5 VULNERABILITIES USING WORDPRESS – COMMON VULNERABILITIES AND EXPOSURE.

Cybersecurity University- Assignment (Week 7)

MAY 22, 2018

SUBMITTED TO: PROF. ANASTACIA WEBSTER
SUBMITTED BY: TANVIBEN SHAH (005781057)

<u>1) CVE-2018-5776</u>

WordPress before 4.9.2 had XSS in the Flash fallback files in MediaElement (under wp-includes/js/mediaelement). Publish Date: 2018-01-18, Last Update Date: 2018-02-01

- <u>How the exploit was found</u>

  According to CVE Details – The ultimate security vulnerability datasource – "An XSS vulnerability was discovered in the Flash fallback files in MediaElement, a library that is included with WordPress.

- <u>What the exploit does</u>

  The CVE Details - The ultimate security vulnerability datasource described the type of this vulnerability as CWE-79 which means Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). Furthermore, in this type of vulnerability, the software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

- <u>How it was fixed/patched</u>

  It was figured that since the Flash files were no longer needed for most use cases, they had been removed from WordPress. WordPress 4.9.2, an update, was made available. This was regarded as a security and maintenance release for all versions since WordPress 3.7 and users were strongly encouraged to update their sites immediately. Also, MediaElement released a new version that contains a fix for the bug, and a WordPress plugin containing the fixed files was available in the plugin repository. Overall, 21 other bugs were fixed in WordPress 4.9.2, particularly of note were: JavaScript errors that prevented saving posts in Firefox, the previous taxonomy-agnostic behavior of get_category_link() and category_description() was restored, switching themes (will now) attempt to restore previous widget assignments, even when there are no sidebars to map.

Reference:

https://wordpress.org/news/2018/01/wordpress-4-9-2-security-and-maintenance-release/

http://cwe.mitre.org/data/definitions/79.html

2) CVE-2018-10102

Before WordPress 4.9.5, the version string was not escaped in the get_the_generator function and could lead to XSS in a generator tag. Publish Date: 2018-04-16, Last Update Date: 2018-05-18

- How the exploit was found

  The weakness was disclosed 04/16/2018. This vulnerability is known as CVE-2018-10102. A vulnerability classified as problematic was found in WordPress up to 4.9.4. Affected by this vulnerability is the function get_the_generator. The manipulation with an unknown input leads to a cross site scripting vulnerability.

- What the exploit does

  WordPress is eventually prone to a cross-site scripting vulnerability. According to Security Focus, an attacker may leverage this issue of XSS to execute arbitrary script code in the browser of an unsuspecting user in the context of the affected site. This may allow the attacker to steal cookie-based authentication credentials and to launch other attacks. Also, the vulnerability is related to the "get_the_generator" function because the version string is not properly escaped in the function. As mentioned, attackers can exploit this vulnerability easily to execute arbitrary malicious Web script or HTML code in the affected browser, resulting in XSS attacks.

- How it was fixed/patched

  Upgrading to version 4.9.5 eliminates this vulnerability. WordPress versions 4.9.4 and earlier were affected by three security issues which were eventually fixed and implemented in 4.9.5. The fixes were: Don't treat localhost as same host by default, use safe redirects when redirecting the login page if SSL is forced and make sure the version string is correctly escaped for use in generator tags. Overall, 25 other bugs were fixed in WordPress 4.9.5.

  Reference:

  https://wordpress.org/news/2018/04/wordpress-4-9-5-security-and-maintenance-release/

  https://www.securityfocus.com/bid/103775/references

  https://vuldb.com/?id.116227

3) CVE-2018-6389

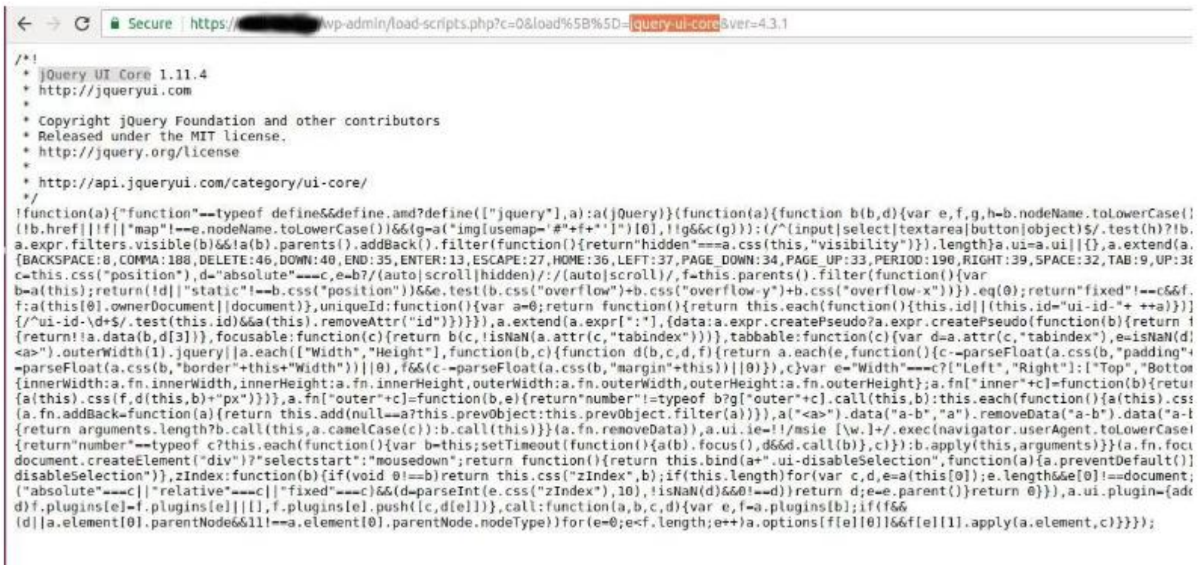
In WordPress through 4.9.2, unauthenticated attackers can cause a denial of service (resource consumption) by using the large list of registered .js files (from wp-includes/script-loader.php) to construct a series of requests to load every file many times. Publish Date: 2018-02-06, Last Update Date: 2018-03-05

- How the exploit was found

  On February 5, an Israeli security researcher, Barak Tawily, discovered a Denial of Service (DoS) attack impacting all 3.x-4.x versions of the Wordpress content management platform. Furthermore, the vulnerability is currently unpatched and relies on a performance boosting feature in Wordpress allowing Javascript and style sheets to be loaded in bulk via a single request. The attack does not affect the Akamai platform, but it does affect any customers using Wordpress unless proper protections are enabled.

- <u>What the exploit does</u>

  The flaw is an application-level DoS issued that affects the WordPress CMS and that could be exploited by an attacker even without a massive amount of malicious traffic. The vulnerability is found in 'load-scripts.php', a script in the Wordpress core code that processes user defined requests (it receives a parameter called load[] with value is ‘jquery-ui-core’). In the response, the CMS provides the JS module ‘jQuery UI Core’ that was requested.



```
Secure | https://[redacted]wp-admin/load-scripts.php?c=0&load%5B%5D=jquery-ui-core&ver=4.3.1

/*!
 * jQuery UI Core 1.11.4
 * http://jqueryui.com
 *
 * Copyright jQuery Foundation and other contributors
 * Released under the MIT license.
 * http://jquery.org/license
 *
 * http://api.jqueryui.com/category/ui-core/
 */
!function(a){"function"==typeof define&&define.amd?define(["jquery"],a):a(jQuery)}(function(a){function b(b,d){var e,f,g,h=b.nodeName.toLowerCase()
(!b.href||!f||"map"!==e.nodeName.toLowerCase())&&(g=a("img[usemap='#"+f+"']")[0],!!g&&c(g))):(/^(input|select|textarea|button|object)$/.test(h)?!b.
a.expr.filters.visible(b)&&!a(b).parents().addBack().filter(function(){return"hidden"===a.css(this,"visibility")}).length}a.ui=a.ui||{},a.extend(a.
{BACKSPACE:8,COMMA:188,DELETE:46,DOWN:40,END:35,ENTER:13,ESCAPE:27,HOME:36,LEFT:37,PAGE_DOWN:34,PAGE_UP:33,PERIOD:190,RIGHT:39,SPACE:32,TAB:9,UP:38
c=this.css("position"),d="absolute"===c,e=b?/(auto|scroll|hidden)/:/(auto|scroll)/,f=this.parents().filter(function(){var
b=a(this);return(!d||"static"!==b.css("position"))&&e.test(b.css("overflow")+b.css("overflow-y")+b.css("overflow-x"))}).eq(0);return"fixed"!==c&&f.
f:a(this[0].ownerDocument||document)},uniqueId:function(){var a=0;return function(){return this.each(function(){this.id||(this.id="ui-id-"+ ++a)})}
{/^ui-id-\d+$/.test(this.id)&&a(this).removeAttr("id")})}}),a.extend(a.expr[":"],{data:a.expr.createPseudo?a.expr.createPseudo(function(b){return f
{return!!a.data(b,d[3])}},focusable:function(c){return b(c,!isNaN(a.attr(c,"tabindex")))},tabbable:function(c){var d=a.attr(c,"tabindex"),e=isNaN(d)
<a>").outerWidth(1).jquery||a.each(["Width","Height"],function(b,c){function d(b,c,d,f){return a.each(e,function(){c-=parseFloat(a.css(b,"padding"+
-=parseFloat(a.css(b,"border"+this+"Width"))||0),f&&(c-=parseFloat(a.css(b,"margin"+this))||0)}),c}var e="Width"===c?["Left","Right"]:["Top","Bottom
{innerWidth:a.fn.innerWidth,innerHeight:a.fn.innerHeight,outerWidth:a.fn.outerWidth,outerHeight:a.fn.outerHeight};a.fn["inner"+c]=function(b){retur
{a(this).css(f,d(this,b)+"px")})},a.fn["outer"+c]=function(b,e){return"number"!=typeof b?g["outer"+c].call(this,b):this.each(function(){a(this).css
(a.fn.addBack=function(a){return this.add(null==a?this.prevObject:this.prevObject.filter(a))}),a("<a>").data("a-b","a").removeData("a-b").data("a-b
{return arguments.length?b.call(this,a.camelCase(c)):b.call(this)}}(a.fn.removeData)),a.ui.ie=!!/msie [\w.]+/.exec(navigator.userAgent.toLowerCase(
{return"number"==typeof c?this.each(function(){var b=this;setTimeout(function(){a(b).focus(),d&&d.call(b)},c)}):b.apply(this,arguments)}}(a.fn.focu
document.createElement("div")?"selectstart":"mousedown";return function(){return this.bind(a+".ui-disableSelection",function(a){a.preventDefault()}
disableSelection")},zIndex:function(b){if(void 0!==b)return this.css("zIndex",b);if(this.length)for(var c,d,e=a(this[0]);e.length&&e[0]!==document;
("absolute"===c||"relative"===c||"fixed"===c)&&(d=parseInt(e.css("zIndex"),10),!isNaN(d)&&0!==d))return d;e=e.parent()}return 0}}),a.ui.plugin={add
d)f.plugins[e]=f.plugins[e]||[],f.plugins[e].push([c,d[e]])},call:function(a,b,c,d){var e,f=a.plugins[b];if(f&&
(d||a.element[0].parentNode&&11!==a.element[0].parentNode.nodeType))for(e=0;e<f.length;e++)a.options[f[e][0]]&&f[e][1].apply(a.element,c)}}});
```

- <u>How it was fixed/patched</u>

  According to WP Security, the best mitigation for this is at the network level. Hosts and WAFs can rate limit this in a way that makes a lot more sense than anything WordPress can do. Caching would also be extremely useful in this case. Limiting the number of scripts that could be loaded at once with those can also help, but the problem with that is all it does is reduce the load by some relatively marginal amount. Some other steps that we can take are: Disable concatenation of JS and CSS files (when using HTTPS, there’s no reason to not use HTTP/2.

With HTTP/2, there's no need to concatenate files. It is an anti-pattern) as well as Protect wp-admin with a static IP VPN.

Reference:

https://blogs.akamai.com/2018/02/wordpress-dos-attack-cve-2018-6389.html

https://bjornjohansen.no/load-scripts-php

https://wordpress.org/support/topic/denial-of-service-warning-with-4-9-4/

4) CVE-2017-16510

WordPress before 4.8.3 is affected by an issue where $wpdb->prepare() can create unexpected and unsafe queries leading to potential SQL injection (SQLi) in plugins and themes, as demonstrated by a "double prepare" approach, a different vulnerability than CVE-2017-14723.

Publish Date: 2017-11-02   Last Update Date: 2018-02-03

- How the exploit was found

  The weakness was presented 11/02/2017. The attack may be launched remotely. No form of authentication is required for exploitation. This vulnerability was found in WordPress up to 4.8.2 and it has been rated as critical having a 7.0 CVSSv3 Temp Score.

- What the exploit does

  WordPress is prone to an SQL-injection vulnerability because it fails to sufficiently sanitize user-supplied data before using it in an SQL query. Exploiting this issue could allow an attacker to compromise the application, access or modify data, or exploit latent vulnerabilities in the underlying database. As reported by Anthony Ferrara, WordPress core is not directly vulnerable to this issue, but they've added hardening to prevent plugins and themes from accidentally causing a vulnerability.

- <u>How it was fixed/patched</u>

  The vulnerability scanner Nessus provides a plugin with the ID 104398 (Debian DLA-1160-1 : wordpress security update), which helps to determine the existence of the flaw in a target environment. It is assigned to the family Debian Local Security Checks. Also, upgrading to version 4.8.3 eliminates this vulnerability. This release includes a change in behaviour for the esc_sql() function.

  <u>Reference:</u>

  https://vuldb.com/?id.108908

  https://wordpress.org/news/2017/10/wordpress-4-8-3-security-release/

<u>5) CVE-2017-9064</u>

In WordPress before 4.7.5, a Cross Site Request Forgery (CSRF) vulnerability exists in the filesystem credentials dialog because a nonce is not required for updating credentials.

Publish Date: 2017-05-18, Last Update Date: 2017-11-03

- <u>How the exploit was found</u>

  The weakness was disclosed 05/18/2017. A vulnerability was found in WordPress up to 4.7.4. It has been declared as problematic.

- <u>What the exploit does</u>

  WordPress is prone to the following security vulnerabilities: 1) An open-redirect vulnerability, 2) Multiple security-bypass vulnerabilities, 3) Multiple cross-site scripting vulnerabilities, 4) A cross-site request-forgery vulnerability. An attacker may leverage these issues to execute HTML and script code in the browser of an unsuspecting user in the context of the affected site, perform certain unauthorized actions, or bypass certain security restrictions. This vulnerability affects an unknown function. Also, the manipulation with an unknown input leads to a cross site request forgery vulnerability and an attacker might be able force legitimate users to initiate unwanted actions within the web application.

- How it was fixed/patched

  Upgrading to version 4.7.5 eliminates this vulnerability. WordPress versions 4.7.4 and earlier were affected by several security issues including Cross Site Request Forgery (CSRF) vulnerability which was discovered in the filesystem credentials dialog as reported by Yorick Koster. The issues seem to have been fixed and implemented in 4.7.5. In addition to the security issues, WordPress 4.7.5 contains 3 maintenance fixes to the 4.7 release series.

  Reference:

  https://www.securityfocus.com/bid/98509/discuss

  https://wordpress.org/news/2017/05/wordpress-4-7-5/