A Lightweight Tool for Detecting Web Server Attacks

Magnus Almgren, Hervé Debar, Marc Dacier
Global Security Analysis Laboratory,
IBM Research Division,
Zurich Research Laboratory,
Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
{maa,deb,dac}@zurich.ibm.com

May 11, 2004

Type of submission: Paper

Contact author: Hervé Debar
Phone: +41-1-724-8499
Fax: +41-1-724-8953
E-mail: deb@zurich.ibm.com

A Lightweight Tool for Detecting Web Server Attacks

Abstract

We present an intrusion-detection tool focused on web server attacks, and describe why such a tool is needed. Several interesting features will be presented, such as the ability to run in real time and to keep track of suspicious hosts, which simplifies the learning of new attacks. The design is flexible and the signatures used to detect malicious behavior are not limited to simple pattern matching of dangerous cgi scripts, but also considers a history of different types of attacks on a host basis to allow detection of a wide variety of malicious behavior. The tool includes mechanisms for reducing the rate of false alarms. We conclude with a discussion of the information gained from deploying the tool at various sites.

1 Introduction

Intrusion-detection systems aim at detecting attacks against computer systems and networks, or against information systems in general. It is difficult to provide provably secure information systems and to maintain them in a secure state for their lifetime and duration of utilization. Sometimes, legacy or operational constraints do not even allow the creation of such a fully secure system from the beginning. Therefore, intrusion-detection systems have the task of monitoring the usage of such systems to detect the apparition of insecure states. They detect active misuse and attempts whether by legitimate users of the information systems or external parties, to abuse one's privileges or exploit security vulnerabilities.

As web servers can be regarded as the electronic front door of a company, they are the most prominent target of attacks. Simply put, there are several ways to break into a web server host [8]. They can be summarized in attacks that target

- the operating system and services other than the web server. In this paper, we assume that the other services are adequately protected, or that the monitored computer serves only as a web server;
- the web server and weaknesses in installed programs executed on the server, where we concentrate on scripts using cgi, the common gateway interface.

In view of the multitude of vulnerabilities associated with the web server, we will present an intrusion-detection tool focused on this service only. This tool has several interesting features which will be introduced in the following sections. In Section 2 we present a well-known vulnerable cgi program to show how easily some of the attacks can be deployed, and then we look at other programs available on the market. In Section 3 we present the requirements imposed on our work, followed by Section 4 where we describe the design concepts. In Section 5, we discuss what we have learned from deploying the tool at real sites. This discussion concerns how the tool has worked with regard to implementation issues as well as with regard to the concepts presented in Section 4. We also include a discussion of the attack patterns discovered. Section 6 points out where the tool will benefit the most from improvements, and Section 7 concludes the article.

2 Background

2.1 Example: the test-cgi program

The main target of our tool is detecting attacks against cgi programs installed at the server, which is of particular interest considering that a number of web server vulnerabilities are related to the default, out-of-the-box installation of the server. Let us take one example of this kind of vulnerability [6]. The NCSA and Apache web servers come with a program called `test-cgi`. It is used to check whether the web server is correctly set up for running cgi programs. Once the server is up, a careful administrator should remove the program, but the script is often left on the computer [4].

The `test-cgi` program lists a few variables in the environment passed to it. As it was originally designed to be used only by the system administrator, no check of the input is made. To run the program, one simply types in the location field of the browser:[1]

`http://vulnerable.host.site/cgi-bin/test-cgi/ExtendedPath?QueryStr` (example 2.1)

The program then lists the value of the query string, as well as the value of the extended path, among other things. The problem arises when a user enters a wildcard (`*`) as the value of the query string. The script then actually prints the contents of the `cgi-bin` directory.[2] If a malicious user sees the contents of this directory, he might obtain information about the presence of other vulnerable programs and is able to launch directed attacks against them.

Although this is a well-known vulnerability and later versions of the program have been patched, many attackers will still check whether the earlier version of the program is present and use it to gain additional

[1]The Extended Path and the Query String are two arguments that can be provided to the invoked cgi script.

[2]More accurately put, it shows the contents of the directory where the file is located.

information about the system. This may lead to a future break-in. By running the monitor on the host, the probe for `test-cgi` would be detected and the system administrator would be warned of the existence of the vulnerable program. The server can be taken off-line before a more serious attack is received.

2.2 Related work

There are many programs available to protect a system from intrusions, but few tools exist that specialize in the analysis of web server log files. A representative, yet not exhaustive, list of tools related to our approach follows. For more information, we refer the interested reader to [2].

WWWstat [3] is mainly a program to collect statistics of the web server usage. This program does not perform intrusion detection per se, but its output can be used for manual intrusion-detection purposes, by allowing abnormal usage statistics to be checked.

Autobuse [9] is a framework for analyzing log files from firewall logs and web server logs. It parses log entries for known attacks and reports them by several mechanisms, such as email.

Logscanner [10] is a framework for analyzing log files where rules can be incorporated. It automatically contacts a responsible person if necessary, and it feeds the logs into functions developed by the user. The development of such functions is considered equivalent to developing the signatures presented in Section 4.2.2.

Swatch [5] analyzes UNIX syslog log files in a similar way as our tool, by grouping similar entries to automate processing.

CyberCop server [7] is a commercial intrusion-detection tool formerly known as WebStalker. This tool includes functionalities for monitoring activity on a web server based on a policy defined by the server's operator, but does not provide log file analysis.

After evaluating the tools, we found several features that are missing.[3] Some of these tools have some knowledge related to web server attacks, but others do not even support the encoding scheme for hexadecimal characters defined in HTTP (as specified in [1]), which means an attacker can easily avoid detection. The language available to express the signatures is limited and restricted to pattern matching. There are methods to filter out false alarms, such as canceling all events from certain domains, but it would be useful to be able to define filters based on other properties. Autobuse allows the specification of a defined threshold per host, which allows reports to be suppressed until a given host has performed several attacks, but there is no distinction of the severity of received events.

Overall, we found a need for a tool written explicitly to detect attacks against the web server. As written, the server is becoming ubiquitous in the computer infrastructure and thus it is important to have sufficient supervision. In Section 3 we present the main concepts we consider particularly important for such a tool.

3 Specification of our Approach

After having evaluated the programs presented in Section 2.2, we had a clear notion of the desired features, which are

- the ability to track hosts that have exhibited malicious behavior, the assumption being that their intentions are malicious and that they might repeat such efforts. By studying the behavior of suspicious hosts, we may deduce new signatures to add to the database – a process that when automated gives the tool the ability to learn new attacks by itself, thus removing one of the major disadvantages of the knowledge-based approach of having to update a database with the latest exploits;

[3]Not all programs lack all of the functionality described below, but none of them had everything. Note that the main objective of these tools may not be to detect web server attacks.

- flexible attack signatures. The signature scheme should allow novice users to put in simple signatures as soon as an attack is published. The scheme should also be powerful enough to craft complex signatures that allow removal of false alarms. The signatures should allow detection of more than malicious cgi scripts;
- modular design for extensibility and efficiency reasons. This enables use of the tool in real time without a service penalty, i.e. without degrading the performance seen from the user's perspective. Moreover, running against "old" log files to verify that they do not contain attacks has to be done in a reasonable amount of time. If the tool is enhanced with reaction capability, i.e. if it implements countermeasures to thwart the attacker's efforts, it is vital that the tool can run in real time.

In principle, the attacks we are interested in can be divided into four areas depending on the intent of the hacker:

1. **Penetration of the system**
    - *Vulnerable cgi programs*: They might be exploitable by meta characters or buffer overflow attacks.
    - *Guessing passwords*: For example, consider the case of a resource protected by a password, but a user keeps failing to access it.
    - *Guessing installed cgi programs*: A user tries to access `/cgi-bin/prog1`, `/cgi-bin/prog2`, `/cgi-bin/prog3`, etc., several thousand times. This is clearly an attack to find out whether the site has any vulnerable cgi programs installed.
2. **Denial of service**
    - Repeated accesses of non-existing resources.
    - Repeated accesses of resources that cause server errors.

   The reason a host fails to access a document *several thousand times* may vary. Even if it is not a hostile attack, an administrator would like to know about it because each request served uses server resources. If it is a broken link, it should be corrected, and if it is a poorly configured robot, the site in question should be informed. This case is less critical than the similar ones described above, where there is more than server resources at stake.
3. **Legal but undesirable activity**
    - *Singular/outlandish use of the HTTP protocol*: Certain behavior may be allowed in the HTTP protocol, but it may also be very undesirable and its use should be questioned. In particular, HTTP specifies the encoding of any character as an hexadecimal value. The only practical use of this feature for "normal" characters is to evade an intrusion–detection system and therefore our monitor looks for this.
    - *Access to sensitive documents*: Some documents should not be accessed through the web server, because their content is confidential. Examples are listings of the `cgi-bin` directory, configuration files of the web server, and password files. However, people can still attempt to request these documents, thus showing a potentially malicious intent.
4. **Security policy violations**
   Every company has a different security policy. For example, an internal web server should only be reached by hosts in the internal network. These hosts comply with a certain name convention (such as `ibm.com`).

The signatures should be flexible enough to detect all the listed families of attacks. Also, because a vulnerable cgi program *and* a successful status code (200) poses a higher threat than merely a failed attempt, it should be possible to combine different pieces of information. As web servers are managed components of an enterprise information system, reports about their behavior should be sent to the management platform.

By deploying the tool at a firewall or a proxy at the perimeter of a company's intranet, outbound traffic can be analyzed. Attacks originating from the inside directed at external sites can thus also be detected. This usage has some impact on the signatures, as the tool is then getting information from another source than a web server, i.e. the success/failure of the request is unavailable.

4 Conceptual Design Description

4.1 Monitor Input Channel

To protect the www server, we need to be able to follow what kind of input is sent to it. As we concentrate on programs executed on the server side, the log files can be used to see what happens. This approach has several advantages:

- *Best source of operation*: The log file contains the exact request that the web server receives, whereas another method (network sniffer, wrapper) has to decompose the packets and interpret the results, which is both time-consuming and error-prone (e.g. in cases of a discrepancy between the interpretations of the URL/request by the monitor and the web server).
- *Provides server portability*: Apart from a custom format, most commercial web servers also support the de facto standard called the *Common Logfile Format* (CLF).
- *Provides platform portability*: Because it merely reads a file, the monitor is very portable between different platforms. Web servers run on almost any platform in existence, and therefore portability is an important issue.

However, this approach also has drawbacks. Anything happening in software layers below the web server is not seen by the monitor. Furthermore, the server has already sent the response, which might prevent certain kinds of reactions, such as shutting down the connection.

The monitor can quite easily be customized to use more information than is available in the CLF format, especially because the Extended Log Format (ECLF) is very similar. However, we decided not to build the first prototype (implemented in Perl) using this format, as most of the log files at our disposal were written according to the CLF format and provided the additional pieces of information present in the ECLF format using separate files.

4.2 The Building Blocks of the Monitor

The structure of the program is represented in Figure 1. It is composed of several building blocks, which perform various checks on the logs. The layered architecture makes it easy to change the functionality of a certain block and extend the scope of the program.

When the program receives a new request, it creates a data structure that encapsulates the log entry. Each module performs certain tests and adds more fields to this data structure, which is *pipelined* through all the blocks. Each subsequent block will be able to use information stored in the object by a previous module. That is, each block merely agglomerates more information to the object passing through. By using this design, new modules can easily be created and inserted into the flow without major modifications. In the following, we will briefly highlight the most important concepts of the existing modules.

4.2.1 Parser Module

The *Parser Module* ensures that a valid request has been written by the web server. It reads the request and breaks it apart according to the fields of the CLF logs (host, date, request, status, etc). It then decodes any characters sent in their hexadecimal form (using the %*dd* syntax) in the HTTP request [1]. Table 1

Figure 1: Layout of the monitor with all the building blocks.

shows the result after a typical log entry has been parsed. As can be seen, the data stored also includes the processing of this module, such as all hexadecimal encodings.

If any of these steps fails, an alert is issued but the request is still passed on to subsequent modules. However, analysis is limited to the data that could be parsed. This can result in reduced or less accurate analysis, because certain modules require certain fields to function properly. For example, if there is no host identification, the *Suspicious–Hosts Module* will not take this log entry into account.

4.2.2 Pattern Module

The *Pattern Module* uses the values stored by the *Parser Module* to look for its attack signatures, and it stores any findings in the object. These signatures have four features. They can

- consist of any regular expression, allowing a novice to simply write the offending name of the cgi script and an expert to use advanced features to limit false alarms;
- match a specific part of the information processed by the *Pattern Module*, e.g. any encoded hexadecimal character found in the request, to simplify the signatures and reduce false alarms;
- be specified as being a signal of an attack, or their presence can be defined as being mandatory for normal requests. For example, normal attack signatures describe malicious behavior and a match indicates such an attempt. On the contrary, signatures enforcing a policy such as allowed IP numbers should always match. The lack of them signifies an abnormal event;
- belong to *classes* grouping similar signatures, which increases speed. To save time, the program does not scan for all vulnerabilities if it finds some attacks, but restricts its search to one match per *class*. Signatures of similar severity should thus be put into the same group. In practice, this means valid requests take slightly longer to treat than nefarious ones, because a valid request requires that all tests be performed.

For example, the request shown in Table 1 would trigger several alarms, depending on the currently available attack signatures. The attempt to run the vulnerable program `phf` would be highlighted, as well as the status code `400 (bad request)`. The presence of the presumably sensitive file `passwd.txt` could also be expressed in a signature.

The module ensures that the overhead introduced by regular expressions is kept to a minimum, in particular by compiling all expressions once at the beginning of execution.

Table 1: Resulting data structure after the *Parser Module* has finished. The actual parsed log entry can be found in the attribute `accessLog`. All of the attributes are searchable in later modules. At this stage, two abnormal events have been reported (shown under the horizontal line), and other modules will add new entries to this list.

Attribute Name	**Value**
reqId	`924182493_1`
accessLog	`hacker.paradise.bad - - [30/Apr/1999:22:25:50 +0200]`
	`"GET /%*7cgi-bin/ph%66?cat%20passwd.txt" 400 -`
host	`hacker.paradise.bad`
rfc931	`-`
user	`-`
date	`30/Apr/1999:22:25:50 +0200`
reqStr	`GET /%*7cgi-bin/ph%66?cat%20passwd.txt`
status	`400`
bytes	`-`
method	`GET`
url	`/%*7cgi-bin/phf`
query	`cat passwd.txt`
version	`0.9`
decodedUrl	`f`
decodedQuery	`[sp]`
suspiciousHexEncoding	`f`
invalidHexEncoding	`%*7`

4.2.3 Combination Module

The *Combination Module* allows the merging of several signatures, as some conjunctional conditions are more dangerous than their separate pieces would suggest. New alerts can be created by Boolean logic of already found suspicious behavior, thereby allowing signatures spanning several attributes of the CLF logs. An elementary example is:

`alert for vulnerable script` $\bigwedge$ `alert for access of password file` $\Longrightarrow$
`generate new serious alert` (example)

4.2.4 Refined Module

The relevance of some attack signatures is contingent on the presence of other patterns. This module allows the creation of attack signatures that will only be used if a previous signature has been found. Either this new signature can be applied to the same field as the prior signature or a new field.

Consider the scenario described in Section 3 with a vulnerable script and a successful status code in the request. As a majority of all requests will be handled successfully by the server, the monitor will waste resources if it continuously looks for successful status codes. Therefore this combined attack signature should be phrased:

`if vulnerable script` $\Longrightarrow$ `look for successful status code` (example)

4.2.5 Suspicious-Hosts Module

The *Suspicious-Hosts Module* checks whether the request originated from a host previously marked as malicious, thus enabling the monitor to keep track of attacking hosts. By studying the new requests and comparing them to the database of the monitor, new attack signatures can be deduced. This alleviates the need of knowing the latest vulnerabilities, because these will be shown to the monitor by the attackers. This is a manual process in the current implementation, but could be automated in the future, at least partially pending review and creation of the appropriate regular expression.

As the request shown in Table 1 triggers several attack signatures, the monitor will save the host name in an internal list. Any other request originating from this host will then be reported.

4.2.6 Trusted Module

Even though carefully written attack signatures limit the number of false alarms, it is impossible to cancel them all by modifying the parameters of the *Parser Module*. For example, a URL containing an attack may still be innocuous if it originates from the system administrator's computer.

The *Trusted Module* removes alerts written by the previous modules if it finds a match for a *trusted signature*. Various levels of trust can be assigned to the signatures to reflect what they are allowed to cancel. A signature encompassing the name of the system administrator's computer can be given permission to cancel all alerts, because this computer is used to probe the network. The path of a Linux distribution including the file `passwd` can give raise to a signature allowed to cancel the alert of only this file being accessed.

4.2.7 Decision Module

The *Decision Module* analyzes the resulting object and decides whether it should be sent on to the management console (file) based on the amount of malicious activity the suspicious host has previously generated. The goal is to prioritize alarms by defining a threshold for reporting. A single instance of a vulnerable cgi script request should be reported immediately, whereas reporting accesses to non-existing documents is done only when the number of requests is associated with a likely denial-of-service attempt.

Each class (as described in Section 4.2.2) has three values associated with it: host-based threshold, domain-based threshold, and validity time. When the tool finds malicious behavior, this module will store information regarding this event, sorted first by the full host name, and then by the class of the signature that was matched. If the internal threshold of either the full host name (`www.ibm.com`) or part of the domain (`ibm.com`) exceeds the threshold for that type of alarm, the abnormal event will be reported to the console (file). By considering both the domain and explicit hosts, the tool can account for attacks generated from a university-like environment or service providers, where users do not have the same host names at every session.

The third parameter regulates how much importance the tool places on recent requests compared to older ones. Each saved state ages, and if enough valid requests are received after a suspicious request, the monitor will place very little value on the latter. This allows users to have some failed accesses to documents (broken links), as long as they also make valid requests.

4.2.8 Print Module

The *Print Module* prints received objects, if any. Depending on the arguments to the program, it will print the resulting output to the syslog facility (real-time use) or the console (running in batch mode).

5 Lessons Learned

5.1 Sites Used for Monitor Evaluation

The program has been run against several sets of log files. In particular, we have made weekly runs against the following web sites:

- two medium-sized commercial sites, with around $7,000$ hits a week for the smaller, and around $58,000$ hits for the larger. The diagrams shown below are results from the smaller site;
- various log files from a university, where the size varied for different web servers, but the two largest had $20,000 - 30,000$ hits per week.

We also experimented with the logs saved from the 1998 Olympic Games in Nagano, which have been recorded in the Guinness Book of World Records as having had almost 650×10^6 requests during its 16 days of operation. The top request rate was 110,414 requests per minute, which is what the medium-sized server above experienced during two weeks of operation. We limited the tests to one day of the Olympic Game logs, which contained some 37 million requests.

The results of the log file analyses are summarized in Table 2. The first column contains a site identifier, instead of the actual host name.[4] Column 2 specifies the type of environment. Column 3 describes the number of log entries analyzed during the supervision time, whereas Column 4 shows the time period spanned. The most interesting column is the fifth, which contains the number of attacks found by the monitor. Column 6 has two parts reflecting the number of hosts the monitor considered suspicious. The first part shows the result when a signature for status code `404 (document not found)` is included, as opposed to the second part, where this signature was omitted from the database.

Table 2: Log files used to test the monitor.

ID	**Type**	**Entries**	**Time** (in days)	**Attacks**	**No. of susp. hosts** with 404	w/out 404
log 1	commercial	104,057	104	82	1,553	48
log 2	Nagano Olympics	37,140,578	1	8	44,584	900
log 3	commercial	3,118,769	455	97	17,435	7,772
log 4	internal commercial	6,203,818	420	11	1,903	1,423
log 5	university	433,515	481	7	2,104	426
log 6	university	879,327	249	10	5,255	516
log 7	university	15,477	320	18	201	27
log 8	university	234,636	58	0	1,768	300
log 9	university	36,668	459	5	290	254
Total:		48,166,950	2,547 (≈7 years)	238	75,093	11,666

The sites described in the table were deliberately chosen from environments with different characteristics to let the monitor experience various types of data. As the supervision lasted well over a year, the log files contain periods of vacation as well as normal semester activity.

The table contains only explicit attacks, such as accesses to vulnerable cgi scripts or requests for sensitive files. Thus, all denial-of-service attacks, password guessing attempts, etc., were excluded. This is not to say that these are not important to monitor, but because their interpretation is subjective we have excluded them from the table. For example, at one of the university sites, a host made more than 40,000 unsuccessful document accesses. As this host also tried to access the file `robots.txt`, we suspect this set of requests was driven by a robot.

In the rest of this section, we will present specific findings. The diagrams are based on the traffic at the smaller commercial site (*log 1*), covering 69 days of monitoring. During these 69 days, the site received 80,030 valid requests, of which 71 were attacks ($< 0.1\%$). We chose this site and time period because it is consistent with the findings at all sites, and the results can be presented concisely.

5.2 Attack Findings

Figure 2 shows the traffic received at the site *log 1* during the first 69 days. Weekends appear as the low–traffic spots in the curve. Numbers above the bar indicate the number of attacks detected for this day, if any. The attacks found by the monitor are shown separately in Figure 3. Each bar represents the number of attacks on each day.

[4]Owing to confidentiality concerns, the actual names have been omitted.

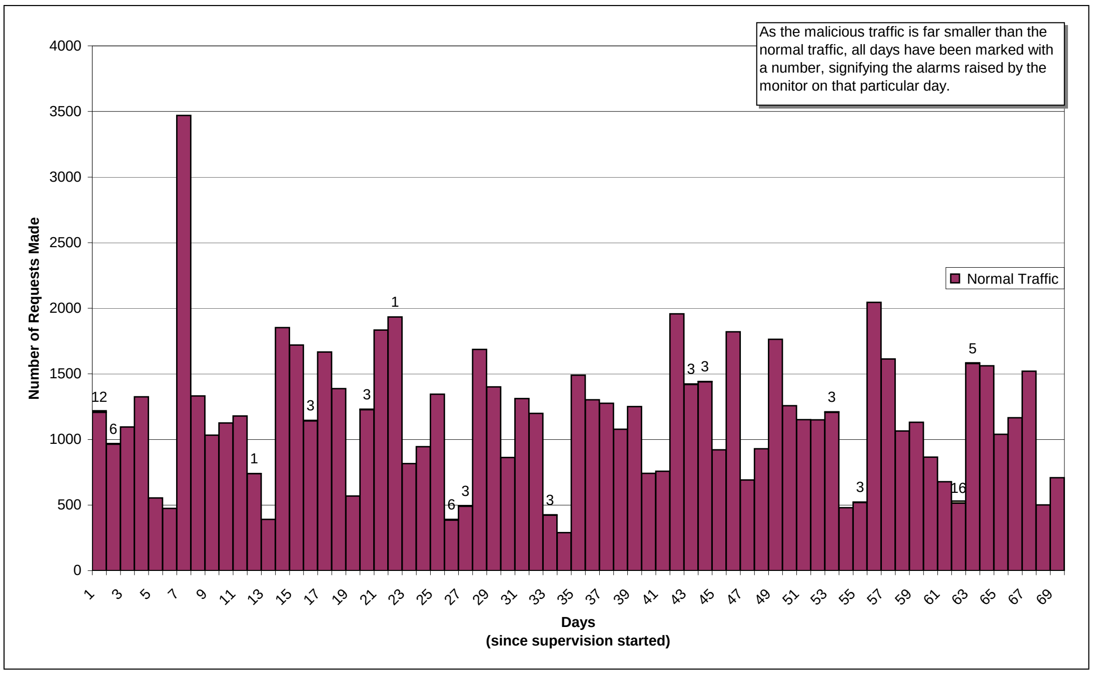

Figure 2: All traffic at a commercial site is shown.

Most attacks are limited to probes for the most obvious and most frequent cgi script vulnerabilities, i.e. `phf`, `test-cgi`, and `handler`. A probe occurs about once every week, even though these vulnerabilities are quite old and well known. These three attacks are usually sent in exactly the order above, and within seconds of each other (see Table 3). This is very similar to the footprint left by the `mscan` scanner (a well-known and widely distributed hacker tool) during our closed lab trials. This scanner also targets lower-level services, including exploiting vulnerabilities found in `statd` and `X`. Other intrusion-detection tools ran on the same network segment, but the number of alerts they produced did not allow us to verify the `mscan` hypothesis by highlighting other scans from the same origin. We are currently working on correlating alarms from various intrusion-detection sensors to realize exactly this functionality automatically. As `phf`, `test-cgi`, and `handler` are old, well-known vulnerabilities,[5] system administrators may not regard the scan as serious if they do not consider the other targets.

In rare cases, a series of vulnerable scripts is tried (see day 1 and day 62) by an automatic scanner targeting the HTTP server. The determination of the fact that the scan is automatic comes from the speed at which the requests are sent. The attacking host always tried at least one of the attack scripts included in the signature file. Therefore, even if the monitor does miss some attacks, it very seldom misses the actual host trying the attacks.

In one case (see Figure 3, day 62), we succeeded in identifying the program used to perform the scans. The hacker had extended the scanner, but the core was still the same. One of these attacks actually led the system administrator to probe the server to verify its security (as can be seen on day 63).

The attacks studied originated from many different countries. From one site, where our supervision only lasted 10 weeks, we detected 16 serious attackers, 8 from the US and 8 from other countries (Norway, Sweden, Brazil, Romania, Switzerland, Italy, Australia, and Chile). The findings shown are convincing arguments that the web server is attacked regularly and that adequate supervision software should be employed.

[5]The `README` file in the scanner package specifically mentions third-world countries as the target for phf, because system administrators may have more serious concerns than this vulnerability.

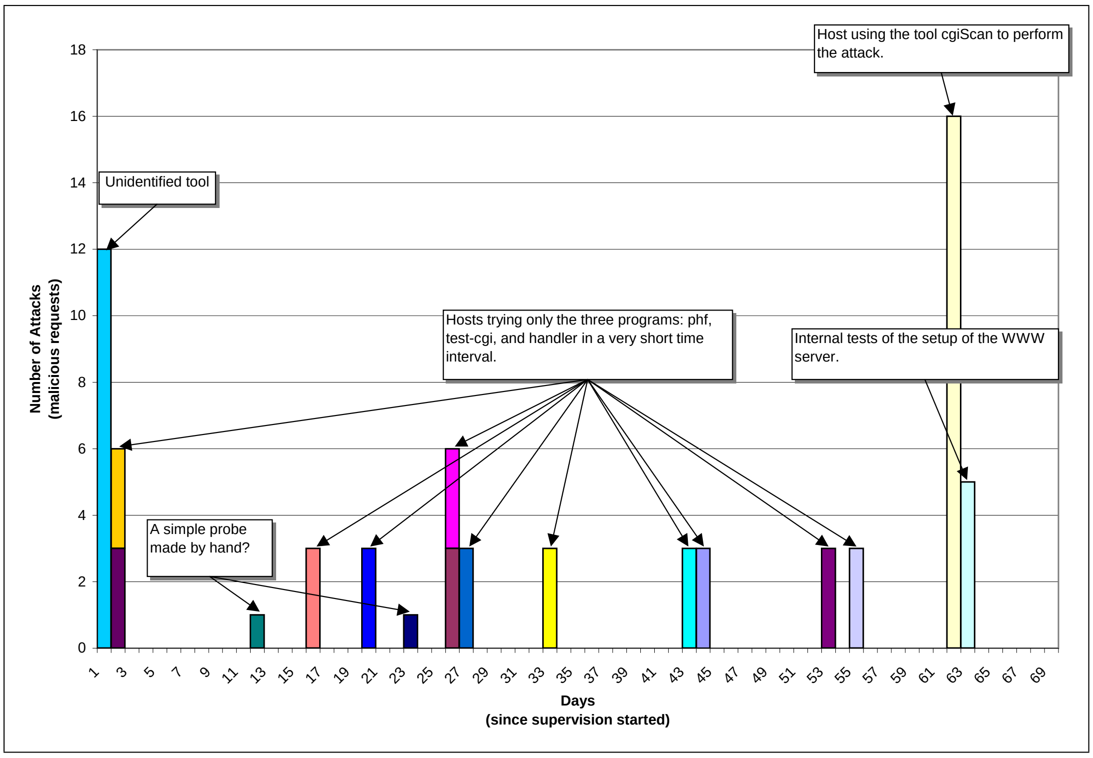

Figure 3: Attack pattern as detected by the monitor.

Table 3: Time between selected requests for three vulnerable programs: `phf`, `test-cgi`, and `handler`.

phf	test-cgi	handler
19:55:13	t+2 s	t+3 s
18:02:49	t+6 s	t+7 s
00:54:23	t+1 s	t+2 s
17:36:29	t s	t+1 s
21:31:43	t+1 s	t+2 s
10:54:06	t+1 s	t+1 s
11:26:36	t+6 s	t+12 s
03:53:27	t+2 s	t+3 s

5.3 Evaluation of Features to Reduce False Alarms

We found it necessary to have a flexible scheme to cancel false alarms. When deploying the monitor at a new site, some innocent requests triggered signatures. By using the additional information available (name of system administrator's computer, explicit evaluation of installed cgi scripts, etc) it was possible to empirically create trusted signatures to cancel the false alarms.

By carefully creating these signatures, the performance of the monitor was improved. Unfortunately, these signatures are specific to particular sites, and not portable.

5.4 Evaluation of the Suspicious-Hosts Concept

The main disadvantage with a knowledge-based approach is that it only detects exploits directed at known vulnerabilities. The effort of keeping an updated database is nontrivial. Thus, one of the most important facets in evaluating our tool was to see whether it is possible to learn about new attacks by tracking suspicious hosts. After one host has made one (unsuccessful) probe, it is likely to continue.

The monitor caught all hosts attacking the supervised site that used at least one known attack. They were added to the *Suspicious-Hosts Module*, and all subsequent requests were reported.

By analyzing alerts reported by the monitor containing only a suspicious host entry but no attack signature, we could determine manually whether the corresponding request was an attack, in which case we retrofit a signature to it. This process was made easier by the fact that most malicious hosts launch only one attack sequence, and then never return to the site. This sequence was sent during a very localized time period, and all the requests were of a malicious nature (i.e. they were not mixed or hidden in normal traffic).

Thus, the overhead of looking at these requests paid off. One attack discovered, `jj`, was first considered as a typo in the client software. Subsequent research showed that it had originally been reported in December 1996.

5.5 Evaluation of the Decision Module Functionality

To allow detection of a wider range of attacks, the monitor must save information about previous requests from suspicious hosts. For example, this allows certain types of denial-of-service attacks to be discovered.

When running the monitor on the smaller commercial site (*log 1*), we have the following figures for the first 69 days:

- Out of 7,049 distinct host names 333 were considered suspicious and 17 (0.24%) reported to the console by the *Decision Module*.
- Out of 80,030 valid requests, 96% of the requests were considered normal, 3.9% as suspicious and only 0.09% were reported (71).

Only a handful of requests are reported and a large majority of hosts only make successful, normal requests. Hence the monitor can pursue suspicious activity without using too many resources on the server. These results are obtained *after* adapting the trusted signatures to the site. There is also only a very small number of documents at the site pointing to nonexistent pages (i.e. broken links resulting in a `404` status code). If there are many broken links, the monitor has to keep track of many more suspicious hosts because each broken link access can be the beginning of a denial-of-service attack.

The signature concerning the status code `404 (document not found)` is the most expensive one in terms of number of matches and, hence, information saved. As can be seen in Table 2, column 6, the differences among hosts tracked by the monitor can be significant, sometimes as much as one order of magnitude with this particular signature present compared to when it is omitted.

The current implementation of the tool does not effectively handle all the matches of this signature. Therefore, depending on the use of the monitor and the memory requirements, the signature for `404 (document not found)` might have to be turned off and possible denial-of-service attacks found through other means.

5.6 Performance Evaluation

Measurement of the monitor performance shows that the average time to process a log entry was $3 - 6$ ms on an RS/6000 43P model 140 200 MHz. Similar analysis on an Intel Pentium II 266 MHz running Linux corroborates these figures. The longer the program runs, the more statistics it has to keep track of, including those from suspicious hosts. We obtained the 6 ms figure while measuring the execution speed on the Nagano logs analysis, which ran on a slightly slower computer.

It should be well within the limits to run the monitor in real time on most web servers that accommodate moderate traffic. To save time, the program does not scan for all vulnerabilities if it finds some attacks. In practice, this means valid requests take slightly longer to treat than nefarious ones, because a valid request requires that all tests be performed.

The memory requirement is more difficult to measure. We examined the amount of memory required by the Perl interpreter for the analyzed logs above containing $80,030$ requests, and found that the Perl interpreter used about 7 MB for the core image of the Perl process.

6 Areas of Usage and Future Work

The monitor's main use is to watch for attacks in real time. It may also be run in batch mode on archived log files if it is impossible to deploy it at the actual site.

It is also possible to deploy the tool at the proxy or at the firewall of a company. By setting up the network properly, all outbound traffic will pass this point and consequently be analyzed. Thus, the tool can enforce company policies, because the signatures created can detect undesirable and malicious behavior equally well. This can be used to restrict surfing to www sites with dubious contents, as well as making sure employees do not leave credit card numbers and passwords in log files. Attacks originating within the company directed towards the outside will be detected as well. This usage has some impact on the program as we are now dealing with another source, e.g. we no longer have access to the success/failure of the request going to the outside.

As specified, the monitor concentrates on cgi scripts and it will not detect attacks against the web server itself nor against the operating system.

After having analyzed the real data presented in Section 5.1, we have identified the following possible improvements:

- The *Suspicious-Hosts Module* would profit from a small buffer of the last requests analyzed. This buffer can be searched if an attack is detected. Attacks that have no matching signature and are being sent by a previously unknown host may be missed. In the current version the analysis of the logs must therefore be rerun. Typically, attacking hosts had very time-localized requests, which facilitates the implementation of this buffer.
- The *Print Module* should be able to summarize the same type of warnings into one message only, so accesses to broken links are printed once and not one message per attempted access.
- By letting the module check for requests that have only been printed because a suspicious host was found, it can directly list possible new attacks. The current version involves searching for these entries by hand.

7 Conclusion

The monitor described in this paper has several interesting features. Foremost, it keeps track of suspicious hosts, which allows it to learn of new attacks by analyzing the request sent by such hosts. Thus, one of the major obstacles to a knowledge-based approach can be alleviated.

It allows the search for flexible attack signatures in any field of the CLF logs. By grouping these into classes, similar attacks can be generalized under one name to save time. It also allows different alerts to be merged, and it will perform refined checks if certain conditions are met. The signatures are not limited to matching simple cgi programs, but extend to detection of denial-of-service attacks.

The design of the tool is modular to allow it to be extended in the future. It is portable between different platforms, and can run in real time. A small subset of all requests is sent to a console for the system administrator to take further actions. The number naturally depends on the web server and the documents, but at one test site, only 71 requests of 80, 030 were sent on. This is well within the abilities of what an administrator can analyze.

We have deployed the monitor at several real sites, and have shown that the concepts described above are sound. Nevertheless, the current implementation may experience memory problems when trying to detect some types of denial-of-service attacks.

The most common attacks affect the three programs `phf`, `test-cgi` and `handler`, and they are probably launched by the scanner `mscan`. These attacks appeared about every week. We also identified a tool used to launch a more elaborate attack. Web servers are probed regularly for weaknesses, which underlines the importance of having adequate supervision in place.

References

[1] Berners-Lee, T., Fielding, R. T., Nielsen, H. F., Gettys, J., and Mogul, J. Hypertext transfer protocol – HTTP/1.1. Internet Request for Comment RFC 2068, Internet Engineering Task Force, Jan. 1997.

[2] Debar, H., Dacier, M., and Wespi, A. Towards a taxonomy of intrusion detection systems. *Computer Networks 31*, 8 (April 1999), 805–822. Special issue on Computer Network Security.

[3] Fielding, R. wwwstat: Httpd logfile analysis software. http://www.ics.uci.edu/pub/websoft/wwwstat/, November 1996.

[4] Garfinkel, S., and Spafford, G. *Web Security & Commerce.* O'Reilly & Associates, 1997.

[5] Hansen, S. E., and Atkins, E. T. Automated system monitoring and notification with swatch. In *Proceedings of the seventh Systems Administration Conference (LISA '93)* (Monterey, CA, November 1993).

[6] Mudge@l0pht.com. test-cgi vulnerability in certain setups. http://www.l0pht.com/advisories/test-cgi-vulnerability, April 1996.

[7] Network Associates Inc. Cybercop server. Available from the company's website at http://www.nai.com/products/security/cybercopsvr/index.asp, 1998.

[8] Rubin, A. D., Geer, D., and Ranum, M. G. *WEB Security Sourcebook.* John Wiley and Sons, 1997.

[9] Taylor, G. Autobuse. Internet, 1998. http://www.picante.com/~gtaylor/autobuse/.

[10] Tuininga, C., and Holak, R. Logscanner. Internet http://logscanner.tradeservices.com/index.html, 1998.