CS HW2 Writeup

B08901164 林霈瑀
username: lpy

trace

1. Use strace to show the system calls used between the tracer and tracee(cs_2022_fall_ouo). Look for “PTRACE_POKETEXT” to check the addresses we will need to modify (total of 5 occurrences).

strace output example:
Here for example, we need to modify address=0x4012c6

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_TRAPPED, si_pid=6257,
si_uid=1000, si_status=SIGTRAP, si_utime=1, si_stime=81} ---
wait4(6257, [{WIFSTOPPED(s) && WSTOPSIG(s) == SIGTRAP}], 0, NULL)
= 6257
ptrace(PTRACE_GETREGS, 6257, {r15=0x7fcdf346f040, r14=0x403e18,
r13=0x4012be, r12=0x7ffc039545f8, rbp=0x7ffc039544e0, rbx=0,
r11=0x7fcdf3450680, r10=0x7fcdf3435908, r9=0x7fcdf343b040,
r8=0x7fcdf3416f10, rax=0x4012be, rcx=0x403e18, rdx=0x7ffc03954608,
rsi=0x7ffc039545f8, rdi=0x1, orig_rax=0xffffffffffffffff,
rip=0x4012c6, cs=0x33, eflags=0x246, rsp=0x7ffc039544e0, ss=0x2b,
fs_base=0x7fcdf31f9740, gs_base=0, ds=0, es=0, fs=0, gs=0}) = 0
ptrace(PTRACE_PEEKTEXT, 6257, 0x4012c6, [0xe8cbccdeadbeefe8]) = 0
ptrace(PTRACE_POKETEXT, 6257, 0x4012c6, 0x9090909090909090) = 0
ptrace(PTRACE_SINGLESTEP, 6257, NULL, 0) = 0
```

1. Replace `0xe8cbccdeadbeefe8` with `0x9090909090909090` in the tracee using IDA.
2. Undefine and redefine the changed code. Patch program and check that it is running correctly and not showing segmentation fault.

```
ubuntu@ubuntu-2204:~/ubuntushared/hw2/src$ ./cs_2022_fall_ouo
Please
Give me flag
blehblehbleh
Try harder :(
```

3. Decompile and see the logic is clear now.
4. Decrypt flag: `FLAG{TrAc3_M3_1F_U_cAN}`

Code:

```
from Crypto.Util.number import long_to_bytes
enc_flag = 0x0C3F30122E242E37402E423C2E42123003250A36303D37
enc_flag = long_to_bytes(enc_flag)
enc_flag = bytearray(enc_flag)

for i in range(len(enc_flag)):
   enc_flag[i] ^= 0x71
flag = bytes(enc_flag[::-1])
print(flag)
```

pwn_myself

First, the program causes a stack overflow and returns to a read loop after the main function.

```
0x007fffffffe400|+0x0000: "AAAAAAAAAAAAAAAAAAAAAAAAA"    ← $rsp
0x007fffffffe408|+0x0008: "AAAAAAAAAAAAAAAAA"
0x007fffffffe410|+0x0010: "AAAAAAAAA"
0x007fffffffe418|+0x0018: 0x33af2aa142fb1600
0x007fffffffe420|+0x0020: 0x0000000000000000      ← $rbp
0x007fffffffe428|+0x0028: 0x005555555bab6b  →   ret
0x007fffffffe430|+0x0030: 0x005555555ba8bf  →   endbr64
0x007fffffffe438|+0x0038: 0x005555555bab1a  →   endbr64
```

Inside the read loop, we can create input_event struct to understand the logic.

```
unsigned int v1; // [rsp+0h] [rbp-30h] BYREF
unsigned int v2; // [rsp+4h] [rbp-2Ch]
int v3; // [rsp+8h] [rbp-28h]
int fd; // [rsp+Ch] [rbp-24h]
input_event buf; // [rsp+10h] [rbp-20h] BYREF
unsigned __int64 v6; // [rsp+28h] [rbp-8h]

v6 = __readfsqword(0x28u);
v2 = 0;
v3 = 0;
fd = sub_6668F();
if ( fd )
{
  v1 = 0;
  ioctl(fd, 0x80044519uLL, &v1);
  v2 = (v1 >> 1) & 1;
  while ( read(fd, &buf, 24uLL) > 0 )
  {
    if ( buf.type == 1 )                        // EVENT_KEY
    {
      if ( buf.value == 1 )                     // EV_KEY, value 1 : keypress
      {
        if ( buf.code == 0x2A )                 // KEY_LEFTSHIFT = 42
          v3 = 2;
      }
      else if ( !buf.value )                    // EV_KEY, value 0 : release
      {
        if ( buf.code == 0x2A )
        {
          v3 = 0;
        }
        else if ( buf.code == 0x3A )            // KEY_CAPSLOCK = 58
        {
          v2 ^= 1u;
        }
        else if ( off_39B960[v3 + v2][buf.code] )
        {
          copy_and_comparestrings_6650C(off_39B960[v3 + v2][buf.code]);
        }
      }
    }
  }
  close(fd);
```

Initially, I patched the code to bypass the conditions to receive the broadcasted message “Congratulations”. This tells us that the flag is related to data used in the broadcasting functions.

```
0x007fffffffe360│+0x0000: 0x0000001300000000      ← $rsp
0x007fffffffe368│+0x0008: 0x0000003000000000
0x007fffffffe370│+0x0010: "[*] Congratulations"  ← $rsi
0x007fffffffe378│+0x0018: "ratulations"
0x007fffffffe380│+0x0020: 0x00000000736e6f ("ons"?)
0x007fffffffe388│+0x0028: 0x0000000000000000
0x007fffffffe390│+0x0030: 0x0000000000000000
0x007fffffffe398│+0x0038: 0x0000000000000000

   0x5555555ba4cf                  mov    rdi, rax
   0x5555555ba4d2                  call   0x5555555ba344
   0x5555555ba4d7                  mov    edi, 0x1
 → 0x5555555ba4dc                  call   0x5555555b89f0 <exit@plt>
   ↳  0x5555555b89f0 <exit@plt+0>     endbr64
      0x5555555b89f4 <exit@plt+4>     bnd    jmp QWORD PTR [rip+0x3321bd]        # 0x5555558eabb8 <exit@got.plt>
      0x5555555b89fb <exit@plt+11>    nop    DWORD PTR [rax+rax*1+0x0]
      0x5555555b8a00 <fstat@plt+0>    endbr64
      0x5555555b8a04 <fstat@plt+4>    bnd    jmp QWORD PTR [rip+0x3321b5]        # 0x5555558eabc0 <fstat@got.plt>
      0x5555555b8a0b <fstat@plt+11>   nop    DWORD PTR [rax+rax*1+0x0]

exit@plt (
   $rdi = 0x0000000000000001,
   $rsi = 0x007fffffffe370 → "[*] Congratulations"
)

[#0] Id 1, Name: "pwn_myself", stopped 0x5555555ba4dc in ?? (), reason: SINGLE STEP

[#0] 0x5555555ba4dc → call 0x5555555b89f0 <exit@plt>
[#1] 0x5555555ba55c → mov DWORD PTR [rip+0x337886], 0x0        # 0x5555558f1dec
[#2] 0x5555555baa15 → jmp 0x5555555ba927
[#3] 0x5555555bab1a → endbr64
```

Many functions referenced `openssl/crypto/evp/evp_enc.c`. We can match the functions in the file to our decompiled code in IDA.

```
__int64 Encryption_66129()
{
  void *v0; // rax
  unsigned int outl; // [rsp+8h] [rbp-18h] BYREF
  unsigned int v3; // [rsp+Ch] [rbp-14h]
  __int64 *ctx; // [rsp+10h] [rbp-10h]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  ctx = malloc_wrapper_6A860();
  if ( !ctx )
    exit(1);
  v0 = sub_6A1C0();
  if ( EVP_CipherInit_ex_enc_6D8A0(ctx, v0, 0LL, &key_397070, &iv) != 1 )
    exit(1);
  if ( EVP_CipherUpdate_6ABC0(ctx, out_39DE00, &outl, copiedstring_in_39DDC0, 44) != 1 )
    exit(1);
  v3 = outl;
  if ( EVP_CipherFinal_6ADC0(ctx, &out_39DE00[outl], &outl) != 1 )
    exit(1);
  v3 += outl;
  sub_6CB40(ctx);
  return v3;
}
```

```
unsigned __int64 __fastcall Decryption_6622D(char *in, int inl, char *out, _DWORD *size)
{
  void *cipher; // rax
  int outl; // [rsp+2Ch] [rbp-14h] BYREF
  __int64 ctx; // [rsp+30h] [rbp-10h]
  unsigned __int64 v10; // [rsp+38h] [rbp-8h]

  v10 = __readfsqword(0x28u);
  ctx = malloc_wrapper_6A860();
  if ( !ctx )
    exit(1);
  cipher = sub_6A1C0();
  if ( EVP_DecryptInit_6D8B0(ctx, cipher, 0LL, &key_397070, &iv) != 1 )
    exit(1);
  if ( EVP_DecryptUpdate_6B0C0(ctx, out, &outl, in, inl) != 1 )
    exit(1);
  *size = outl;
  if ( EVP_DecryptFinal_6B6E0(ctx, &out[outl], &outl) != 1 )
    exit(1);
  *size += outl;
  sub_6CB40(ctx);
  return v10 - __readfsqword(0x28u);
}
```

```
unsigned __int64 compare_and_broadcast_6643F()
{
  int size; // [rsp+4h] [rbp-6Ch] BYREF
  unsigned int i; // [rsp+8h] [rbp-68h]
  int v3; // [rsp+Ch] [rbp-64h]
  __int64 out_payload[11]; // [rsp+10h] [rbp-60h] BYREF
  unsigned __int64 v5; // [rsp+68h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  v3 = Encryption_66129();
  for ( i = 0; i <= 47 && out_39DE00[i] == comparestring_397040[i]; ++i )
    ;
  if ( i == 48 )
  {
    Decryption_6622D(&in, 32, out_payload, &size);
    broadcast_66344(out_payload, size);          // Congratulations
    exit(1);
  }
  broadcast_66344(out_39DE00, v3);
  return v5 - __readfsqword(0x28u);
}
```

The program encrypts the data from the keyboard to `out_39DE00`, then it is compared with `comparestring_397040`.

With the key, iv and encrypted data, we can decrypt it to get the plaintext.

I guessed that `EVP_aes_128_cbc()` was used as the algorithm.

Code:

```
// g++ test.cpp -lssl -lcrypto
// FLAG{I5_tH15_cHA1l3nGe_t0O_eA5Y_0R_tO0_HARD}
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/aes.h>
```

```
int main(){
    unsigned char in[] =
"\xD2\xB2\x40\xF2\xDE\x77\xE0\x85\xFD\xE5\xBF\xB1\xEB\xF7\x64\x18\xE4\xAD\x85\xEF\x80\x68\xDA\x2C\x25\x2D\xE1\xF8\xDD\xE7\x0B\x59\xE8\xD7\x57\x37\x2F\xB5\x41\x25\x78\x5A\xB9\x82\x22\x8D\x81\x26";
    unsigned char iv[] =
"\xB5\x9A\xEC\x92\x51\xE2\x5E\x3F\x90\x81\xE4\x27\x19\x2E\x50\x29"
;
    unsigned char key[] =
"\x4B\x29\x47\x0F\x38\xD4\xA3\x4D\x1C\x9F\x4F\xC7\x74\xE4\x29\x6A"
;
    unsigned char * out = (unsigned char *)malloc(sizeof(in)*2);
    int outl = sizeof(out) / sizeof(unsigned char);
    int inl = sizeof(in) / sizeof(unsigned char);

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    EVP_CIPHER_CTX_init(ctx);
    EVP_DecryptInit(ctx, EVP_aes_128_cbc(), key, iv);
    EVP_DecryptUpdate(ctx, out, &outl, in, inl);
    printf("%s", out);
    EVP_DecryptFinal(ctx, out, &outl);
    EVP_CIPHER_CTX_free(ctx);
}
```

Reference:
https://github.com/majek/openssl/blob/master/crypto/evp/evp_enc.c
Discussed with: LJP, b08901162

ooxx

Patch the program so that we can bypass the winning condition to get the flag message.
We do this by replacing the `call OWins` instruction with `nops (0x90)`.
Before patch:

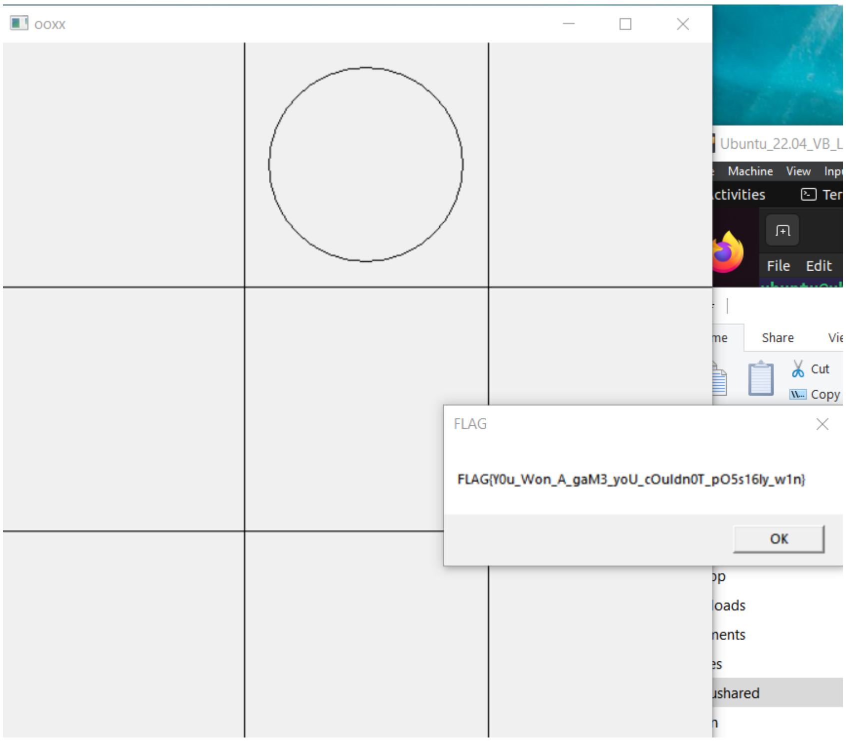
```
__int64 Result()
{
...
  if ( XWins() )
  {
   ...
  }
  else if ( OWins() )
```

```
  {
   // show flag
  }
  else if ( Tie() )
  {
   ...
  }

...
```

After patch:

```
__int64 Result()
{
...
  if ( XWins() )
  {
   ...
  }
  else
  {
   // show flag
  }
  return 1i64;
}
```

This way after each round, the flag will be printed.



Discussed with: LJP

trojan

From the source code, we can tell that trojan.exe takes screenshots and stores them under the temp folder. It also starts a new processthread and acts like a server. The packet payload is xored with a key.

1. Write a simple client to connect to the server and decrypt the payload. The client sends `"cDqr0hUUz1"` to trigger the server. We see "PNG" at the beginning of the decrypted payload → PNG format. This is the screenshot trojan.exe took.

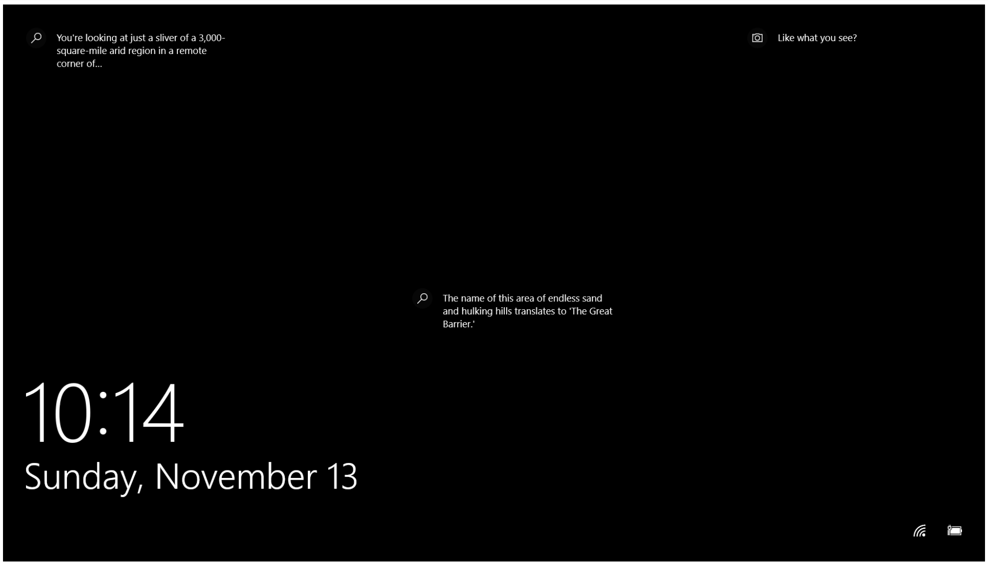
Decrypted payload:

```
b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x07\x800F1\x13Xl
\x04\x07\xf2\x
...
```

client.py:

```
import socket
from Crypto.Util.number import bytes_to_long
from PIL import Image
import io

HOST = '127.0.0.1'
PORT = 19832

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
mystring = "cDqr0hUUz1".encode()
s.send(mystring)
dlen = bytes_to_long(s.recv(1024))

data = b''
while True:
    indata = s.recv(1024)
    if len(indata) == 0: # connection closed
        s.close()
        print('server closed connection.')
        break
    data += indata
data = bytearray(data)

key = "0vCh8RrvqkrbxN9Q7Ydx".encode()
key += b'\x00'
key = bytearray(key)
```

```
for i in range(len(data)):
    c = data[i]
    c = c ^ key[i % 21]
    data[i] = c

stream = io.BytesIO(data)
img = Image.open(stream)
img.save("test.png")
```



2. Parse the log.pcap, decrypt TA's payload and get the flag png file.
Code:

```
from scapy.all import *
from PIL import Image, ImageFile
import io
from Crypto.Util.number import bytes_to_long

key = '0vCh8RrvqkrbxN9Q7Ydx'.encode()
key += b'\x00'
key = bytearray(key)

f = rdpcap('log.pcapng')

data = b''
count = 0
for p in f:
    if p[TCP].payload:
        d = bytes(p[TCP].payload)
        if count >= 2:
```

```
            data += bytes(p[TCP].payload)
        else:
            count += 1

img_data = []
data = bytearray(data)

for i in range(len(data)):
    k = key[i % 21]
    c = data[i]
    c = k ^ c
    img_data.append(c)
img_data = bytes(img_data)
img_data = io.BytesIO(img_data)
ImageFile.LOAD_TRUNCATED_IMAGES = True
img = Image.open(img_data)
img.save("flag.png")
```

Discussed with: b08901162

dropper

1. Unpack the file: `upx -d .\dropper.exe -o dropper_unpacked.exe`
2. In the main function, we can see an encryption is being done on many char buffers. Use x64-dbg to break after encryption is done to get the values. They are module names and function names.
3. The PEB is traversed to match the module and function names. We can get the function pointers.
4. With the APIs, we can see that some sort of encryption is being done.
5. Patch the Sleep function to sleep for 0 seconds.
6. Run in x64-dbg again. Monitor the address of the `Block` variable in the dump section.
7. The flag appears in the dump section.

The last part of my decompiled code in IDA:

```
...
  encrypt(Sleep, 6u);
  v31 = Copy(v59, Sleep);
  v32 = Copy(v46, Kernel32_dll);
  Sleep_func = PEB_Related(v32, v31);
  if ( !(CryptAcquireContextW_func)(&v76, 0i64, 0i64, 1i64, 0) )
  {
    if ( (GetLastError_func)() != 0x80090016 )
      return 0;
    if ( !(CryptAcquireContextW_func)(&v76, 0i64, 0i64, 1i64, 8) )
      return 0;
  }
  if ( !(CryptoCreateHash_func)(v76, 32772i64, 0i64, 0i64, &v75) )
    return 0;
  if ( !v75 )
    return 0;
  if ( !(CryptHashData_func)(v75, &unk_14000B048, 1i64, 0i64) )
    return 0;
  if ( !(CryptDeriveKey_func)(v76, 26625i64, v75, 1i64, &v77) )
    return 0;
  (CryptDestroyHash_func)(v75);
  (Sleep_func)(0i64);
  LODWORD(Size) = 30;
  Block = malloc(0x1Eui64);
  if ( !Block )
    return 0;
  memset_wrapper(Block, Size);
  data_copy(Block, Size, &unk_14000B050, Size);
  if ( !(CryptEncrypt_func)(v77, 0i64, 1i64, 0i64, Block, &Size,
Size) )
    return 0;
  if ( (RegCreateKeyA_func)(-2147483647i64, "CS_2022", &v61) )
    return 0;
  if ( !(RegSetValueExA_func)(v61, "CS_2022", 0i64, 1i64, Block,
Size) )
  {
    (RegCloseKey_func)(v61);
    free(Block);
  }
```

```
return 0;
```

The flag in the dump section: