Global Information Assurance Certification Paper

Edgar D. Glasheen
GSEC Practical ver 1.4b - Option 2
October 2003

Low- to No-Cost Methods to Review Webserver Logs for Potential Security Issues

1. Abstract

This is a description of the inexpensive methods I devised to extract and tally records of interest in order to analyze webserver logfiles for potential security problems, compromise attempts, while also obtaining IP address statistics. The tools used are the Unix fgrep utility and Analog, the freely distributed logfile analysis tool for multiple platforms. These techniques can easily be adapted to the logfiles produced by almost any platform and use other text extraction utilities than fgrep. If implemented on another site, the adaptation process will require some analysis of the logfiles of a given site as well as additional customization to eliminate "false positives" without the introduction of "false negatives."

The purpose of this monitoring on my site was to observe the compromise attempts used on the site and replicate them on the internal development system which mirrors the internet-facing production site. This replication should verify whether the attempt had any harmful effects on the production platform. This development platform runs on a server whose hardware and software are the same as the production webserver, with no connection to the internet. It is used as a "quality assurance" server in the final stage of pre-production testing. I have been able to verify that all of the suspicious requests that apply to our platform appear to be unsuccessful.

2. Before Snapshot

In March 2000, I was webmaster for the public website of the XYZ organization. I was assigned the task of automating the process of tallying and reporting hitcounts (counts of website HTML page, application, and other file requests by the browsing public) to management and the various program areas that contributed content to the website. Several weeks of logfiles from the organizations' webserver were analyzed.

The initial analysis of the HTTP requests revealed some unpleasant facts. It was apparent that port 80 was being used as an attack vector far more often than we had suspected. These compromise attempts consisted of programmatic means as well as simple manipulation of the contents of the browser URL entry field. The suspicious activity discovered included attempts to "fingerprint" the webserver or application, directory traversal attempts, buffer overflows, attempts to"break" applications by passing crafted URLs as well as attempts to compromise the server by using known webserver vulnerabilities. The skill level of the compromise attempts varied from the rudimentary to the highly sophisticated. There did not appear to be any compromises of the webserver as of the time of the initial logfile analysis.

My organization diligently followed all the best practices known at the time, as we continue to do today. Only the minimum ports necessary for website functionality were allowed through the firewall. Only the minimal services needed for functionality were installed on the server and no modules for streaming media or FTP services were installed. The IT staff monitored the firewall and other system logs, but the webserver logs were only irregularly monitored. Given the level of potentially malicious activity, it was determined that monitoring on a weekly basis (at the minimum) should begin as soon as possible.

It also became obvious that "eye-balling" the logfiles to look for compromise attempts was arduous, time consuming, and prone to missing records of interest. In March of 2000, the weekly logs consisted of an average of 80,000 records per week with an average file size of 10 megabytes. By August of 2003 the average was 960,000 records weekly, with an average file size of 130 megabytes. Even the initial 80,000 records represented a considerable challenge for unaided inspection.

3. During Snapshot

My analysis showed many parallels between the proposed inspection and that performed by an intrusion detection system (IDS). There are two primary differences between the two processes. One is that an IDS dynamically monitors system logs and transactions in a real-time mode while the webserver log file is analyzed after it has been spooled to disk for a given period (post de facto analysis.) Another significant difference is that an IDS ideally examines all network activity, while this analysis is limited to webserver logfiles.

The free application Analog[1], written and maintained by Stephen Turner of Britain, had been chosen to provide the hitcount statistics. It also offered the option to tally requests by file type, IP address and HTTP return codes - valuable information with the potential to help monitor suspect behavior. In addition, it was decided that developing or purchasing a program or script to extract the records with potentially malicious input would both expedite the process and lead to greater accuracy. While the use of an IDS had been discussed when the need for webserver activity had become apparent, the immaturity of the technology, the cost, and the fact that an IDS was potentially vulnerable to both compromise and being deceived by techniques such as packet fragmentation had led to the choice of a less sophisticated solution with far less cost in monetary, time and manpower expenditures.

4. Suspicious Record Extraction Method and Optimization Issues

a. Suspicious Record Extraction

[1]Turner, S. http://www.analog.cx/

Several possible methods and their associated costs in money, time and effort were considered. Commercial programs like Webtrends were considered, as well as an internally-developed program in some language like Perl, C++ or Visual Basic. Considering both the expense of Webtrends as well the fact that multiple vulnerabilities had been found for it, I chose not to utilize it. I decided the best filtering and extraction method would be the use of a UNIX script with multiple calls to the fgrep utility to extract files of suspicious record types. Since the initial expenditure for this approach was minimal, the overall cost to a small organization lies primarily in time and effort. If this solution proves to be inadequate, the organization has not invested a large sum into non-refundable hardware and software. I presented this solution to my management, who felt that this method provided the functionality needed for our purposes. The task of implementing this solution fell to me.

It become apparent that breaking down these abnormal records into categories aided in analysis. An output file was created for each category as detailed below. Further analysis indicated that some of the strings that potentially indicated compromise attempts also appeared in some “normal" record requests, so that one or more passes to extract the "false positive" HTTP requests would be needed to facilitate analysis. The fgrep utility features the ability to exclude records with given strings, so no other utility was needed for this function. (See “A Brief Discussion of fgrep”)

The extracted records are then imported into a spreadsheet such as OpenOffice Calc, Quattro Pro or some equivalent package. (See Importing Logfile Extracts into a Spreadsheet). Once imported, it is possible to sort the records by IP address, time and date, HTTP request string, HTTP return code and any other field available in a given logfile that one wishes to examine.

b. File Optimization

If the logfiles generated by your webserver(s) are large, it may be advantageous to strip out request for graphic elements that occur in HTML pages. The advantages are that the processing time required and the storage needed to retain an archived version of the abbreviated file is greatly reduced. The only possible disadvantage is the removal of a request for a graphic file that may contain malicious content. In three years of analysis, I have not encountered any recognizable examples of such. On my site, which has been recently redone to have up to 30 graphic elements per page (from a previous maximum of 8), the reduction in file size has ranged from 50% to 72%.

This may be done by adding the following fgrep line to the beginning of the script parselog.bat: (or an equivalent line - as is mentioned later, this can be de done using a different file extraction utility if it meets the reverse extraction criteria mentioned in “A Brief Description of fgrep”) :

fgrep -f -v graphics.lst mmddyyyy.log > mmddyylog_abbr.log,

and replacing all subsequent references to mmddyyyy.log with mmddyylog_abbr.log.

The list of graphic file extensions (graphics.lst) will vary from site to site. If your organizational standard restricts the graphics files used on your site to a given file format - JPEG (.jpg), GIF, (.gif) or PNG (.png) filtering can be limited to the appropriate string representing the file extension. If multiple graphics file formats are supported, one must use multiple input arguments or a input file with the graphics files extensions in use at your site in the extraction file.

c. Eliminating "False Positives" While Not Introducing "False Negatives"

In the early stages of implementing my system to extract potentially suspicious records for examination, the issue of "false positives" and "false negatives" arose, emphasizing the similarity of this analysis to intrusion detection systems.

As defined in Andy Cuff's Intrusion Detection Terminology (Part One)[2], a "false positive" or "false alarm" is defined as "An event that is picked up ....and declared an attack but is actually benign." "False positives" in this context would refer to normal HTTP requests that contain a sequence of characters that we associate with one of our extraction categories (potentially suspicious requests). For example, one of the sequences extracted as part of the programmatic strings was "bin". This would also include the character sequence "binghamton". As our site is in New York State in the United States of America, and we do receive and supply information on a geographic basis, these are filtered from the programmatic extract file ONLY. If the record also has a suspect return code or control character, it would be extracted by the scripts that look for those record categories. Thus, a record with the string "binghamton" (case insensitive) is stripped from the extract file. The string "etc" as it would appear in an attempt to either access or update the "/etc/passwd" file would also find all records with the name "fletcher" as well. Thus, records with the string "fletcher" would be extracted from the programmatic string extract only. This is done for the reason mentioned above, that this record may contain the string "binghamton" or "fletcher" but also contain control character(s), embody a directory traversal attempt, have a unusual return code or be from an IP address on our "watch list". This approach can be likened to multiple security cameras monitoring a physical site - if one camera misses suspicious individuals or activities, the others will likely record it.

The removal of possible "false positives" should not be made too granular or universally applied to all of the extract files, as overly ambitious removal of possible "false positives" may result in the removal of records of interest - i.e, they will be classified as "false negatives", and not written to the extract file(s). Andy Cuff's article describes this phenomena as a "false negative" or "miss" and defines it as:

[2]Cuff,A.,http://www.securityfocus.com/infocus/1728

" [it] occurs when an attack or an event is either not detected...or is considered benign by the analyst....Ordinarily the term false negative would only apply to... not reporting an event. However, I have seen this same problem at the analyst level. The scenario is this: the analyst sees a certain signature day after day and knows it to be benign so ignores it. However, one day the IDS alerts on a genuine attack with the same signature. The analyst however chooses to ignore it believing it to be benign, thus a False Negative."[2]

My goal was to reduce the extraction of "false positives" as much as possible, while avoiding at all costs the classifications of records of interest as "false negatives." It should be emphasized in our discussion that the analyst should beware of the natural tendency to allow analysis to become a routine and thus being insufficiently attentive to a character sequence associated with a "false positive" which is accompanied by a character sequence that is actually malicious.

Another source of "false positives" are normal calls to valid website files offered for use to the public that have extensions that are filtered by the programmatic extraction function. For example, your site probably contains files or applications that contain the one or more of the file extensions or directory names that are filtered by the programmatic list. For example, the site may offer a list of rules and regulations of the Robert Goddard Model Rocketry Club in a self-extracting compressed file called "rulesreg.exe", or provide some useful function through the use of an application called "coolapp.cgi". Depending on the contents of the individual site, the same inadvertent extraction may occur with valid references to other file types such as ".php" or ".asp". In order to eliminate these records from unnecessarily appearing in the programmatic extract file, one should extract them from the final programmatic records file. As stated previously, it seems prudent to not eliminate these records from all extract files. They may contain other deviations from the norm which would make them suspect.

The elimination of "false positives" in this case would consist of assigning an intermediate file name to the output of the initial suspicious string/character extraction step, then applying a similar "reverse extraction" step to the one that can be utilized to remove unwanted graphics file HTTP requests. In addition, I feel that is important to monitor attempts to locate and/or exploit non-used webserver system components even though they should be (and hopefully have been) removed from your webserver, irrespective of the platform or software you use. It can be argued that it is important to monitor attempts to locate or access components foreign to your platform or software, in that the potentially malicious party may return with tools or exploits that are specific to your platform or software.

[2]Cuff, A..,http://www.securityfocus.com/infocus/1728

4. Extraction Categories with Brief Descriptions

The record extraction categories used were:

A. Control characters and escape sequences not usually found in standard HTTP requests for website files e.g., ">", "<", ";", "|" (the pipe character), etc.

B. Strings associated with system utilities, e.g. "passwd", "etc", "cgi-bin" on the Unix/Linux platform.

C. Strings or string components associated with traversal attempts: "." or ".."" in combination with "/" and "\" (forward and backslash characters)

D. Requests from IP addresses recorded in a manually-maintained "watch list". This list should be frequently updated as many IP address ranges are assigned to a given party on a strictly temporary basis.

E. Records with Microsoft-specific file- and subdirectory-specific strings in them e.g. "vti_","iis", "MSADC" "default.id?", etc. Since our servers are a non-Microsoft platform there is no reason for such strings to appear in the logs.

F. Records with non-usual HTTP return codes. I extracted records with return codes that were not:

I. 200 - Success
II. 206 - Request not completed - usually a large file that the requestor has lost interest in downloading.
III. 304 - Request Fulfilled from Cache - file is in browser cache
IV. 404 - Not Found - outdated or incorrect URL

5. A Brief Discussion of fgrep

fgrep is a version of of the UNIX grep utility. It may be replaced with any other text scanning and display utility under any other platform that can perform the following functions (The fgrep arguments are given for reference):

A. Display of lines in a file containing a given string of text (fgrep normal operation)
B. Display of lines that do not contain a given string of text ("fgrep -v")
C. The ability to enter a list of target strings in a parameter file ("fgrep -f")
D. The ability to ignore case differences ("fgrep -i")
E. The output of these extraction operations may be redirected to an output text file.

The syntax of fgrep is well documented. The most involved sets of fgrep calls that define a given extraction may be summarized thus:

fgrep -i -f “input text target list file name” “source log file name” > “workfile name1”
fgrep -v -i -f “false positive list filename1” “ work file name1” > “workfile name2"
fgrep -v -i -f “false positive filename2” “workfile name2" > “final extract filename”

More than one set of exclusionary extractions may be performed if the same “false positive” text string removal is to be performed on multiple files. In that case we create a separate file for that group of text strings and apply it as a intermediate step to the multiple categories in question rather than adding them to the primary “false positve” exclusionary extraction file. This provides a minor efficiency in that if editing that exclusion group is necessary, it will only have to be performed on one file.

6. Metacharacters and Hex Characters

The first extraction category I devised was records that contain non-alphanumeric characters that might indicate a compromise attempt. I began by compiling a list of characters that should not normally appear in requests for web content on a given site. These would include characters that might be used in compromise attempts, the hex equivalent thereof (include brief discussion of hex characters and their appearance in HTTP requests), as well as the HTML character and numerical entities.

What characters should be viewed as suspicious? One of my first sources was the odd records themselves. If my knowledge of the static and dynamic content of the site seemed to indicate that the HTTP request was not normal, I would include the odd character(s) in the draft extract list. I also searched the internet for articles that described both “cracking” methods as well as discussions of “sanitizing” input received by internet-facing applications and devices. In early 2000, the articles on breaking applications were more plentiful than those on writing secure code or making existing code secure. The initial list of characters and strings owes much to the 1999 phrack article by rain.forest.puppy, "Perl CGI Problems"[3], in which the author discusses the potential for compromise of Perl and other CGI programs in the passing of crafted character strings that may allow a malicious party to either obtain system information (“fingerprint”) or compromise the webserver running the CGI application. He also details the characters that should be considered for removal in the “sanitization” of input. The initial list that rain.forest.puppy used was taken from the W3C WWW Security FAQ[4], and consisted of the following characters:

& ; ` ' \ " | * ? ~ < > ^ ( ) [ ] { } $ \n \r

I dealt with the forward- and backslash characters in the section on file traversal, so I omitted them from the control character list. The ampersand (“&”) and

[3] rain.forest.puppy, http://www.wiretrip.net/rfp/txt/phrack55.txt

[4] Stein, L. and Stewart, J., http://www.w3.org/Security/Faq/www-security-faq.html

question mark (“?”) occur normally in HTML requests, usually as delimiters in input to applications, so I did not include them either. It could be argued that they should be included, but (at least on my site) the use of input forms to pass parameters to CGI applications produces a unacceptably large number of “false positives.”

Let’s examine the remaining members of this list to see if they warrant inclusion in our list:

“;” - the semicolon - used to sequentially run multiple commands in the Unix/Linux environment. There is no reason for this to appear in an HTML request on the vast majority of webservers, so it deserved a place in my extraction file.

“`” - the backtick character - most commonly used to run commands in Perl scripts. Allowing commands to be run via an HTTP request is most certainly dangerous, as simple URL manipulation may allow some unwanted and certainly dangerous command to execute with the privileges of the Perl application. In “"Fingerprinting Port 80 Attacks: A look into web server, and web application attack signatures"[5], the author, identified as admin@cigisecurity.com , states “The backtick character is often used in perl to execute commands. This character isn't normally used in any valid web application, so if you see it in your logs take it very seriously.” So I certainly wanted to include this character.

“ ' “ - the single quote - can be used in Perl scripts to execute arbitrary commands, as referenced in “CSNews Remote Command Execution Vulnerability”[6] and CAN-2002-0924[7] for a discussion of how this technique was used against the csNews.cgi. So this character is included in my list.

“ ” “ - the double quote - as it appears as a delimiter in all our logfile records I could not use it as an extraction signature. Doing so would result an extract file that would be a full copy of our logfile.

“ | “ - the “pipe” character. rain.forest.puppy[3], in the article cited above, refers to it as “That pesky pipe”, and comments, “In Perl appending a '|' (pipe) onto the end of a filename in a open statement causes Perl to run the file specified, rather than open it.” He gives the example of how, in a less-than-robustly secured system, it would be possible to pass to a perl script a bogus file name that is really a command, such as “"/bin/ls|" - the Unix directory display command followed by the pipe character - and

---

[5] admin@cgisecurity.com, http://www.cgisecurity.com/papers/fingerprint-port80.txt

[6] SecurityFocus, http://online.securityfocus.com/bid/4451

[7] Mitre.org,http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0924

[3] rain.forest.puppy, http://www.wiretrip.net/rfp/txt/phrack55.txt

have Perl execute the command. Obviously the ability to execute commands on the webserver is useful to the cracker and perilous to the integrity of the webserver. So this character was added to my list.

" * " and **"~"-** the asterisk and the tilde - when used as a filename wildcard and local directory shorthand, respectively, have the potential for use in "globbing" overflow attempts. This is described in an explanation of a variety of FTP globbing exploits in a CERT-NL advisory[8]: "...when an FTP daemon receives a request involving a file that has a tilde as its first character, it typically runs the entire filename string through globbing code in order to resolve the specified home directory into a full path. This has the side effect of expanding other metacharacters in the pathname string, which can lead to very large input strings being passed into the main command processing routines. This can lead to exploitable buffer overflow conditions, depending upon how these routines manipulate their input." Two more entries for the list.

" < " and **" >"** - the redirection characters under Microsoft and Unix/Linux versions. These are often used in attempts to append data to files on a webserver, as in the famous RDS exploit by rain.forest.puppy. They are usable in cross-site-scripting attacks and various "include file injections."[5] Two more characters added to this list.

" [" ,"] ", " { " " } " - the bracket characters - as they appear as delimiters in our logfiles, including them in our extraction criteria would produce the same effect as including the double quote character - an extract file identical to the source logfile. So I did not include them.

" ^ ", " (", ") " and " $ " - My research to date has shown no specific usage of the characters, but some of them have occurred in odd HTTP requests received by my site. They were included in my extract file candidates as normal HTTP requests in my site had no occurrences of them.

"\n" and **"\r"** - the C language representation of the carriage return and line feed characters are not included in the control character extraction file as the initial backslash character already appears in the file, making specifically extracting them redundant.

Hex characters - these are the representation of non-alphanumeric character in a URL by the percent sign ("%") followed by the hex number that represents the given character's position in the ASCII collating sequence in the form **%hh**. While some of these characters occur normally in my logs - often generated by non-alphanumeric characters such as "#" or "/" (as in"RD#" and "c/o") included in address information accepted from input pages on our site via a "GET" or "POST", they are also often

---

[8] Schuurman, J., http://cert-nl.surfnet.nl/s/2001/S-01-39.htm

[5] admin@cgisecurity.com, http://www.cgisecurity.com/papers/fingerprint-port80.txt

included in suspicious HTTP requests, such as the Code Red variants (see “Four Examples of Compromise Attempts Revealed by This Method”). Often the hex equivalents of the metacharacters discussed above are substituted in the HTTP request in order to fool intrusion detection systems. In "Fingerprinting Port 80 Attacks: A look into web server, and web application attack signatures”[5], the author states, “If you ever view your logs and notice a large amount of hex, or unusual characters, then its possible an attacker has attempted to exploit your system in some manner.”

The valid range of ASCII characters spans hex 00 (the “poison null”[3] - which can be used subvert Perl scripts) to hex 7F (0 to 127 in decimal notation). Thus my control character extract list looked for the strings “%0" through “%7". You may wish to add additional reverse extraction steps to eliminate “false positives” specific to your site.

For example, if URLS that contain a single space in the directory or filename (permissible under Microsoft webserver software, but not under the Unix/Linux variants) occur on your site you may wish to either not include or reverse extract single occurrences of “%20" - the hex representation of a space. Note - even characters that are usually permitted and considered innocuous, such as the space character are extracted using the default character list enumerated in this paper. The implementer is free to “fine-tune” their extract files as they see fit, but even characters usually considered “harmless” can often be utilized in compromise attempts. Here is a record extracted from a logfile of my site in February of 2003:

```
      xx.xxx.xxxx.xx - - [28/Feb/2003:19:16:44 -0500] "GET
/%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
404 292 404 292 - - 527 161 620 161 0
```

It is not a successful or even very frightening example of a buffer overflow, but illustrates the point that even allowed characters can be dangerous if used in a specific manner with malicious intent.

7. Program-Related Strings

The contents of this extract category can be broken down into two sub-categories: directory names and program names. The strings themselves are almost entirely Unix-Linux oriented, with the exception of the variants on the “password” file/directory names, which occur under many platforms. Microsoft-specific file and directory names are extracted separately.

---

[5] admin@cgisecurity.com, http://www.cgisecurity.com/papers/fingerprint-port80.txt

[3] rain.forest.puppy, http://www.wiretrip.net/rfp/txt/phrack55.txt

The directory names searched for are those where utilities and program reside in many default installations of Unix/Linux. The initial list included “bin”, “etc”, “usr”, “cgi-bin”, “local” and “x11". More were added as they were encountered in research and in the logfiles. Two variation of the “cgi-bin” subdirectory name are included: “cgibin”and cgi_bin”. It is recommended that one use both an extensive directory list as well as an extensive program filename list. A common security practice is to install server components in directories with names other than the installation default. Those directory names should be included in your list in case an intruder gains knowledge of your non-default directory structure and attempts to leverage that knowledge.

A significant contribution from the "Fingerprinting Port 80 Attacks: A look into web server, and web application attack signatures.[5]" was the inclusion of the "chown, chmod, chgrp, chsh " commands, as well as the “uname” and “id” commands. The “ch..” commands are used to change important parameters, such as file and process permissions, and should be monitored even if unsuccessful. The “uname” and “id” commands can be used to obtain the hostname and user name the webserver is running under.

Several variants of the password directory/filename were included (pswd, passwd), as well as the related terms “shadow” and “master” to detect attempts to find the encrypted “shadow” password file or a BSD Unix password file.

The list has been added to since its inception, and has become quite detailed. I am sure that it does not include every possibility, however, and urge the parties that utilize this technique to add any omissions that you encounter.

8. Microsoft-specific Strings

This extraction category is very much like the programmatic category previously described, but with the focus on the Microsoft platform. Again the focus is on requests that reference directory or file names that may indicate attempts to elevate privileges or otherwise compromise the webserver or components. Note that the default extraction strings include “vti_” a default Microsoft directory/file name which on a Microsoft platform may be perfectly innocuous. Fine-tuning may be needed on a Microsoft webserver, to eliminate “false positives” generated by this string, but please refer to the discussion “Many, many, many security holes in the Microsoft Frontpage extensions[9]” to review the potential problems associated with this category of requests on a Microsoft webserver.

9. HTTP Return Codes of Interest

---

[5] admin@cgisecurity.com, http://www.cgisecurity.com/papers/fingerprint-port80.txt

[9] pedward@WEBCOM.COM, http://www.insecure.org/sploits/Microsoft.frontpage.insecurities.html

I have previously stated that I extracted all HTTP requests with return codes that do not correspond with the four that we consider "normal" on our website. The program Analog, previously mentioned, produces a count of all HTTP return codes analyzed in a given run. This chart (with the return codes considered innocuous bolded) shows the results of processing the records for a one-week period on our site:

Count	Return Code	Description
784005	**200**	**OK**
2780	**206**	**Partial content**
881	**302**	**Document found elsewhere**
181142	**304**	**Not modified since last retrieval**
6	400	Bad request
21	403	Access forbidden
10888	**404**	**Document not found**
36	405	Method not allowed
9	413	Request too long
22	500	Internal server error
48	504	Gateway timeout

The section "Four Examples of Compromise Attempts Revealed by This Method" shows some examples of why I felt that extracting HTTP requests with return codes that are non-innocuous is important. In other cases, it often occurred that a request with a return code of "404 - Not Found" may contain suspicious content and would not be extracted by the return code process. Robust and well-maintained extraction lists for control character, programmatic, Microsoft-specific and "watch list" extraction processes should result in it being culled by one of those operations. This seemed to validate my "multiple security camera" concept.

10. How to Import A Logfile Extract Into a Spreadsheet

The extracted files gave a closer view of HTTP requests of interest, but contained many fields that may be of little or no immediate interest, or may not be supported by your webserver. In addition, the actual URI fields often vary greatly in terms of size, so that examination of the raw logfile via a text editor may be quite difficult.

I found that importing the extract files into a spreadsheet and discarding unwanted fields made the records more readable and also allowed analysis by IP address, time/date, HTTP request content and HTTP return code. These were the fields that we felt offered the most usable data. In the interest of vendor-agnosticism, and since logfile formats vary, I describe a generalized approach:

A. Open the logfile extract in the spreadsheet of your choice as a delimited text file. Most spreadsheets will allow the specification of additional delimiters - add

the space character as an additional delimiter. Additional characters may be required depending on your logfile format.

B. Most spreadsheets allow the user to select fields not to be imported. I opted to only select IP address, date/time, actual HTTP request and return code. If your logfiles offer DNS resolution, you may wish to include that field.

C. The analyst can then use the various spreadsheet features to obtain counts of specific record type, requests by IP address and other useful information.

11. After Snapshot

A. Four Examples of Compromise Attempts Revealed by This Method.

I present four of the many interesting groups of HTTP requests that have been noted since the inception of this analysis method on our site. These have been numerous and often very intriguing - easily enough to be the subject of a separate paper. I will start with the two most infamous security issues of 2001 - the Code Red and Nimda . I then look at two from earlier this summer, the compromise of sites using CCBill credit processing software and the Windows Media Services IIS nsiislog.dll ISAPI overflow.

The document "Internet Storm Center" by the Sans Institute and the Internet Storm Center[10] details the initial detection and the excellent analysis of the initial and subsequent version of Code Red. The first logfile record containing the initial version of Code Red was submitted to www.dshield.org on July 12, 2001. It did not appear in my site logs due to the fact noted in the above article that the initial version of the worm had a defective random number generator and every iteration attacked the same IP address sequence. On July 19, the author(s) apparently corrected that fault and the worm was off and running. Here is the first Code Red record received by my webserver:

xx.xxx.xxx.xxx - - [19/Jul/2001:12:39:47 -0400] "GET
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u90
90%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
HTTP/1.0" 403

While my organization was fortunate not to be running IIS, the networking staff certainly viewed this development with alarm. The logs also revealed the succeeding

[10]incidents.org, http://www.sans.org/NS2001/1-9_Inet_Storm_Center.pdf

versions as they propagated. Note that there are no hex characters that are in our list (I immediately added the "default.ida" signature to my Microsoft-specific extract list) the records were detected by the non-normal "403 - Access Denied" HTTP return code.

As described in "Internet Storm Center"[10], Nimda, using a multifaceted "Swiss Army knife" propagation method, began spreading on September 18, 2001. Here is an abbreviated extract of the first records I examined on my site:

HTTP Request	Return Code
"GET /scripts/root.exe?/c+dir	404
"GET /MSADC/root.exe?/c+dir	404
"GET /c/winnt/system32/cmd.exe?/c+dir	404
"GET /d/winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir	404
"GET /_vti_bin/..%255c../..%255c..r, etc.	404
"GET /_mem_bin/..%255c../..%255c../	404
"GET /msadc/..%255c../..%255c../..%255c/...etc.	404
"GET /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%c0%2f../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%%35%63../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%%35c../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir	404
"GET /scripts/..%252f../winnt/system32/cmd.exe?/c+dir	404

Even if one did not have a "suspicious character" extract criteria, many of the records would have been detected and extracted due to the presence of the ".exe" file extension. It should be also noted that if non-normal return codes had been the only extraction criteria, we would have not seen these compromise attempts at all. The utility of the "multiple security camera" concept would seem to be validated.

In late May of 2003, analyzing the "programmatic" extract results, the following sanitized HTTP requests were noted:

xxx.xx.xx.xxx - - [23/May/2003:15:39:57 -0400] "GET /cgi-bin/ccbill/secure/ccbill.log HTTP/1.0" 404 439 404 439 - - 348 161 367 161 0

yy.yyy.yyy.yyy - - [23/May/2003:15:40:03 -0400] "GET /cgi-bin/secure/ccbill.log HTTP/1.0" 404 432 404 432 - - 390 161 439 161 0

zz.zzz.zzz.zz - - [23/May/2003:15:40:06 -0400] "GET /cgi-bin/secure/ccbill.log HTTP/1.0" 404 432 404 432 - - 339 161 358 161 0

nnn.nn.nnn.nn - - [23/May/2003:15:40:06 -0400] "GET /cgi-bin/ccbill/secure/ccbill.log HTTP/1.0" 404 439 404 439 - - 363 161 436 161 0

It is very interesting to note although the four source IP addresses are different, the HTTP requests were all within 10 seconds and all were looking for the directory/file string "/cgi-bin/ccbill/secure/ccbill.log". It could be speculated that they were from

[10] www.incidents.org, http://www.sans.org/NS2001/1-9_Inet_Storm_Center.pdf

"zombie" machines controlled by the same source, a party using "IP Spoofing" or perhaps different crackers trying the same "hot, new 'sploit". Their contemporaneous nature would argue that the first conjecture might be more likely. A Google search led to the www.ccbill.com site, "CCBill, Your Trusted eMerchant Since 1998, Processing Millions Of Transactions A Year. " I noticed that the words "secure" or "security" did not appear on the home page, or indeed any of the many customer, business client or service pages I viewed. They still do not as of the time this paper was written.

A search of the Neohapsis Archives (http://archives.neohapsis.com/) produced an entry dated Thursday, October 04, 2001 from Guy Poizat of www.parstonline.fr , detailing an apparent automated compromise attempt, with one of the entries being "GET /cgi-bin/ccbill/secure/ccbill.log". It included a reply from "agent33 at geeksquad.com" ("at" character removed deliberately.) "agent33" (Steve Halligan) attributed the requests to the open-source vulnerabilty scanner "whisker"[11].

Another entry from Dayne Jordan, dated Wednesday, December 19, 2001 was entitled "*MAJOR SECURITY BREACH AT CCBILL** " Mr. Jordan, apparently a sysadmin at www.completeweb.net - "Website Hosting - Server Hosting - IP Solutions - Data Center Facilities - Since 1995 -" details the apparent compromise of CCBill clients hosted by Completeweb and the insertion of rogue code and IRC bots on them[12].

While possibly automated attempts to find vulnerable CCBilling components on our site continued until late July, I only had to wait until early July to find a possible explanation. The SANS Critical Vulnerability Analysis July 7, 2003 Vol. 2. No. 26[13], "CCBill whereami.cgi Remote Command Execution", assigned a risk value of "HIGH" to the vulnerability of the CCBill whereami.cgi component. I conjectured that the HTTP requests we observed might have been an attempt to test for the presence of the CCBill software on our webserver. It is important to note that Guy Poizat reported this HTTP request in October of 2001.

In late June of 2003, the following two HTTP requests were extracted by the Microsoft-specific, the programmatic and the return code extraction processes:

```
xxx.xxx.xx.xx - - [27/Jun/2003:08:08:42 -0400] "GET /scripts/nsiislog.dll" 403 - - - - - - - - - -
zzz.zzz.zzz.zzz - - [27/Jun/2003:17:37:22 -0400] "GET /scripts/nsiislog.dll" 403 - - - - - - - - - -
```

Internet research revealed a newly-publicized vulnerability in the Windows Media Services installed by default on Microsoft Windows 2000 Server, Advanced Server, and Datacenter Server, and available as an add-on for Windows NT[14]. While

---

[11]Halligan, S., http://archives.neohapsis.com/archives/incidents/2001-10/0022.html

[12]Jordan D., http://archives.neohapsis.com/archives/incidents/2001-12/0208.html

[13]SANS Institute, http://www.sans.org/newsletters/cva/vol2_26.php

[14]Microsoft Corporation, http://www.microsoft.com/technet/security/bulletin/MS03-022.asp

serving streaming media content to clients, Windows Media Service contains a vulnerability in the logging component nsiislog.dll. This component is implemented as an Internet Services Application Programming Interface (ISAPI) extension. If a deliberately malformed request is received, a buffer overrun condition can result,which according to the Microsoft Bulletin, "could enable an attacker to execute code of his or her choice on a computer running IIS with Windows Media Services installed."

This Bulletin also states: "An attacker attempting to exploit this vulnerability would have to be aware which computers on the network had Windows Media Services installed on it and send a specific request to that server. " Given the number of both licit and illicit tools to send requests to webservers in an automated fashion, it is my belief that the reason for the above HTTP request being received by my organization's webserver (and undoubtedly many others - see "Heads up! distributed scans and attacks targeting nsiss.dll"[15] submitted by Russell Fulton on August 7, 2003) was to locate vulnerable IIS servers.

B. Conclusions

This method offers a cost-effective method of monitoring potentially suspicious or malicious activity in HTTP requests received by a webserver via port 80. The effort involved in setting up, customizing and maintaining the extract files, as well as the time spent analyzing and researching HTTP requests of interest are certainly worth the investment. The "return on investment" (ROI) is an up-to-date and constantly changing view of the attempts to compromise your webserver(s). If your installation includes, as mine does, a mirror of the production server(s), it can allow a quick and accurate validation of whether the suspicious input represents a threat. This view also reflects the activity from the entire internet and often provides a preview of important security issues that may become front-page news tomorrow.

While this monitoring cannot eliminate the need for robust security efforts and practices, it allows one to see what site components are of interest to those who wish to compromise your site, and what methods they are currently utilizing. In my organization, it provided me with "hard evidence" of the incessant attempts by those who wished to hijack our webserver for their own personal gain - as a "spam engine", to use it as a gateway to our other systems or to deface the website to gain "cred" with the other "leet". This evidence was valuable in several areas. It gave credence to the "incessant harping on security" (actual quote) that my security-conscious networking co-workers had been often criticized for. It allowed me to more fully document the need for code hardening and input sanitization to the web development staff.

The main drawbacks are the period manual effort involved and the lack of the dynamic, real-time alerts that a true IDS would provide. It can be counter-argued that

[15] Fulton, R., http://archives.neohapsis.com/archives/incidents/2003-08/0090.html

this static analysis system avoids the potential for compromise that an IDS may introduce to the network. I would urge any organization whose site is not monitored by an IDS or whose site security is not provided by a hosting entity to consider implementing it.

Appendix A. A Listing of a generic version of parselog.bat and the extract files.

parselog.bat

```
fgrep -i -f retcodes.lst 24Aug03.log > 24Aug03_ret.txt
fgrep -i -f program.lst 24Aug03.log > 24Aug03_prog0.txt
fgrep -v -i -f okapps.lst 24Aug03_prog0.txt > 24Aug03_prog1.txt
fgrep -v -i 'binghamton'  24Aug03_prog1.txt > 24Aug03_prog.txt
fgrep -i -f ctrlchar.lst 24Aug03.log > 24Aug03_ctrl1.txt
fgrep -v -i -f okchars.lst 24Aug03_ctrl1.txt > 24Aug03_ctrl.txt
fgrep -i -f ntlist.lst 24Aug03.log > 24Aug03_nt.txt
fgrep -i -f traverse.lst 24Aug03.log > 24Aug03_trav.txt
fgrep -i -f watch.lst 24Aug03.log > 24Aug03_watch.txt
```

retcodes.lst - HTTP return codes of interest. Note: that the " - <quote><space> sequence on my logfiles occurs between the actual HTTP request and the return code. It serves to distinguish the return code from a similar sequence of digits in the logfile.

```
" 100
" 101
" 201
" 202
" 203
" 205
" 400
" 401
" 402
" 403
" 405
" 406
" 407
" 408
" 409
" 410
" 411
" 411
" 412
" 413
" 414
" 415
" 416
" 417
" 500
" 501
" 502
" 503
" 504
" 505
```

program.lst - Unix/Linux program/utility file-related character sequences.

cgibin
cgi-bin
cgi_bin
.pl
.cgi
passwd
chmod
chown
chgrp
csh
pwd
pswd
.php
bin
perl
cg..
.exe
/id
uname
.htpasswd
.htaccess
.htgroup
/ls
/ps
/etc

okapps.lst

/anydir/rulesreg.exe
/otherdir/coolapp.cgi

cntrlchar.lst

%0
%1
%2
%3
%4
%5
%6

%7
>
<
!
|
;
`
~
'
*
!

ntlist.lst - Microsoft program/utility file-related character sequences

ntiis
ntis
winnt
iis
sample
script
msadc
htaccess
.idc
.dll
.asp
.php
.ida
.idq
.htr
.htw
webhits
_vti
tools

traverse.lst - File traversal-related character sequences.

/\
./
.\
\..\
/../

Works Cited

adnin@cgisecurity.com. Fingerprinting Port 80 Attacks: A look into web server, and web application attack signatures. Cgisecurity.com. November 2001. URL:http://www.cgisecurity.com/papers/fingerprint-port80.txt . (2 August 2003)

Cuff, Andy. Intrusion Detection Terminology (Part One) SecurityFocus HOME Infocus. 3 September 2003. URL: http://www.securityfocus.com/infocus/1728. (2 October 2003)

Fulton, Russel. Heads up! distributed scans and attacks targeting nsiss.dll. Neohapsis Archives. 7 August 2003. URL: http://archives.neohapsis.com/archives/incidents/2003-08/0090.html. (29 August 2003)

Halligan, Steve. RE: Automated scan-for-webserver-vulns tool ?. Neohapsis Archives. E-Mail to Guy Poizat. 4 October 2001. URL: http://archives.neohapsis.com/archives/incidents/2001-10/0022.html. (29 August 2003)

Incidents.org. The Internet Storm Center. SANS Network Security 2001. 15 October 2001. URL: http://www.sans.org/NS2001/1-9_Inet_Storm_Center.pdf (12 August 2003)

Jordan, Dayne. *MAJOR SECURITY BREACH AT CCBILL** . Neohapsis Archives. 19 December 2001. URL: http://archives.neohapsis.com/archives/incidents/2001-12/0208.html. (29 August 2003)

Microsoft Corporation. Flaw in ISAPI Extension for Windows Media Services Could Cause Code Execution. Microsoft Security Bulletin MS03-022. 25 June 2003. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-022.asp. (3 September 2003)

Mitre.org. CAN-2002-0924. Common Vulnerabilities and Exposures. 30 August 2002. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0924 . (29 August 2003)

pedward@WEBCOM.COM. Many, many, many security holes in the Microsoft Frontpage extensions. Insecure.org. 23 April 1998. URL: http://www.insecure.org/sploits/Microsoft.frontpage.insecurities.html. (29 July 2003)

rain.forest.puppy. Perl CGI problems. Phrack Magazine Vol. 9 , Issue 55. 9 September 1999.URL: http://www.wiretrip.net/rfp/txt/phrack55.txt. (29 July 2003)

SANS Institute. CCBill whereami.cgi Remote Command Execution. Critical Vulnerability Analysis (CVA) - Volume 2, Issue #26. 7 July 2003. URL: http://www.sans.org/newsletters/cva/vol2_26.php. (22 September 2002)

Schuurman, Jacques. Vulnerability (globbing) in various FTP,.CERT-NL S-01-39. 10 April 2001 URL: http://cert-nl.surfnet.nl/s/2001/S-01-39.htm (9 September 2003)

SecurityFocus. CSNews Remote Command Execution Vulnerability. SecurityFocus HOME Vulns Info. 11 June 2002, URL: http://www.securityfocus.com/bid/4451. (28 August 2003)

Stein, Lincoln D. and John N. Stewart. The World Wide Web Security FAQ The World Wide Web Consortium (W3C). Version 3.1.2. 4 February 2002. URL: http://www.w3.org/Security/Faq/www-security-faq.html. (4 August 2003)

Turner, Stephen. Analog: WWW logfile analysis. Version 5.32 . 23 August 2003. URL: http:// www.analog.cx (12 September 2003)