focus is on detecting malicious incoming traffic to a server by building a probabilistic profile of web application parameters exported by the web server. The methods employed in Web Tap differ from previous work by targeting outbound rather than inbound communications, with the primary focus being on detection of HTTP tunnels. The fundamental barrier in analyzing outbound traffic is that, because of multitude of web services that can exist outside the system, it is difficult to collect enough data to do probabilistic analysis for every web application to build reasonable profiles; users go to new web sites very often when browsing the web. For inbound HTTP requests to a web server, on the other hand, a profile can be built of appropriate requests, sizes of parameters, and typical sequences of web pages accessed over time, since all incoming requests are for the same web server [22].

Zhang and Paxson describe a method for detecting backdoors [37]. They look at the timing of packet arrivals and packet sizes in order to characterize an interactive shell. For delay times, they exploited the observation that keystroke inter-arrival periods follow a Pareto distribution with a high variance [27]. For packet sizes, they excluded connections that did not contain a large enough percentage of small requests. The Pareto model is difficult to extend to Web Tap. The interactive shell component of a backdoor program controlled by a remote hacker will not send requests when the hacker types them; the backdoor server has to wait for a callback from the client before sending any data. Instead of following a Pareto distribution, the delay times will follow a distribution according to whatever algorithm the backdoor client uses to schedule callback times. Packet size filtering does not apply as well to web tap either because commands can easily be hidden in larger HTML pages.

Significant research exists on human browsing patterns for proxy cache and web server performance optimizations [3, 10, 20]. Web Tap measures some of the same browsing patterns such as inter-request delay time, request size, and bandwidth usage. Web Tap, however, uses this information to determine if the traffic is coming from a legitimate user, while previous research only looked at human browsing patterns for performance reasons.

A substantial body of work exists on covert channel analysis, including detection of covert channels between processes or users on the same machine [24, 26]. A report by McHugh [24] defines a covert channel as "A mechanism that can be used to transfer information from one user of a system to another using means not intended for this purpose by the system developers." Examples include manipulating CPU usage or disk utilization to communicate information. In contrast to previous work, Web Tap does not deal with covert channels *per se*. The communications that Web Tap detects may be covert, but the channel is not. There is nothing inherently secret about HTTP transactions; they are designed to allow the exchange of information. Backdoors, however, hide data within the noise of legitimate web traffic in order to talk to their owners. These lines of communication are covert even though the channel is not. For this reason the data paths used by backdoors to secretly send information in legitimate web traffic will be referred to as *tunnels*. Furthermore, Web Tap does not claim to entirely eliminate tunnels. The main emphasis is on detection of covert tunnels, with a secondary objective of slowing them down, all without disrupting normal web browsing activity.

It is also possible to prevent some HTTP tunnel activity by deploying a content-filter at the proxy server [25, 35]. Such a filter can be used to prevent people from accessing any website not on an approved list. Besides being a very restrictive policy for many organizations, this will not stop the operation of all backdoors. A well-designed tunnel could still take advantage of web e-mail via an approved site to communicate to its host. A hacker could also compromise a web server on the list of approved sites and use it for communication. If the hacker is able to place a CGI script on one of these servers, the tunnel can communicate with the script to leak information. We do not consider content filters further in this paper, though they can nicely complement Web Tap's capabilities.

3. THREAT MODEL

3.1 HTTP Tunnels

In general, if a protocol is available for communication, people have found ways to tunnel other protocols through it, bypassing any firewall restrictions based on protocols or communication ports. HTTP is no exception. Several programs provide HTTP tunneling to allow users within an organization to access non-HTTP services via HTTP proxies. One such program, Wsh [12], communicates over HTTP and provides file transfer capability as well as a remote shell from machines inside a protected network to remote servers. The program can also encrypt data if desired. Another one, Firepass [11], creates a tunnel between a client process and a remote service running on an arbitrary machine and port.

3.2 Backdoor Programs

While HTTP tunnel programs can be convenient at times for allowing legitimate users to bypass firewalls and get access to remote services, they can also present a serious security threat. The scenario presented in this section is a modest extension of such a program that would allow a remote user to acquire a shell on a machine behind the firewall. For the purposes of this paper, we assume that use of such programs is considered to be a security risk and their detection is a legitimate goal of Web Tap.

To get a better of idea how a backdoor could work, here is a model of an intrusion using such a program:

1) The hacker sends a Trojan horse program to the user, or the user to views a malicious site which exploits the browser [7]. (Much like how spyware programs can be installed.)
2) The payload of the hacker's program contains a backdoor that executes on the remote machine.

Once the backdoor program is running on the remote machine, the hacker needs some way of communicating with it. In this model, the network either has firewall rules in place to block all incoming traffic, or uses a proxy server. If the network uses a firewall, then it also blocks all outgoing packets except HTTP (TCP port 80) and DNS (UDP port 53). In our implementation, Web Tap addresses the HTTP proxy server scenario.

After the backdoor has been installed, it calls back to a web server controlled by the hacker (or a server hosting a script written by the hacker) using HTTP requests. Callbacks can be scheduled according to a fixed-wait timer, a random-wait timer, or times of actual browsing activity. Due to the nature of HTTP protocol, all transactions must be initiated by the client computer.

The threat model assumes that the hacker may make an effort to disguise messages as legitimate HTTP transactions. The communication protocol for the backdoor can hide outbound information in any header field (including the URL), or in data trailing the header for a POST request. The backdoor then receives commands from the hacker hidden within what appears to be a normal web page. Web Tap does not attempt to filter out covert transactions based on the content of web pages returned by the server. There are many clever ways of hiding data [29], and it would be fruitless to try to detect them all.

3.3 Spyware

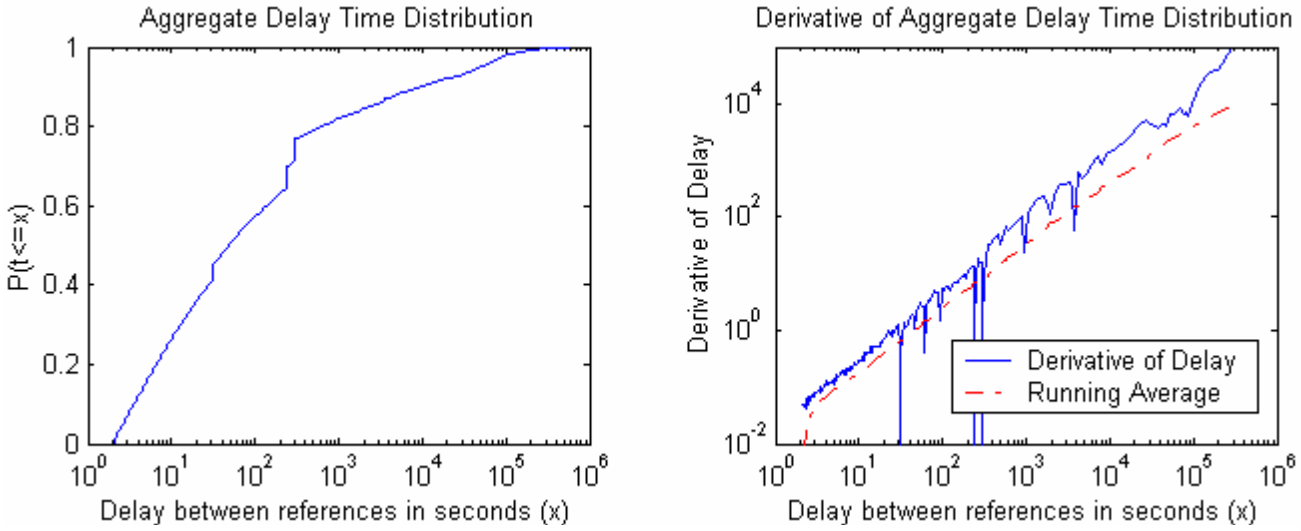
For spyware, the threat model is exactly the same except the initial mode of compromise is different. Spyware programs often install themselves by piggybacking on legitimate software, exploiting browser vulnerabilities, or tricking a user into downloading them voluntarily [4, 30]. Once they are installed, they can use the same method of communication as the backdoor program described above.

4. WEB TAP FILTER DESIGN

The first phase of Web Tap's design was to monitor and record a number of web browsing statistics. Web Tap was set up as an extended proxy server written in Python. Python was chosen because it is easy to code, type-safe, and platform-independent. The original proxy code was taken from a post to a python discussion group [17]. We extended the proxy server to pass requests to a Web Tap measurement module before sending them to their final destination. The module did not block or modify any of the requests. The proxy server was run on a department computer within the University network. Web Tap can also be set up so that the statistics module is inactive and all the requests are logged. For each request, the complete request line and text of the headers with their values need to be logged. Also, the source IP address and the timestamp of the request need to be stored along with the request. If it is a POST request, then the proxy must log the number of bytes sent following the headers to make sure it matches the content-length header value. Once the logging is complete, Web Tap can later run offline and perform analysis on the log files.

Users consisted of students, faculty, friends, and family members who voluntarily configured the browsers on their computers to use the Web Tap proxy server. Thirty users participated in the study for a period of 40 days. The first week of data collection was designated as a training period and used to help design the filter thresholds. During that time, no filters were active. Web Tap passively monitored a number of statistics to detect trends in human web surfing behavior. The following measurement results and filter thresholds are based on the trends observed during the one-week training period. We will examine Web Tap's performance on traffic recorded after the training period later on in section five.

Note that bandwidth and request size measurements do not reflect the actual number of bytes sent from the workstation to the web server. Only bytes that can be modified without setting off a header alert are counted. (A header alert is raised when an HTTP request is sent that is formatted differently from a browser request. An example would be adding a field at the end of an HTTP request "webtap-data: a7Hc..." The field would contain additional data, but would be detected first by the header filter.) This includes bytes in the URL following the hostname, any information after the headers in a post request, one bit for the presence or absence of each optional header field, and variable header values. An additional two bytes are added to help mitigate the effects timing and other covert channels. The two bytes help by increasing the significance of very small requests, which makes it harder for a hacker to send many one or two-byte messages and leak out more than one or two bytes of information through auxiliary channels. Some header fields, such as cookie, can only have a constrained set of values. A backdoor, however, could forge cookies and include information not sent by the server. Right now, Web Tap calculates the number of bytes in the request assuming that all the headers have legitimate values. In the future, we plan to keep additional state in Web Tap to verify header integrity.



(a) **(b)**

Figure 1. (a) Aggregate delay time cumulative distribution with jumps at t = 30 seconds, 4 minutes, and 5 minutes. (b) Derivative of cumulative distribution and running average used to detect anomalies.

Web Tap is only configured to measure a limited set of browsing statistics. Other possible measurements not performed by web tap include: request type (Image, HTML, CGI, etc.), request content, inbound bandwidth, and inbound content. Request type frequencies varied greatly from host to host and thus were too not useful for inferring anomalies. Some sites only served CGI, while others only served images or HTML documents. Web Tap did not attempt to perform content analysis on the transactions because there are too many data hiding techniques such as steganography [29] which are very difficult to detect. Web Tap also does not attempt to monitor inbound bandwidth usage. Generally speaking, web server replies are much larger than client requests. Even the simplest web page, www.google.com, contains approximately 3000 bytes. The aggregate inbound bandwidth usage would be so large that it would be hard to detect any additional hidden data in replies. Instead, Web Tap focuses on outbound requests because they tend to contain far less data and are more useful for detecting covert traffic.

4.1 Header Formatting

Every internet browser has a unique header signature and utilizes a certain set of header fields. Web Tap parses each header and generates an alert when it sees a header that is indicative of a non-browser request. Web Tap also monitors the version of browser and operating system from which requests are being made. If a backdoor sends out Internet Explorer with Windows XP headers when all the computers are running Windows 98, it can be easily detected by the header format filter. Header formatting detection proved to be a useful tool for detecting non-browser web requests. In addition to the standard browsers, (IE and Netscape/Mozilla) it was able to recognize requests from seven different clients in only 24 hours: Windows Update, iTunes, Gator, AIM Express, Windows Media Player, McAfee Web Update, and BitTorrent.

The header format filter does a good job of detecting unwanted clients, as well as many HTTP covert tunnels in their default settings. It can also be used to enforce policies that prevent employees from using clients such as iTunes and AIM Express, and detect some adware programs like Gator. Incidentally, one of the clients detected during the training period, Unicast (unicast.com), set off the header format alarm because it spelled the word "referrer" correctly in the HTTP request. According to the HTTP specification [14] it should be spelled incorrectly as "referer." This is good example of how easy it can be for a hacker to make a mistake when designing a tunneling program.

$$\text{Derivative of Delay} = \begin{cases} 0 & t < 2 \\ V(t) - V(t-1) & 2 \le t \le Size(V) \end{cases}$$

$$\text{Running Average} = \frac{.8 * \sum_{i=1}^{a} Derivative(t-i)}{a} \qquad 1 \le t \le Size(V)$$

Figure 2. Equations for the derivative and average of the delay times seen in Figure 1.

4.2 Delay Times

Web Tap measures the inter-request arrival time for a specific web server from a specific client so programs that make periodic outbound requests, operating on timers, can be detected. Delay times were measured on a per-site basis for each user and stored both in individual vectors and in an aggregate format for all users. The aggregate vector was used to observe the general distribution of inter-request delays. Figure 1a shows the probabilistic distribution of all delay times between site accesses. You can notice jumps in the cumulative distribution curve at 30 seconds, 4 minutes, and 5 minutes. There are also less-pronounced jumps at 15 minutes, 30 minutes, and one hour. These jumps indicate the presence of sites that refresh using a timer.

The jumps can be observed more clearly if we take the derivative with respect to the y-axis of the distribution. This can be seen in Figure 1b. The derivative is plotted along with its running average multiplied by 0.8. The average helps to illustrate

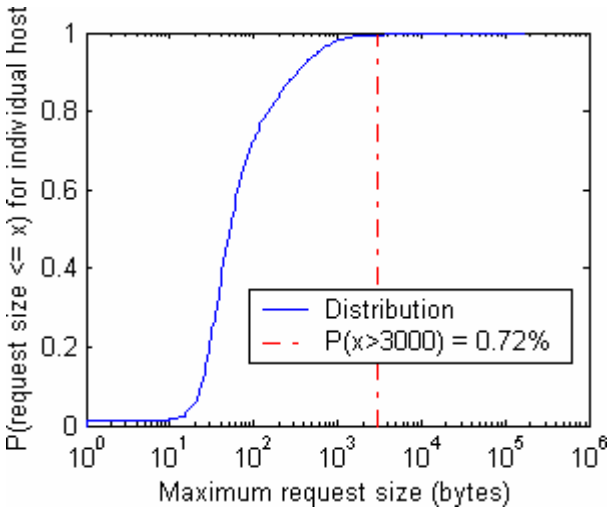


Figure 3. Cumulative distribution of maximum request sizes for ~1600 different sites

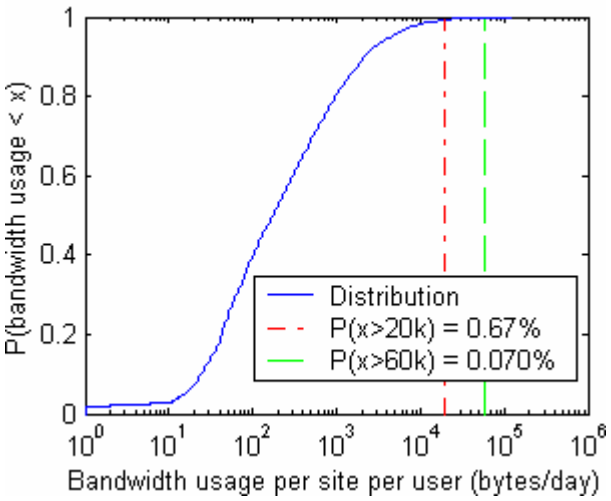


Figure 4. Cumulative distribution of bandwidth usage per site per day for 30 users over a one week period.

places where the derivative drops below the amount of a normal fluctuation. The dips in the derivative that drop below the dotted line correspond to jumps in the distribution at times 30 seconds, 60 seconds, 90 seconds, 4 minutes, 5 minutes, 15 minutes, 30 minutes, 60 minutes respectively. Equations for the derivative and the average can be found in Figure 2. *V* is a vector of delay times taken from every *nth* element in the full delay vector for a site. We chose the maximum of the square root of the full vector size or five for *n*. The value *a* represents the number of values used in the running average. We picked the maximum of the square root of the size of *V* or 3 for *a*.

4.3 Individual Request Size

We found that requests to most sites contain very little information. A hacker would need to send out large amounts of data in order to transfer files and view big directory listings on the remote host. Web Tap monitors request sizes to help detect use of such techniques. Figure 3 illustrates the distribution of maximum request sizes for all users and sites. Out of approximately 1600 unique web servers which received requests, only eleven saw requests of over 3 KB, and only four had requests in excess of 10 KB. Most of these large requests were indicative of file uploads. A good number of them, however, were attributed to ASP scripts with large forms. Some ASP POST requests got as large as 6 KB.

After examining the maximum request size distribution, the most effective filter threshold for single requests appears to be about 3 KB. This setting allows Web Tap to detect almost any file upload using HTTP POST requests. It is also high enough to keep the false alarm rate low. Preventing large post requests will force a hacker to break up transactions and increase the chance of getting caught by one of the other filters.

4.4 Outbound Bandwidth Usage

Since most HTTP requests are small, normal web browsing activity rarely utilizes much *outbound* bandwidth. When a hacker is using HTTP requests for covert communication, outbound bandwidth usage is expected to be higher than the norm. The reason for this is that the hacker usually only sends short requests and small tools (executables) inbound to the computer. Outbound bandwidth, however, is needed to download sensitive documents and directory listings. From a secrecy point of view, a system administrator should be more worried about outbound than inbound traffic.

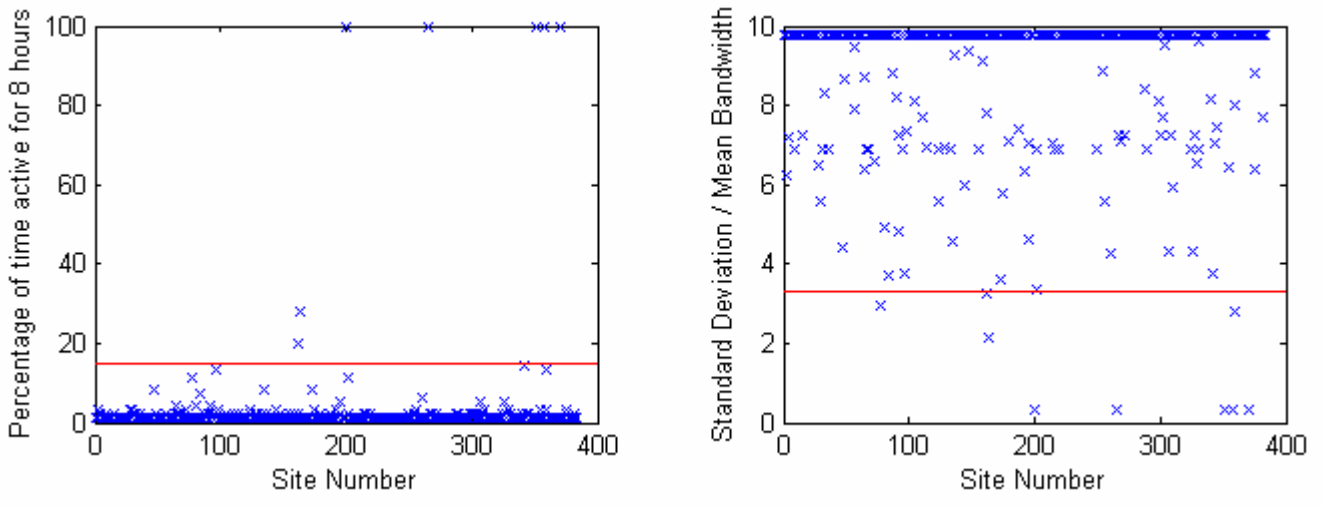
During the training period, Web Tap measured outbound bandwidth on both an aggregate and per-site basis for each user. Total bandwidth consumption was found to contain no useful information that could not be found in the per site measurements, and thus is omitted here. Per user bandwidth was measured mainly to determine what times of day the user is active, not to filter out large bandwidth consumers.

Much like single request sizes, per site bandwidth numbers are pretty low. Figure 4 shows the cumulative distribution of total daily bandwidth usage for thirty users on a per destination-site basis during the training period. For a majority of the sites (>99%), users did not consume more than 20 KB of request bandwidth in a single day. Based on the data, 20 KB appears to be a good lower bound for a daily bandwidth filter threshold. During the one-week training period, 48 of the daily byte-counts exceeded 20 KB. This could be a large amount for a system administrator to sift through, especially considering that these would also contain a significant number of false positives. It may be a good threshold, however, for a high-security network. If the filter was set any lower than 20 KB, then there would be too many false positives.

Of the sites measured, less than 0.1% used over 60 KB of request bandwidth in a single day. All of these were true positives generated by non-browser clients and data mining advertisements. Thus, based on the data, we concluded that the daily bandwidth threshold should be set no higher than 60 KB.

4.5 Request Regularity

Since most hackers need a lot of outbound bandwidth, but have little available to them (without being detected), they are made to spread their requests out over a long period of time. Legitimate web traffic, on the other hand, typically occurs in short bursts. These opposing traffic patterns can be measured by request regularity. The resulting filter is able to detect a backdoor that makes frequent callbacks, even if the requests are interlaced with legitimate traffic to avoid delay time detection.



(a) **(b)**

Figure 5. (a) Seven sites were detected by usage counts for ~400 sites over 8 hours with a detection threshold of 16% (b) Nine Sites were detected using the deviation over mean during an 8 hour period with a detection threshold of 3.3

We used two different methods to measure request regularity. The first is to count the number of time periods where a particular site is accessed by the user being observed. Web Tap counts the number of 5-minute intervals that have non-zero bandwidth usage over 8-hour and 48-hour time periods. If requests appear too often, then the site is classified as being accessed by an automated process. Figure 5a shows a plot of bandwidth counts over an 8-hour time period for approximately 400 sites accessed by a single user. Using a threshold of 16% activity, seven sites were filtered out with no false positives. Five of these were in blatant violation of the threshold; they were active consistently throughout the whole eight hours. The two other sites detected served advertisements that refresh frequently and are persistent throughout browsing sessions. There were also five sites close to the detection threshold (between 10% and 16%), only two of which were false positives, both from livejournal.com. The 16% threshold was chosen conservatively to avoid false alarms and could be lowered even further for a low-traffic network.

The second method used is the coefficient of variation technique. To determine the regularity, the standard deviation of bandwidth usage is calculated and divided by the mean bandwidth usage. Conceptually, this number represents a normalized deviation of bandwidth usage. If a site is accessed in short bursts, which is characteristic of normal human activity, then the coefficient of variation will be high. Low variation in bandwidth usage is indicative of abnormal or non-human activity. The plot of the coefficient of variation measurements for an 8-hour time period can been seen in Figure 5b. We found thresholds of 3.3 for 8 hours and 4.5 for 48 hours to be effective. At those settings, the coefficient of variation method detected nine sites in violation of the threshold both over an 8-hour and a 48-hour period, none of which were false positives. Much like the sites close to the threshold for the counting method, three of the five sites just above 3.3 were false positives, all associated with livejournal.com. For a smaller network, the threshold for this filter could be effectively raised to around 4.0 for an 8-hour period without producing many false positives. All of the seven sites filtered by the counting method were also filtered by the coefficient of variation method.

Web Tap uses both of these measurements, even though they would have resulted in identical filtering during the training period. The primary reason is our assumption that more filters will not hurt, unless they are generating additional false alarms. For example, if an adversary had to only evade the coefficient of variation filter and wanted to make callbacks every 10 minutes, then he or she would just have to make a single large request (~2500 bytes) every 48 hours, and a smaller big request (~200 bytes) every 8 hours. The hacker's solution would, however, fail if both filters are deployed.

```
Hour:        | 1                        24
Mar 26(Fri) | ---------------XXXXXXXXX
Mar 27(Sat) | --X--------------------
Mar 28(Sun) | ------------------------
Mar 29(Mon) | ---------------XX--XXX-X
Mar 30(Tue) | X--XX----------XXXXXXXXX
Mar 31(Wed) | ---------------XXXXXXX-
```

Figure 6. Activity by time of day for one randomly chosen user. 12 AM to 1 AM is on the left, and 11 PM to 12 AM is on the right.

4.6 Request Time of Day

Web Tap recorded the time of day when users typically browse the web on a per-user basis. People tend to follow a schedule and do their browsing at set times. This is illustrated in Figure 6 by the browsing activity seen for a randomly selected user during the first six days of observation. The activity times stay fairly consistent from day to day. If requests are seen during a time when the user is usually inactive, then an intrusion alert can be raised. This approach would be even more effective when applied in a workplace instead of a home environment, where users tend to have more rigid schedules. As an extension of this, day of week and holidays could be monitored in a work environment where schedules are predetermined.

Table 1. Number of alerts and the false alarm rate for each filter. The aggregate row shows results from running all the filters in parallel.

Filter Name		Number of Alerts Over 40 Days	Average Alerts Per Day	False Alarm Rate (Approximate)	False Alarm Requests
Header Format		240	6.00	0% (Exact)	Legitimate browser
Delay Time		118	2.95	5%	Not running on a timer
Individual Request Size		38	0.95	34%	Not upload or spyware
Request Regularity	8-Hour	132	3.30	11%	Not spyware/adware, ad server, or a timer.
	48-hour	65	1.63	7%	
Daily Bandwidth	20 KB	106	2.65	32%	Not spyware/adware, not an ad server, not a file upload, not running on a timer, and not a non-browser client
	30 KB	59	1.48	17%	
	40 KB	39	0.98	15%	
	50 KB	26	0.65	12%	
	60 KB	18	0.45	0%	
	70 KB	14	0.35	0%	
	80 KB	12	0.30	0%	
Time of Day		68	2.62 (Not Including	28%	Anything when the user is present and active
Aggregate		767	19.2	12% (2.3 per day)	

In a home environment, unlike a place of work, users' schedules are more subject to change. This is especially true for college students who have part time jobs and subsequently have schedules that vary from day to day. Even though most of the test subjects were college students, they still showed striking patterns in usage times.

5. EVALUATION

After the one-week learning period, the filters were put to the test against several HTTP tunneling programs, as well as 40 days of web traffic from 30 users. During the evaluation, all the filters were active for every site and user. The purpose was to determine how difficult it would be for a hacker to avoid detection by Web Tap, how much bandwidth would be available to the hacker, and the false alarm rate for the filters and thresholds used by Web Tap. No special filter rules or settings were used to reduce the number of false positives. We plan to add support for customization of Web Tap filters in the future.

5.1 Filter Performance over Full Observation Period

After the first week of measurements designated as the training period, Web Tap was run with its new filters on traffic from the same clients over 40 days. The following section describes the results from the alert logs over the full observation period for approximately 30 clients. From the 40 days worth of data collected, 428,608 requests were made to 6441 different websites, and the proxy log file was 300 Megabytes in size.

Following collection, the information was analyzed by web Tap in offline mode. An overview of the analysis results from the 40-day period can be found in Table 1. The set of requests that were considered false positives varied depending on the filter under consideration. For this experiment, we evaluated false positives by hand. The type of requests that are false positives can be seen in Table 1 in the right-hand column. (An ad server is a web server that hosts refreshing advertisements, such as doubleclick.net.) In addition to the short description given for false positives, any alert generated by a Trojan or HTTP tunnel was considered a true positive, though none were observed during the evaluation period.

Including those mentioned in section 4.1, Web tap detected 17 different non-browser clients and one non-standard browser using the header format filter. Six of the clients detected were unwanted spyware programs. The other eleven clients included those mentioned earlier such as Windows Update, McAfee Web Update, and iTunes.

We also found that 5 out of the 30 observed unique clients had some form of adware on their computer just based on the header filter results. In addition to the spyware clients, others were detected that may not be desirable in a work environment. These included Kazaa, iTunes, AIM Express, and BitTorrent. Once detected, Web Tap (or the proxy server) could be set to block, allow but log, or completely allow certain clients according to network policy.

The large number of header alerts can be attributed to the fact that Web Tap raises an alarm when it sees a bad header once for each web server per user. This means that if iTunes were to access 10 different sites, each would generate an alarm. We plan to reduce the repetition of positives in the future by only raising an alarm once per client type per user and also filtering out alerts from allowed clients.

For the delay time measurements, we logged website access times using one-second granularity. The reason we did not use more precision is that none of the timers observed had periods of less than 30 seconds. In order to detect shorter-period timers, additional precision would be required to differentiate a timer from repeated short delay times.

Although the false positive rate for the delay time filter was low (average of one false alarm every 6 days for our test group), several legitimate websites that refresh using a timer set off alarms. News and sports sites, such as espn.com and nytimes.com, tended to be the primary culprits. One way of filtering out

legitimate sites with timers, which we plan to explore, is creating a list of trusted sites that are allowed to have fixed-interval callbacks. One must be careful, however, since callbacks to trusted sites can be used to leak information.

For the individual request size filter, approximately 35% of the alarms were associated with file uploads. Almost all the other true positives came from data-mining spyware programs. The false positives observed were largely ASP and shopping cart scripts from sites such as www.americanexpress.com and www.sprintpcs.com. Some websites contained forms with very large amounts of data that would be sent in single post requests. A possible method for dealing with this problem, similar to the solution proposed for delay time false positives, and with similar risks, would be to create a database of trusted servers. The database could include popular websites that have very large forms, but do not allow a user to leak data outside of the network through file uploads, public message posting, or other means.

As seen in Table 1, the daily bandwidth filter generates more alerts as well as more false positives as its threshold decreases. The reason that we took multiple measurements is so that a security administrator can decide between increased security and false alarms. For the lowest number of false positives, a threshold of 60 KB appears to be reasonable for small group sizes. The threshold can, however, be set lower. For the 20 KB limit, the false alarm rate is just under one per day. Depending on the level of security desired, a moderate threshold between 30 and 50 KB will keep the false positives at a manageable level, and make sure that most of the real positives are caught. The false positives seen for the daily bandwidth filter, much like the request size filter, consisted mainly of sites with large ASP or shopping cart scripts such as www.tvguide.com and www.sephora.com. The performance of the daily bandwidth filter could be enhanced by giving a higher threshold to popular sites from a list that tend to generate false positives. If the number of false positives is reduced, then a system administrator can lower the detection threshold and find more malicious traffic. Also, true positives could be consolidated by communication between the different filters. Many of the sites detected by the bandwidth filter, such as Gator and doubleclick.net, were found by other filters as well.

The regularity filter results seen in Table 1 consisted of both count and coefficient of variation measurements. We considered the number of false positives generated by this filter to be acceptable (approximately one false alarm every three days). The sites that caused false alarms were only the very popular ones such as ebay.com and livejournal.com. Many of the sites flagged by the regularity filter were found by the delay time filter as well. The regularity filter did, however, find an additional type of adware that the delay filter was unable to detect: browser search bars. This particular breed of advertising program imbeds itself into the person's browser and calls back to its host every time the browser opens up as well as throughout browsing sessions. These are different from other adware programs because their callbacks are triggered by human activity and thus cannot easily be differentiated from a person based on inter-request delay times. Web Tap successfully detected sites that used frequent requests with this filter, even if they coincided with human usage.

The time of day filter was initially configured so that the first week of the 40-day period was used for training. After seeing preliminary results, we lengthened the training time to the first two weeks of the 40-day period in order to increase the effectiveness of the filter. During the first two weeks, no alerts were generated because all requests were representative of normal activity. It is important to note, however, that for the training period, spyware and adware programs were active. We did not attempt to remove fixed-length timer or other non-human activity from the training data. The effectiveness of training could be improved to generate more true positives by removing traffic for sites which set off the delay time or regularity alarms. Nevertheless, we were still able to detect programs such as Gator and Wildtangent even though they had been active during the training period.

5.2 Filter Performance for HTTP Tunnels and a Web Backdoor Program

For the purposes of this experiment, we used several available programs that help tunnel TCP traffic over HTTP. These included Wsh [12], Hopster[18] , and Firepass [11]. The programs we tested are primarily designed to help people inside a network bypass firewall restrictions, for example, to get a shell from a firewalled machine to a remote machine outside the network. We also included a backdoor program that we designed, *Tunl* which allows a user (hacker) outside the network to get a shell on a machine inside the firewalled network. We present the results from running these four programs in the following sections.

5.2.1 Third Party HTTP Tunnels

We installed the three tunneling programs on a workstation using the Web Tap proxy and attempted to send out information using each. Web Tap was immediately able to detect both Wsh and Firepass since they used custom header fields in their requests. After the initial connection, we were unable to successfully transfer any data using Firepass. Wsh did work properly, and it was detected by Web Tap's single request size filter upon transferring a small file (approximately 5 KB). Wsh, however, had to be initiated by the client and so did not call back or set off any other alarms.

We used Hopster to tunnel traffic from AOL Instant Messenger in our experiments. It began running at 10:30 PM and no messages were sent during the night. The next day, 10 KB of data was sent out around Noon. Hopster was not detected immediately like Firepass and Wsh because it used browser-like request headers. Unlike the other two programs, Hopster did make frequent callbacks to the server that were detected by Web Tap's regularity filter after 80 minutes, delay time filter after two hours, and 20 KB daily bandwidth filter after three hours.

5.2.2 Tunl Design

To further evaluate our system, we also designed a prototype remote shell backdoor called *Tunl*. It is made to simulate the situation described in Section 3.2, which allows a user (hacker) outside the network to get a shell on a machine inside the firewalled network. Tunl is written for Windows, a popular target for hackers. It consists of two executables, TunlCli.exe (to run on the compromised host) and TunlServ.exe (to run on hacker's machine), which together provide a remote command shell on the compromised Windows machine. It is designed to tunnel through an HTTP proxy server, but can be set to go through a non-proxy web connection as well.

The first thing TunlCli does when it starts up is launch a hidden command shell with redefined standard input, output, and error handles. It then listens indefinitely to the standard output from the shell, and sends any information back to TunlServ. In addition to sending data from the standard output, it makes

periodic callbacks to check for server commands. Any data received from the server is then piped to the standard input of the command shell for processing. Custom get and put commands, which are not piped to the shell, are included in Tunl for easy file transfers. To avoid sending too many small requests, data is buffered and sent out every 100 milliseconds. Because only outbound requests from the compromised machine via the proxy are allowed, when the hacker enters a command on his computer, the shell has to wait for the client to make an HTTP request before sending the command in the body of an HTTP reply. The response to the server's command is sent back in a subsequent HTTP POST request. All messaged exchanged are designed to emulate Internet Explorer on Windows XP accessing an Apache web server.

Though Tunl did not provide compression or encryption, we tested it with transferring compressed content. Though some tunneling programs support encryption (e.g., Wsh), encryption was largely immaterial to Web Tap since it does not examine data content. Tunl uses a one hour timer to schedule its callbacks and retries twice at 30 second intervals in the case of a failed connection. Note that since Tunl's communication protocol was modeled after Internet Explorer requests and Apache replies, it did not set off any of the header alerts in any test cases.

5.2.3 Tunl with Callback Only Workload

To evaluate the performance of Web Tap, we installed the Tunl program on one of the machines. The first workload we tested only contained callbacks to the Tunl server. This represents the time when a machine is first compromised and the backdoor begins making callbacks, but the hacker has not yet started using the shell to execute commands on the compromised machine.

The results for the Tunl client only making callbacks using a timer were very promising. Even though no data was being transferred, the traffic from this trace was caught by the request regularity, the delay time, and the time of day filters. The 8-hour coefficient of variation bandwidth filter detected the web tunnel 6 hours and 40 minutes after the first callback. The 8-hour activity count filter was unable to detect the backdoor. The program did, however, break the threshold for the 48-hour count filter after about 26 hours. Since the backdoor was running on a timer, the delay time filter was able to detect it in 2 hours and 10 minutes. As far as the time of day filter, the delay until detection varies depending on the individual user's habits as well as the time of initial callback. The time of day filter was triggered by the backdoor very shortly after a time of usual inactivity began.

5.2.4 Minimal Workload

The second test case consists of a hacker using the Tunl shell to go to the local documents directory (containing approximately 180 documents), listing all the files, and downloading a single 500-word uncompressed document with minimal formatting (approximately 25 KB). This is a minimal activity scenario where the hacker only lists one directory and downloads a single small file. We assumed that any hacker who compromises a machine is almost sure to do at least this much.

The minimal workload immediately violated the threshold for the maximum request size filter. It also exceeded the total daily bandwidth threshold of 40 KB. Even though the file was only 25 KB, the uncompressed directory listing for around 180 different files was larger than 15 KB. It also went on to trip the delay time and request regularity filters. The presence of more concentrated activity, however, made the backdoor harder to detect using the coefficient of variation regularity measurement. Instead of detecting Tunl in around 7 hours, the coefficient of variation measurement did not pass the threshold until after the file transfer activity was beyond the 8-hour measurement window.

5.2.5 Moderate Workload

The third test case we used represented a moderately intense hacker session. It consisted of listing all local documents and desktop directories for one user on the machine. Following the directory list requests, a variety of files including two income tax returns (PDF format), one JPG image, three small Word documents, and a text file containing a 1000-address mailing list were all compressed and downloaded. Using a common compression utility, all these files together amounted to a 300 KB zip file. 300 KB it is actually a moderate, if not small, amount of data to download from a compromised machine; it represents less than 1/100,000 the disk space available on a common 40 GB hard drive.

The alert logs for the moderate workload were almost the same as those or the minimal workload. The only difference between alerts is that the moderate workload surpassed the highest daily bandwidth usage threshold of 80 KB instead of the just the 40 KB seen during the minimal workload. The moderate workload did take longer than the minimal workload to complete. The difference, however, was from two minutes to ten minutes. Since Web Tap records bandwidth in 5-minute intervals, this did not really change results from other filters.

6. WEB TAP FILTER VULNERABILITIES

A serious question that comes to mind when evaluating Web Tap is whether an HTTP tunneling or backdoor program could be designed to evade Web Tap. This section describes vulnerabilities that could be exploited to avoid detection by each of the Web Tap's filters.

- *Single Request Size Filter:* Large data transfers can be broken up into multiple smaller requests (though that risks violating other filters if the requests are not scheduled carefully).
- *Delay Time Filter:* The delays could be randomized so as to bypass thresholds (though this can still trip day-of-time filter, if the user is not usually active at that time.)
- *Time-of-day Filter:* The hacker can schedule requests when a user is normally active (though that increases the risk of detection by the user.) Of course, if the user is an insider who wishes to leak data via a tunnel, avoiding this filter is straightforward.
- *Request Regularity Filter:* If the hacker knows the thresholds used, he can attempt to stay below them by keeping a running count of activity. If the hacker does not know the thresholds used, then this filter could be avoided by emulating the regularity of a common site.
- *Bandwidth Limit Filter:* A tunneling program can keep a running count of the bytes that have been sent over the past day. If the count is going to exceed the detection threshold, it can stop sending data. This requires knowledge of the thresholds.

In general, it is hard for Web Tap to detect a tunneling program that closely mimics the browsing patterns of a legitimate site. This can be accomplished by logging all outbound web requests and seeing which site is visited the most by the user (this may be done by compromising the OS or the web browser). The backdoor can then issue requests of similar size at similar time

intervals to make sure that its traffic does not exceed the regularity measurement for a legitimate site.

Even though Web Tap can be evaded by copying the browsing patterns of a legitimate site, the hacker would have to be careful that the site being mimicked does not generate a false positive. If the site generates a false positive, and it is copied by a backdoor, then the backdoor will generate a real positive. This is one reason why it is very important to set thresholds low enough so that *some* false positives still occur. Doing so makes it much more difficult for hackers to copy the request patterns of other sites and avoid detection.

It is important to note that if an HTTP tunnel is calling back to multiple computers, then the amount of bandwidth available is going to increase. If a system administrator is worried about this, then it would be possible to set a per user aggregate bandwidth threshold. Web Tap was unable to measure a good number for this, since many of the host names actually translated to multiple computers (at least six in one case) behind one NAT device. This may, however, be something to examine in the future.

Another possible enhancement to Web Tap would be allocating different bandwidth quotas for different groups of sites. Trusted sites that are known to be frequently used can be given higher bandwidth limits than others. This could help mitigate attacks by tunnels that mimic access patterns of high-usage legitimate sites. Such a strategy would allow for the tightening of thresholds for the remaining sites (or groups of them), thus limiting the bandwidth available for covert communication.

7. FUTURE WORK

We note that in our study, aggregating over all the filters for 30 users and using the most aggressive bandwidth filter in our setup of 20 KB/day, 92 false alarms were raised over a 40-day period (average of 2.3 false alarms per day). The number goes down to 57 false alarms over a 40-day period (average of approximately 1.5 false alarms per day) if the bandwidth filter is set at 60 KB/day. The false positive rate should be manageable for smaller groups. For larger groups, however, it could become a concern. There are several techniques to decrease false positive rate, some of which were pointed to in the previous section. One simply way of reducing false positives for all filters is to create a database that keeps track of hosts that tend to set off alarms. If a system administrator wishes to allow users to access AIM express and Weatherbug, for example, then these sites may be added to an ignore list for header, delay time, and request regularity alerts. As noted in the previous section, however, this increases the risk of leakage of data via permitted sites, and this is a tradeoff that system administrators will have to make.

Another intriguing possibility for the reduction of requests and bandwidth usage for hosts is proxy caching (with the proxy cache placed before Web Tap). If proxy caching is enabled, then all the hits do not need to be recorded by Web Tap. Removing cache hits would help isolate web tunnel activity because all of the tunnel requests would miss. The number of legitimate requests going through Web Tap would decrease, while the number of anomalous requests would remain the same thus making them potentially easier to detect.

Another way of reducing false positives for bandwidth filters could be to compress all large transactions. This would dramatically reduce the size of large requests that would normally generate false positives. Just to give an example of the effects of compression, a 3.87 KB POST request that triggered an alarm was compressed to 2.07 KB, nearly half of its original size. Compression could help isolate malicious traffic in similar scenarios. Good hackers are also likely to compress and encrypt data before sending it over the network, which prevents it from being compressed any further. Legitimate requests would be significantly reduced in size, while tunnel requests would not, thus increasing the chances of the bandwidth filter catching true positives.

8. CONCLUSION

In contrast to prior work that analyzes attacks on web servers, Web Tap focused on traffic from internal hosts to the outside network in order to detect covert HTTP tunnels and spyware. We presented the design of Web Tap, an anomaly detection system that can augment an HTTP proxy server to aid in the detection of spyware and outbound covert HTTP traffic on an otherwise secure network. Web Tap consists of several anomaly detection filters. The filters were configured using data from approximately 30 users over a one-week training period. During a 40-day evaluation period with the same users, Web Tap detected many spyware programs that used outbound HTTP requests from compromised clients to send out information. Web Tap was also successful at detecting several HTTP tunneling programs, including a test backdoor program.

Different types of tunneling programs set off different alarms. Data-mining software that sends out personal information tended to set off bandwidth usage alarms. Browser search bars that made frequent requests, but operated at times of user activity, were caught by the request regularity filter. Once detected, spyware can be blocked or removed to increase system performance and security. Web Tap can also help detect attempts by hackers to communicate with backdoors on compromised machines during normal inactivity times through usage profiling. In addition, Web Tap detected frequent callbacks from HTTP tunneling programs, even during times of normal usage.

We also considered the question of what vulnerabilities exist in Web Tap's filters. In general, evasion of Web Tap’s filters requires the adversary to know several details, such as Web Tap's thresholds, a user's normal activity schedule, as well as the typical browser and platform used on the workstations. Then, the backdoor must be designed to stay below those thresholds. Another way that Web Tap can be evaded by the adversary is to monitor and analyze a user’s outbound traffic then mimic the access patterns of a legitimate site. Even in that case, the amount of bandwidth available will be still limited by Web Tap’s bandwidth filters and the attacker cannot guarantee that he is not detected since the legitimate site could trigger a false positive. We also discussed ways to extend Web Tap to further limit the bandwidth available to a hacker using such a strategy.

While intrusion detection is a never-ending battle between hackers and security professionals, Web Tap places significant barriers in front of hackers and spyware designers who target computers behind proxy servers or restrictive firewalls. Web Tap produced manageable number of false positives per day for our 30-user group. We also discussed how to manage larger groups by suggesting additional mechanisms that can help reduce false positives (such as creating a list of trusted sites that are ignored by Web Tap). Overall, Web Tap proved to be very effective at reducing both inadvertent and deliberate information leakage via outbound HTTP requests.

9. ACKNOWLEDGMENTS

Thanks to students from the University of Michigan who participated in the study. We also appreciate feedback from Patrick McDaniel and the reviewers on earlier versions of the paper. This research is supported in part by grants from the National Science Foundation (grant CCR 0082851) and gifts from Microsoft and Intel.

10. REFERENCES

[1] Ad-Aware, http://www.lavasoftusa.com/software/adaware/, 2004.

[2] D. Barbara, R. Goel, and S. Jajodia. Mining Malicious Data Corruption with Hidden Markov Models. *16th Annual IFIP WG 11.3 Working Conference on Data and Application Security*, July 2002.

[3] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, Changes in Web client access patterns: Characteristics and caching implications, *BU Computer Science Technical Report*, BUCS-TR-1998-023, 1998.

[4] J. Berman, Prepared Statement of Jerry Berman, President, the Center For Democracy & Technology On the SPY BLOCK Act, *Before the Senate Committee On Commerce, Science, And Transportation Subcommittee on Communication*, March 2004.

[5] BlackICE PC Protection, http://blackice.iss.net/, 2004.

[6] CERT Vulnerability Note VN-98.07, http://www.cert .org/vulnotes/VN-98.07.backorifice.html, October 1998.

[7] CERT Advisory CA-2003-22 Multiple Vulnerabilities in Microsoft Internet Explorer,http://www.cert.org/advisories/ CA-2003-22.html, August 2003.

[8] B. Cheswick, An Evening with Berferd in which a cracker is Lured, Endured, and Studied, *USENIX proceedings*, January 1990.

[9] D.E. Denning, An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222-232, February 1987.

[10] B. Duska, D. Marwood, and M. J. Feeley, The measured access characteristics of World Wide Web client proxy caches, *Proc. of USENIX Symposium on Internet Technology and Systems*, December 1997.

[11] A. Dyatlov, Firepass, http://www.gray-world.net/pr_firepass. shtml, 2004.

[12] A. Dyatlov, S. Castro, Wsh 'Web Shell', http://www.gray-world.net/pr_wsh.shtml, 2004.

[13] EyeOnSecurity, http://eyeonsecurity.org/advisories/Gator/, 2002.

[14] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol HTTP/1.1, *RFC 2616*, June 1999.

[15] S. Forrest, A. Hofmeyr, A. Somayaji, and T. A. Longstaff, A Sense of Self for Unix Processes, *Proc. of the IEEE Symposium on Security and Privacy*, pp. 120-128, May 1996.

[16] A.K. Ghosh, J. Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions Against Programs. *Proc. of the Annual Computer Security Applications Conference (ACSAC'98)*, pp. 259-267, December 1998.

[17] S. Hisao, Tiny HTTP Proxy, http://mail.python.org/ pipermail/python-list/2003-June/168957.html, June 2003.

[18] Hopster, http://www.hopster.com/, 2004.

[19] H.S. Javitz and A. Valdes. The SRI IDES Statistical Anomaly Detector, *Proc. of the IEEE Symposium on Security and Privacy*, May 1991.

[20] T. Kelly, Thin-client Web access patterns: Measurements from a cache-busting proxy, *Computer Communications*, 25(4):357--366, March 2002.

[21] C. Kruegel, T. Toth, and E. Kirda. Service-specific Anomaly Detection for Network Intrusion Detection. *Symposium on Applied Computing (SAC)*, ACM Scientific Press, March 2002.

[22] C. Kruegel and G. Vigna, Anomaly Detection of Web-based Attacks, *Proceedings of ACM CCS'03*, pp. 251-261, 2003.

[23] T. Lane and C.E. Brodley, Temporal sequence learning and data reduction for anomaly detection, *Proc. of the 5th ACM Conference on Computer and Communications Security*, pp. 150-158, 1998.

[24] J. McHugh, "Covert Channel Analysis", *Handbook for the computer Security Certification of Trusted Systems*, 1995.

[25] MIMEsweeper,http://www.mimesweeper.com/products/msw /msw_web/default.aspx, 2004.

[26] I. S. Moskowitz and M. H. Kang, Covert channels --- Here to stay?, *Proc. of COMPASS '94*, pp. 235 - 243, 1994.

[27] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Proc. of the 7th Usenix Security Symposium*, January 1998.

[28] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACMTransactions on Networking*, 3(3), pp. 226-244, June 1995.

[29] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, Information hiding --- A survey, *Proceedings of the IEEE*, special issue on protection of multimedia content, 87(7):1062-1078, July 1999.

[30] S. Saroiu, S. D. Gribble, and H. M. Levy, Measurement and Analysis of Spyware in a University Environment, *Proc. of the First Symposium on Networked Systems Design and Implementation*, pp. 141–153, March 2004.

[31] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. *Proc. of the USENIX LISA '99 Conference*, November 1999.

[32] Spybot – Search and Destroy, http://www.safer-networking .org/, 2004.

[33] SpywareBlaster,http://www.javacoolsoftware.com/ spywareblaster.html/, 2004.

[34] K. Tan and R. Maxion. Why 6? Defining the Operational Limits of Stide, an Anomaly-Based Intrusion Detector. *Proc. of the IEEE Symposium on Security and Privacy*, pp. 188-202, May 2002.

[35] Websense, http://www.websense.com/products/about/ howitworks/index.cfm, 2004.

[36] N. Ye, Y. Zhang, and C.M. Borror. Robustness of Markov chain model for cyber attack detection. *IEEE Transactions on Reliability*, 52(3), September 2003.

[37] Y. Zhang, V. Paxson, "Detecting Backdoors", *Proc. of the 9th USENIX Security Symposium*, August 2000.