Darren Sun
3035675174
7/7/2022

CS161 Project 1 Writeup

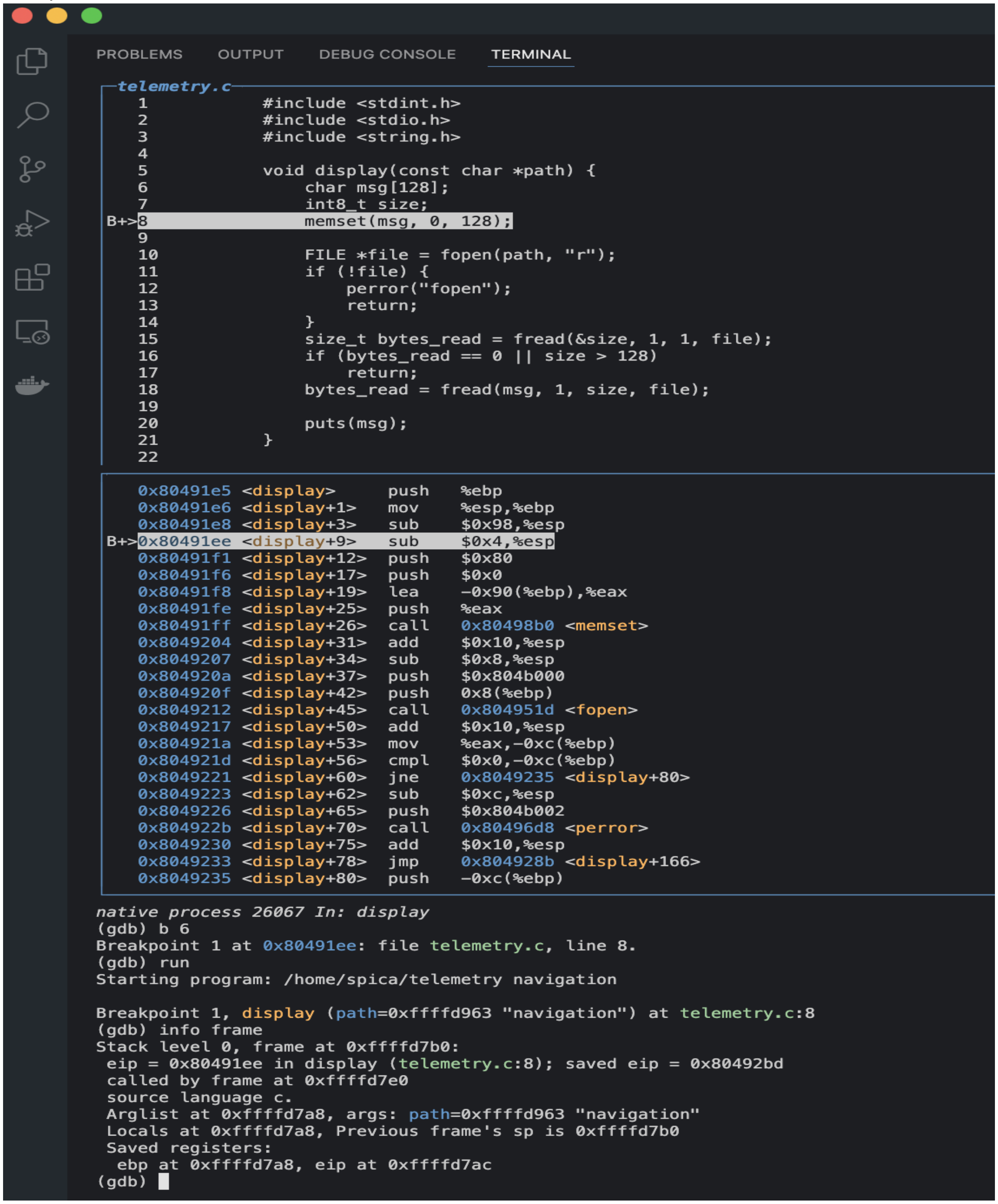
Question 2: Spica
Vulnerability: The vulnerability in telemetry.c was that the call to fread writes a file of size size from the file to the stack.
GDB Processes: I used ./debug-exploit to initiate GDB. Then I used layout split to view the code. I set a breakpoint at line 5 with ‘b 6’, and ran ‘run.’ ‘Info frame’ then revealed that the saved register EIP was 0xffffd7ac and saved EIP was 0x80492bd. The EBP was at 0xffffd7a8, 4 bytes less than the EIP, and 20 Bytes of compiler padding and 128 Bytes of msg followed. The address of msg from ‘p &msg’ was 0xffffd718.
Exploit Structure Description: To exploit this program, I used a print statement to write -1 bytes of a file containing 148 As, the address of shellcode, and the shellcode itself onto the stack to reveal the contents of README. Stack:

[4] RIP Main
[4] SFP Main
[N] Compiler padding
[4] RIP Display (Saved Reg EIP: 0xffffd7ac, Saved EIP: 0x80492bd)
[4] SFP Display (EBP: 0xffffd7a8)
[20] Compiler padding
[128] Msg (Address: 0xffffd718)
[1] size

GDB Output: I made ed posts and showed up to office hours but I could not get it the exploit to work. My final print exploit statement, after a week of trying to figure it out was

print('\xff' + 'A' * 148 + '\xb0\xd7\xff\xff' + SHELLCODE)

This prints me a ton of As.

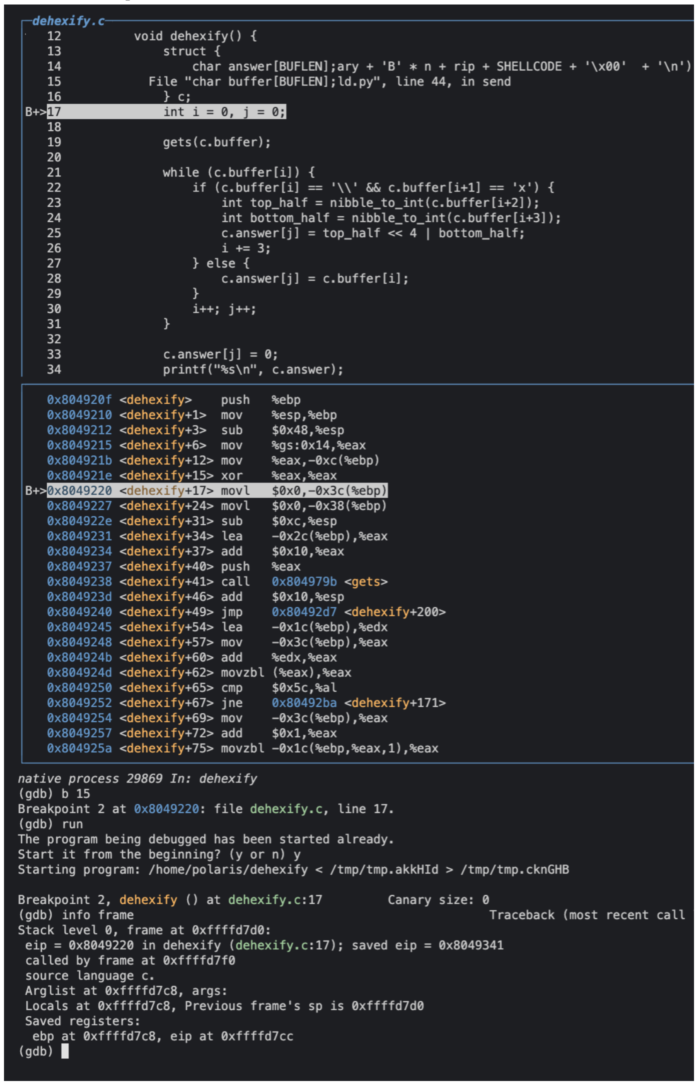


```
telemetry.c
        #include <stdint.h>
        #include <stdio.h>
        #include <string.h>

        void display(const char *path) {
            char msg[128];
            int8_t size;
B+>         memset(msg, 0, 128);

            FILE *file = fopen(path, "r");
            if (!file) {
                perror("fopen");
                return;
            }
            size_t bytes_read = fread(&size, 1, 1, file);
            if (bytes_read == 0 || size > 128)
                return;
            bytes_read = fread(msg, 1, size, file);

            puts(msg);
        }

    0x80491e5 <display>      push   %ebp
    0x80491e6 <display+1>    mov    %esp,%ebp
    0x80491e8 <display+3>    sub    $0x98,%esp
B+> 0x80491ee <display+9>    sub    $0x4,%esp
    0x80491f1 <display+12>   push   $0x80
    0x80491f6 <display+17>   push   $0x0
    0x80491f8 <display+19>   lea    -0x90(%ebp),%eax
    0x80491fe <display+25>   push   %eax
    0x80491ff <display+26>   call   0x80498b0 <memset>
    0x8049204 <display+31>   add    $0x10,%esp
    0x8049207 <display+34>   sub    $0x8,%esp
    0x804920a <display+37>   push   $0x804b000
    0x804920f <display+42>   push   0x8(%ebp)
    0x8049212 <display+45>   call   0x804951d <fopen>
    0x8049217 <display+50>   add    $0x10,%esp
    0x804921a <display+53>   mov    %eax,-0xc(%ebp)
    0x804921d <display+56>   cmpl   $0x0,-0xc(%ebp)
    0x8049221 <display+60>   jne    0x8049235 <display+80>
    0x8049223 <display+62>   sub    $0xc,%esp
    0x8049226 <display+65>   push   $0x804b002
    0x804922b <display+70>   call   0x80496d8 <perror>
    0x8049230 <display+75>   add    $0x10,%esp
    0x8049233 <display+78>   jmp    0x804928b <display+166>
    0x8049235 <display+80>   push   -0xc(%ebp)

native process 26067 In: display
(gdb) b 6
Breakpoint 1 at 0x80491ee: file telemetry.c, line 8.
(gdb) run
Starting program: /home/spica/telemetry navigation

Breakpoint 1, display (path=0xffffd963 "navigation") at telemetry.c:8
(gdb) info frame
Stack level 0, frame at 0xffffd7b0:
 eip = 0x80491ee in display (telemetry.c:8); saved eip = 0x80492bd
 called by frame at 0xffffd7e0
 source language c.
 Arglist at 0xffffd7a8, args: path=0xffffd963 "navigation"
 Locals at 0xffffd7a8, Previous frame's sp is 0xffffd7b0
 Saved registers:
  ebp at 0xffffd7a8, eip at 0xffffd7ac
(gdb)
```

Question 3: Polaris
Vulnerability: The vulnerability here is that the canary can be leaked. We could do this by sending an input that causes the two null bytes appended by gets() to be overwritten. If we run a p.recv(), the answer array and stack canary are both printed out.
GDB Processes:
Exploit Structure Description: The stack looks like this:

%rip
Compiler Padding
Stack Canary
Buffer
Answer

GDB Output:



```
┌dehexify.c──────────────────────────────────────────────────────────────
│   12          void dehexify() {
│   13              struct {
│   14                  char answer[BUFLEN];ary + 'B' * n + rip + SHELLCODE + '\x00'  + '\n')
│   15            File "char buffer[BUFLEN];ld.py", line 44, in send
│   16              } c;
B+>17              int i = 0, j = 0;
│   18
│   19              gets(c.buffer);
│   20
│   21              while (c.buffer[i]) {
│   22                  if (c.buffer[i] == '\\' && c.buffer[i+1] == 'x') {
│   23                      int top_half = nibble_to_int(c.buffer[i+2]);
│   24                      int bottom_half = nibble_to_int(c.buffer[i+3]);
│   25                      c.answer[j] = top_half << 4 | bottom_half;
│   26                      i += 3;
│   27                  } else {
│   28                      c.answer[j] = c.buffer[i];
│   29                  }
│   30                  i++; j++;
│   31              }
│   32
│   33              c.answer[j] = 0;
│   34              printf("%s\n", c.answer);

    0x804920f <dehexify>    push   %ebp
    0x8049210 <dehexify+1>  mov    %esp,%ebp
    0x8049212 <dehexify+3>  sub    $0x48,%esp
    0x8049215 <dehexify+6>  mov    %gs:0x14,%eax
    0x804921b <dehexify+12> mov    %eax,-0xc(%ebp)
    0x804921e <dehexify+15> xor    %eax,%eax
B+> 0x8049220 <dehexify+17> movl   $0x0,-0x3c(%ebp)
    0x8049227 <dehexify+24> movl   $0x0,-0x38(%ebp)
    0x804922e <dehexify+31> sub    $0xc,%esp
    0x8049231 <dehexify+34> lea    -0x2c(%ebp),%eax
    0x8049234 <dehexify+37> add    $0x10,%eax
    0x8049237 <dehexify+40> push   %eax
    0x8049238 <dehexify+41> call   0x804979b <gets>
    0x804923d <dehexify+46> add    $0x10,%esp
    0x8049240 <dehexify+49> jmp    0x80492d7 <dehexify+200>
    0x8049245 <dehexify+54> lea    -0x1c(%ebp),%edx
    0x8049248 <dehexify+57> mov    -0x3c(%ebp),%eax
    0x804924b <dehexify+60> add    %edx,%eax
    0x804924d <dehexify+62> movzbl (%eax),%eax
    0x8049250 <dehexify+65> cmp    $0x5c,%al
    0x8049252 <dehexify+67> jne    0x80492ba <dehexify+171>
    0x8049254 <dehexify+69> mov    -0x3c(%ebp),%eax
    0x8049257 <dehexify+72> add    $0x1,%eax
    0x804925a <dehexify+75> movzbl -0x1c(%ebp,%eax,1),%eax

native process 29869 In: dehexify
(gdb) b 15
Breakpoint 2 at 0x8049220: file dehexify.c, line 17.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/polaris/dehexify < /tmp/tmp.akkHId > /tmp/tmp.cknGHB

Breakpoint 2, dehexify () at dehexify.c:17         Canary size: 0
(gdb) info frame                                                Traceback (most recent call
Stack level 0, frame at 0xffffd7d0:
 eip = 0x8049220 in dehexify (dehexify.c:17); saved eip = 0x8049341
 called by frame at 0xffffd7f0
 source language c.
 Arglist at 0xffffd7c8, args:
 Locals at 0xffffd7c8, Previous frame's sp is 0xffffd7d0
 Saved registers:
  ebp at 0xffffd7c8, eip at 0xffffd7cc
(gdb) 
```

Question 4: Vega
Vulnerability:
The vulnerability in this program is an off by one error. The last byte of SFP can be overwritten.
GDB Processes:
Exploit Structure Description:

Question 5: Deneb
Vulnerability:
In this program, the error checking occurs and then user input is awaited. We can exploit this code and modify the file after the error check and before the file is read in, passing the security length check. We then overwrite the buf, the 20 bytes of garbage, and inject the shellcode.
GDB Processes:
Exploit Structure Description:

1. Set fake info in file that is size checked
2. Wait until after size check is assed and program pauses for user input
3. Edit file contents to contain a filled buffer, 20 bytes of garbage, rip + 4, and the shellcode
4. Tell the program to read in the same number of bytes as you wrote to the file(length of file)

Question 7: Rigel
Vulnerability:
This program can be exploited with a ret2esp attack. We can find a specific value in the function that forces the esp to point to the shellcode at the start of the next execution.
GDB Processes:
Exploit Structure Description: