Graham Dungan
March 24, 2025
UIN: 332001764

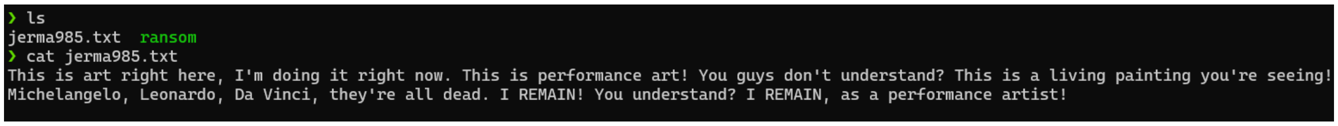
CSCE 413: Software Security
Class 25: RNGs

Video Demo

Due to the nature of this assignment, there is no demo file associated with this report. Instead, a video demonstration of my process can be found here;
https://youtu.be/Lg4AcF2OmuQ
The files `jerma985.txt`, `jerma985_xor.txt` have been included. `jerma985_xor.txt` has been XOR'd with the time value `0x67e448ff`.

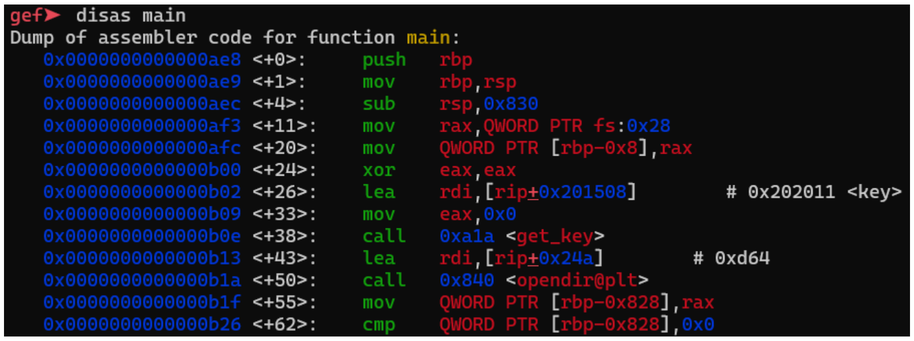
Disassembling

The provided file `ransom` will target and encrypt other files within the same directory when run. Within the same directory, there is a file named `jerma985.txt` such that its contents are as follows;



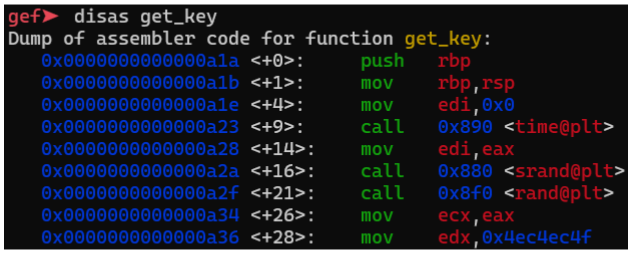
```
❯ ls
jerma985.txt  ransom
❯ cat jerma985.txt
This is art right here, I'm doing it right now. This is performance art! You guys don't understand? This is a living painting you're seeing!
Michelangelo, Leonardo, Da Vinci, they're all dead. I REMAIN! You understand? I REMAIN, as a performance artist!
```

We can use gdb (more specifically, I am using GEF) to analyze the behavior of the program while it runs. In doing so, we can hope to view what the key is and how it is used.
We will first disassemble the `main` function of `ransom` to look for any helpful functions.



```
gef➤  disas main
Dump of assembler code for function main:
   0x0000000000000ae8 <+0>:     push   rbp
   0x0000000000000ae9 <+1>:     mov    rbp,rsp
   0x0000000000000aec <+4>:     sub    rsp,0x830
   0x0000000000000af3 <+11>:    mov    rax,QWORD PTR fs:0x28
   0x0000000000000afc <+20>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000000b00 <+24>:    xor    eax,eax
   0x0000000000000b02 <+26>:    lea    rdi,[rip+0x201508]        # 0x202011 <key>
   0x0000000000000b09 <+33>:    mov    eax,0x0
   0x0000000000000b0e <+38>:    call   0xa1a <get_key>
   0x0000000000000b13 <+43>:    lea    rdi,[rip+0x24a]        # 0xd64
   0x0000000000000b1a <+50>:    call   0x840 <opendir@plt>
   0x0000000000000b1f <+55>:    mov    QWORD PTR [rbp-0x828],rax
   0x0000000000000b26 <+62>:    cmp    QWORD PTR [rbp-0x828],0x0
```

While there are many important instructions for a ransomware function within `main` such as file reads and writes, the most important is the function that generates the key, `get_key`. We can similarly disassemble `get_key` to see how it was generated,

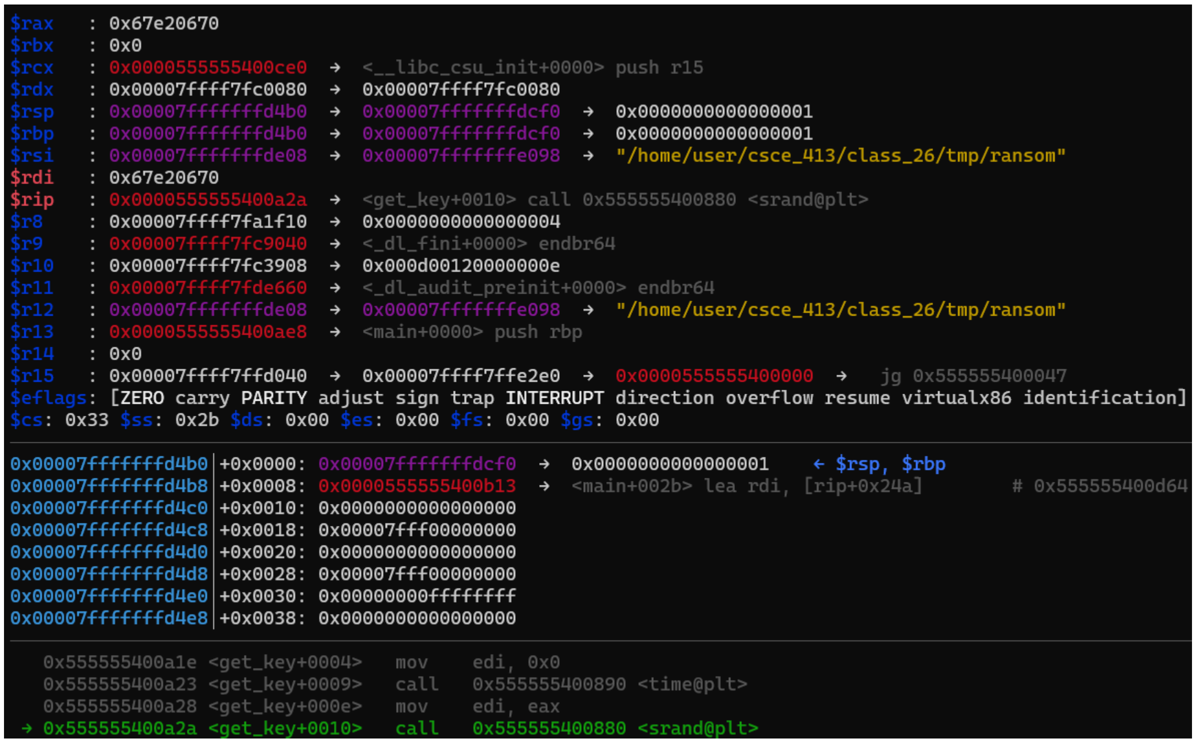


```
gef➤  disas get_key
Dump of assembler code for function get_key:
   0x0000000000000a1a <+0>:     push   rbp
   0x0000000000000a1b <+1>:     mov    rbp,rsp
   0x0000000000000a1e <+4>:     mov    edi,0x0
   0x0000000000000a23 <+9>:     call   0x890 <time@plt>
   0x0000000000000a28 <+14>:    mov    edi,eax
   0x0000000000000a2a <+16>:    call   0x880 <srand@plt>
   0x0000000000000a2f <+21>:    call   0x8f0 <rand@plt>
   0x0000000000000a34 <+26>:    mov    ecx,eax
   0x0000000000000a36 <+28>:    mov    edx,0x4ec4ec4f
```

As seen on the instruction located at `+9`, a function to get the current timecode is called. After this, `srand` is used to generate a seeded random number. We can assume that the output of `+9` is stored in `$rax` and is then moved to `$rdi` (arg 1) for `srand`.

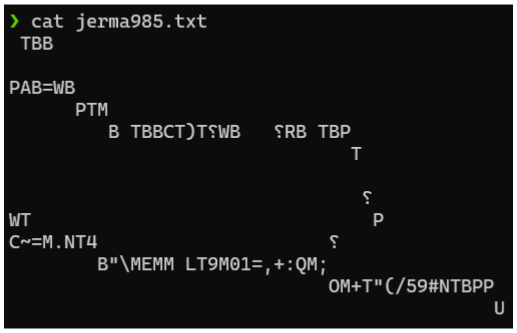
get_key Behavior

We can run the program in gdb, observing what value is passed into `$rdi`. This will encrypt the `jerma985.txt` file in the same directory, so we can hope to use this value to XOR it again later.
Creating a breakpoint at `get_key`, we can save the value passed into `$rax`.



```
$rax   : 0x67e20670
$rbx   : 0x0
$rcx   : 0x0000555555400ce0  →  <__libc_csu_init+0000> push r15
$rdx   : 0x00007ffff7fc0080  →  0x00007ffff7fc0080
$rsp   : 0x00007fffffffd4b0  →  0x00007fffffffdcf0  →  0x0000000000000001
$rbp   : 0x00007fffffffd4b0  →  0x00007fffffffdcf0  →  0x0000000000000001
$rsi   : 0x00007fffffffde08  →  0x00007fffffffe098  →  "/home/user/csce_413/class_26/tmp/ransom"
$rdi   : 0x67e20670
$rip   : 0x0000555555400a2a  →  <get_key+0010> call 0x555555400880 <srand@plt>
$r8    : 0x00007ffff7fa1f10  →  0x0000000000000004
$r9    : 0x00007ffff7fc9040  →  <_dl_fini+0000> endbr64
$r10   : 0x00007ffff7fc3908  →  0x000d00120000000e
$r11   : 0x00007ffff7fde660  →  <_dl_audit_preinit+0000> endbr64
$r12   : 0x00007fffffffde08  →  0x00007fffffffe098  →  "/home/user/csce_413/class_26/tmp/ransom"
$r13   : 0x0000555555400ae8  →  <main+0000> push rbp
$r14   : 0x0
$r15   : 0x00007ffff7ffd040  →  0x00007ffff7ffe2e0  →  0x0000555555400000  →   jg 0x555555400047
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00

0x00007fffffffd4b0│+0x0000: 0x00007fffffffdcf0  →  0x0000000000000001    ← $rsp, $rbp
0x00007fffffffd4b8│+0x0008: 0x0000555555400b13  →  <main+002b> lea rdi, [rip+0x24a]        # 0x555555400d64
0x00007fffffffd4c0│+0x0010: 0x0000000000000000
0x00007fffffffd4c8│+0x0018: 0x00007fff00000000
0x00007fffffffd4d0│+0x0020: 0x0000000000000000
0x00007fffffffd4d8│+0x0028: 0x00007fff00000000
0x00007fffffffd4e0│+0x0030: 0x00000000ffffffff
0x00007fffffffd4e8│+0x0038: 0x0000000000000000

   0x555555400a1e <get_key+0004>   mov    edi, 0x0
   0x555555400a23 <get_key+0009>   call   0x555555400890 <time@plt>
   0x555555400a28 <get_key+000e>   mov    edi, eax
 → 0x555555400a2a <get_key+0010>   call   0x555555400880 <srand@plt>
```

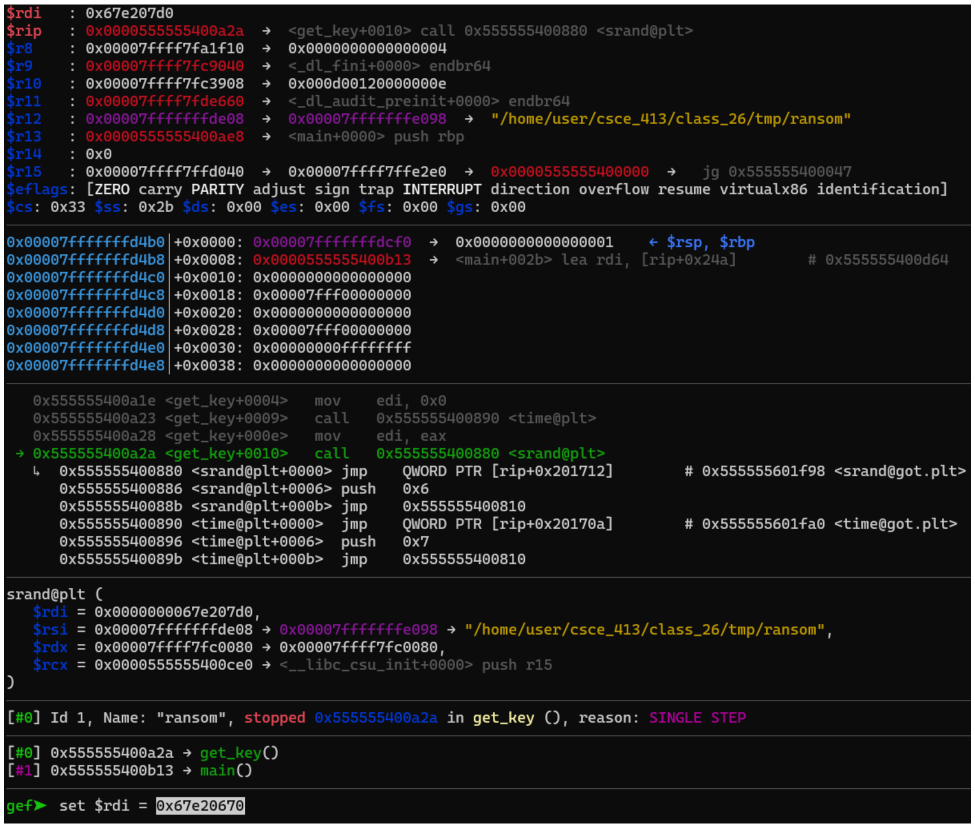
After the execution of the `mov` instruction at the instruction `get_key+000e`, we can find that both `$rax` and `$rdi` contain the value `0x67e20670`.
Once the program has terminated, we can find that the `jerma985.txt` file has been mangled,



```
❯ cat jerma985.txt
 TBB

PAB=WB
      PTM
         B TBBCT)T؟WB   ؟RB TBP
                               T

                                ؟
WT                               P
C~=M.NT4                     ؟
        B"\MEMM LT9M01=,+:QM;
                             OM+T"(/59#NTBPP
                                            U
```

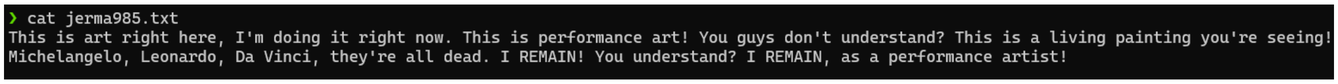
Double XOR

We can undo this encryption by applying the XOR operation again with the same key. Even though running the program again will generate a new key, we can break before `srand` is called to replace any existing time code with the previous `0x67e20670`. Using gdb, we can run `set $rdi 0x67e20670` to set the value in the register,



```
$rdi   : 0x67e207d0
$rip   : 0x0000555555400a2a  →  <get_key+0010> call 0x555555400880 <srand@plt>
$r8    : 0x00007ffff7fa1f10  →  0x0000000000000004
$r9    : 0x00007ffff7fc9040  →  <_dl_fini+0000> endbr64
$r10   : 0x00007ffff7fc3908  →  0x000d00120000000e
$r11   : 0x00007ffff7fde660  →  <_dl_audit_preinit+0000> endbr64
$r12   : 0x00007fffffffde08  →  0x00007fffffffe098  →  "/home/user/csce_413/class_26/tmp/ransom"
$r13   : 0x0000555555400ae8  →  <main+0000> push rbp
$r14   : 0x0
$r15   : 0x00007ffff7ffd040  →  0x00007ffff7ffe2e0  →  0x0000555555400000  →   jg 0x555555400047
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00

0x00007fffffffd4b0│+0x0000: 0x00007fffffffdcf0  →  0x0000000000000001    ← $rsp, $rbp
0x00007fffffffd4b8│+0x0008: 0x0000555555400b13  →  <main+002b> lea rdi, [rip+0x24a]        # 0x555555400d64
0x00007fffffffd4c0│+0x0010: 0x0000000000000000
0x00007fffffffd4c8│+0x0018: 0x00007fff00000000
0x00007fffffffd4d0│+0x0020: 0x0000000000000000
0x00007fffffffd4d8│+0x0028: 0x00007fff00000000
0x00007fffffffd4e0│+0x0030: 0x00000000ffffffff
0x00007fffffffd4e8│+0x0038: 0x0000000000000000

   0x555555400a1e <get_key+0004>   mov    edi, 0x0
   0x555555400a23 <get_key+0009>   call   0x555555400890 <time@plt>
   0x555555400a28 <get_key+000e>   mov    edi, eax
 → 0x555555400a2a <get_key+0010>   call   0x555555400880 <srand@plt>
   ↳  0x555555400880 <srand@plt+0000> jmp    QWORD PTR [rip+0x201712]        # 0x555555601f98 <srand@got.plt>
      0x555555400886 <srand@plt+0006> push   0x6
      0x55555540088b <srand@plt+000b> jmp    0x555555400810
      0x555555400890 <time@plt+0000>  jmp    QWORD PTR [rip+0x20170a]        # 0x555555601fa0 <time@got.plt>
      0x555555400896 <time@plt+0006>  push   0x7
      0x55555540089b <time@plt+000b>  jmp    0x555555400810

srand@plt (
   $rdi = 0x0000000067e207d0,
   $rsi = 0x00007fffffffde08 → 0x00007fffffffe098 → "/home/user/csce_413/class_26/tmp/ransom",
   $rdx = 0x00007ffff7fc0080 → 0x00007ffff7fc0080,
   $rcx = 0x0000555555400ce0 → <__libc_csu_init+0000> push r15
)

[#0] Id 1, Name: "ransom", stopped 0x555555400a2a in get_key (), reason: SINGLE STEP

[#0] 0x555555400a2a → get_key()
[#1] 0x555555400b13 → main()

gef➤  set $rdi = 0x67e20670
```

Following, the program can terminate, revealing that the `jerma985.txt` file has been restored to its former state.



```
❯ cat jerma985.txt
This is art right here, I'm doing it right now. This is performance art! You guys don't understand? This is a living painting you're seeing!
Michelangelo, Leonardo, Da Vinci, they're all dead. I REMAIN! You understand? I REMAIN, as a performance artist!
```