Software Security via Program Analysis

Project 2: Emulating Partial Program

Yu-Sheng Wang (bqk3dj)

Example 4

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py
==========================================
1000000: eb 2c                  jmp 0x100002e
100002e: e8 cf ff ff ff         call 0x1000002
        | Memory (WR): addr 11ffffc, size: 4, value: 1000033
        | ESP: 011ffffc
1000002: 5e                     pop esi
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | ESI: 01000033
        | ESP: 01200000
1000003: 31 c0                  xor eax, eax
1000005: b0 17                  mov al, 0x17
        | EAX: 00000017
1000007: 50                     push eax
        | Memory (WR): addr 11ffffc, size: 4, value: 17
        | ESP: 011ffffc
1000008: cd 80                  int 0x80
[SYSCALL(INT 0x80)] SYS_SETUID, EAX: 17
```

```
    elif eax == 0x17:   # sys_setuid
        output = ("[SYSCALL(INT 0x80)] SYS_SETUID, EAX: %x") % (eax)
        out(output)
```

The code first run the system call 0x17 (setuid).

```
[SYSCALL(INT 0x80)] SYS_SETUID, EAX: 17
100000a: 31 c0                  xor eax, eax
        | EAX: 00000000
100000c: 50                     push eax
        | Memory (WR): addr 11ffff8, size: 4, value: 0
        | ESP: 011ffff8
100000d: 68 6e 2f 73 68         push 0x68732f6e
        | Memory (WR): addr 11ffff4, size: 4, value: 68732f6e
        | ESP: 011ffff4
1000012: 68 2f 2f 62 69         push 0x69622f2f
        | Memory (WR): addr 11ffff0, size: 4, value: 69622f2f
        | ESP: 011ffff0
1000017: 89 e3                  mov ebx, esp
        | EBX: 011ffff0
1000019: 50                     push eax
        | Memory (WR): addr 11fffec, size: 4, value: 0
        | ESP: 011fffec
100001a: 66 68 2d 63            push 0x632d
        | Memory (WR): addr 11fffea, size: 2, value: 632d
        | ESP: 011fffea
100001e: 89 e7                  mov edi, esp
        | EDI: 011fffea
1000020: 50                     push eax
        | Memory (WR): addr 11fffe6, size: 4, value: 0
        | ESP: 011fffe6
1000021: 56                     push esi
        | Memory (WR): addr 11fffe2, size: 4, value: 1000033
        | ESP: 011fffe2
1000022: 57                     push edi
        | Memory (WR): addr 11fffde, size: 4, value: 11fffea
        | ESP: 011fffde
1000023: 53                     push ebx
        | Memory (WR): addr 11fffda, size: 4, value: 11ffff0
        | ESP: 011fffda
1000024: 89 e7                  mov edi, esp
        | EDI: 011fffda
1000026: 50                     push eax
        | Memory (WR): addr 11fffd6, size: 4, value: 0
        | ESP: 011fffd6
```

Next, there are multiple pushed where the addresses and arguments of the instructions are recorded.

```
000024: 89 e7               mov edi, esp
       | EDI: 011fffda
000026: 50                  push eax
       | Memory (WR): addr 11fffd6, size: 4, value: 0
       | ESP: 011fffd6
000027: 57                  push edi
       | Memory (WR): addr 11fffd2, size: 4, value: 11fffda
       | ESP: 011fffd2
000028: 53                  push ebx
       | Memory (WR): addr 11fffce, size: 4, value: 11ffff0
       | ESP: 011fffce
000029: 50                  push eax
       | Memory (WR): addr 11fffca, size: 4, value: 0
       | ESP: 011fffca
00002a: b0 0b               mov al, 0xb
       | EAX: 0000000b
00002c: cd 80               int 0x80
SYSCALL(INT 0x80)] SYS_EXECV (Path: '//bin/sh', Arg: '/sbin/kldload /tmp/o.o')
```

Next, it runs the system call 0x11 (sys_execv) with path “//bin/sh” and argument “/sbin/kload /tmp/o.o”.

```
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000036: 69 6e 2f 6b 6c 64 6cimul ebp, dword ptr [esi + 0x2f], 0x6c646c6b
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
100003d: 6f                 outsd dx, dword ptr [esi]
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
100003e: 61                 popal
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
100003f: 64 20 2f           and byte ptr fs:[edi], ch
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000042: 74 6d              je 0x10000b1
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000044: 70 2f              jo 0x1000075
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000046: 6f                 outsd dx, dword ptr [esi]
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000047: 2e 6f              outsd dx, dword ptr cs:[esi]
This is not an instruction
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
no remaining -- done
```

The rest of the code is string and we can skip executing them with the following code:

```
231         i = 0
232         for c in filename:
233             addr = i + ebx
234             out("String Byte (filename): (%x)" % (addr))
235             i = i + 1
236             known_string.add(addr)
237         ## Collect address (Args)
238         i = 0
239         for c in args:
240             addr = i + esi
241             out("String Byte (args): (%x)" % (addr))
242             i = i + 1
243             known_string.add(addr)
```

Example 5

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py
=========================================
1000000: 31 d2              xor edx, edx
1000002: eb 0e              jmp 0x1000012
1000012: e8 ed ff ff ff     call 0x1000004
        | Memory (WR): addr 11ffffc, size: 4, value: 1000017
        | ESP: 011ffffc
1000004: 31 db              xor ebx, ebx
1000006: 5b                 pop ebx
        | Memory (RD): addr 11ffffc, size: 4, value: 0
        | EBX: 01000017
        | ESP: 01200000
1000007: b1 19              mov cl, 0x19
        | ECX: 00000019
1000009: 83 2c 1a 01        sub dword ptr [edx + ebx], 1
        | Memory (RD): addr 1000017, size: 4, value: 0
        | Memory (WR): addr 1000017, size: 4, value: 6951c131
100000d: 42                 inc edx
        | EDX: 00000001
100000e: e2 f9              loop 0x1000009
        | ECX: 00000018
1000009: 83 2c 1a 01        sub dword ptr [edx + ebx], 1
Already covered (stop analysis):: addr 1000009 (repeated: 1)
```

```
102     elif address in inst_executed_local:
103         if address in cnt_repeated:
104             cnt_repeated[address] = cnt_repeated[address] + 1
105         else:
106             cnt_repeated[address] = 1
107         output = "Already covered (stop analysis):: addr %x (repeated
108         out(output)
109         uc.emu_stop()
110         return
```

The program stops when encountering repeated instructions (loop).

```
102     elif address in inst_executed_local:
103         if address in cnt_repeated:
104             cnt_repeated[address] = cnt_repeated[address] + 1
105         else:
106             cnt_repeated[address] = 1
107 
108         if cnt_repeated[address] == 100:
109             output = "Already covered (stop analysis):: addr %x (repeated: %d)" % (ad
110             out(output)
111             uc.emu_stop()
112         return
```

Let's allow the program to run repeated instructions up to 100 times with the count array *cnt_repeated*. By doing so, we can get the following result:

```
100001f: 68 2f 62 69 6e      push 0x6e69622f
        | Memory (WR): addr 11ffff4, size: 4, value: 6e69622f
        | ESP: 011ffff4
1000024: 31 db               xor ebx, ebx
        | EBX: 00000000
1000026: 89 e3               mov ebx, esp
        | EBX: 011ffff4
1000028: 50                  push eax
        | Memory (WR): addr 11ffff0, size: 4, value: 0
        | ESP: 011ffff0
1000029: 54                  push esp
        | Memory (WR): addr 11fffec, size: 4, value: 11ffff0
        | ESP: 011fffec
100002a: 53                  push ebx
        | Memory (WR): addr 11fffe8, size: 4, value: 11ffff4
        | ESP: 011fffe8
100002b: 50                  push eax
        | Memory (WR): addr 11fffe4, size: 4, value: 0
        | ESP: 011fffe4
100002c: b0 0b               mov al, 0xb
        | EAX: 0000000b
100002e: cd 80               int 0x80
[SYSCALL(INT 0x80)] SYS_EXECV (Path: '/bin//sh', Arg: '')
```

The program executes *sys_execv* with path */bin/sh.* However, it fails when we run continue because we lost the previous state memory

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue
==========================================
1000020: 30 63 6a            xor byte ptr [ebx + 0x6a], ah
        | Memory (RD): addr 6a, size: 1, value: 0
        | Memory (WR): addr 6a, size: 1, value: 0
1000023: 6f                  outsd dx, dword ptr [esi]
        | Memory (RD): addr 0, size: 4, value: 0
        | ESI: 00000004
1000024: 32 dc               xor bl, ah
1000026: 8a e4               mov ah, ah
1000028: 51                  push ecx
        | Memory (WR): addr 11ffffc, size: 4, value: 0
        | ESP: 011ffffc
1000029: 55                  push ebp
        | Memory (WR): addr 11ffff8, size: 4, value: 0
        | ESP: 011ffff8
100002a: 54                  push esp
        | Memory (WR): addr 11ffff4, size: 4, value: 11ffff8
        | ESP: 011ffff4
100002b: 51                  push ecx
        | Memory (WR): addr 11ffff0, size: 4, value: 0
        | ESP: 011ffff0
100002c: b1 0c               mov cl, 0xc
        | ECX: 0000000c
100002e: ce                  into
100002f: 81 00 00 00 00 00   add dword ptr [eax], 0
        | Memory (RD): addr 0, size: 4, value: 0
        | Memory (WR): addr 0, size: 4, value: 0
1000035: 00 00               add byte ptr [eax], al
Out of range: addr 1000035, addr range 1000000 -- 1000030
```

To solve this issue, we write the memory & register state when the execution stops and read them back in the next execution:

```
47  def save_mem_reg_state(uc):
48      filename = "memdump.mem1.bin"
49      # mem1 = read memory of ADDRESS (for 3MB)
50      mem1 = uc.mem_read(ADDRESS, 3 * 1024 * 1024)
51      with open(filename, 'wb') as f:
52          f.write(mem1)
53          pass
54      filename = "memdump.mem2.bin"
55      # mem2 = read memory of 0 (for 3MB)
56      mem2 = uc.mem_read(0, 3 * 1024 * 1024)
57      with open(filename, 'wb') as f:
58          f.write(mem2)
59          pass
60
61      filename = "regdump.txt"
62      with open(filename, 'w') as f:
63          # save register values
64          f.write(str(uc.reg_read(UC_X86_REG_EAX))); f.write('\n')
65          f.write(str(uc.reg_read(UC_X86_REG_EBX))); f.write('\n')
66          f.write(str(uc.reg_read(UC_X86_REG_EDX))); f.write('\n')
67          f.write(str(uc.reg_read(UC_X86_REG_ESP))); f.write('\n')
68          pass
```

```
70  def load_mem_reg_state(uc):
71      filename = "memdump.mem1.bin"
72      with open(filename, 'rb') as f:
73          mem1 = f.read()
74          uc.mem_write(ADDRESS, mem1)
75
76      filename = "memdump.mem2.bin"
77      with open(filename, 'rb') as f:
78          mem2 = f.read()
79          uc.mem_write(0, mem2)
80
81
82      filename = "regdump.txt"
83      with open(filename, 'r') as f:
84          lines = f.readlines()
85          i = 0
86          for ln in lines:
87              if i == 0:  # eax
88                  uc.reg_write(UC_X86_REG_EAX, int(ln))
89              elif i == 1:    # ebx
90                  uc.reg_write(UC_X86_REG_EBX, int(ln))
91              elif i == 2:    # edx
92                  uc.reg_write(UC_X86_REG_EDX, int(ln))
93              elif i == 3:    # esp
94                  uc.reg_write(UC_X86_REG_ESP, int(ln))
95              i = i + 1
```

Finally, the program runs *sys_execv* twice*:*

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue loadstate
========================================
        | EAX: 0000000b
        | EBX: 011ffff4
        | EDX: 00000019
        | ESP: 011fffe4
1000020: 2f                 das
        | EAX: 00000005
1000021: 62 69 6e           bound ebp, qword ptr [ecx + 0x6e]
        | Memory (RD): addr 6e, size: 4, value: 0
        | Memory (RD): addr 72, size: 4, value: 0
1000024: 31 db              xor ebx, ebx
        | EBX: 00000000
1000026: 89 e3              mov ebx, esp
        | EBX: 011fffe4
1000028: 50                 push eax
        | Memory (WR): addr 11fffe0, size: 4, value: 5
        | ESP: 011fffe0
1000029: 54                 push esp
        | Memory (WR): addr 11fffdc, size: 4, value: 11fffe0
        | ESP: 011fffdc
100002a: 53                 push ebx
        | Memory (WR): addr 11fffd8, size: 4, value: 11fffe4
        | ESP: 011fffd8
100002b: 50                 push eax
        | Memory (WR): addr 11fffd4, size: 4, value: 5
        | ESP: 011fffd4
100002c: b0 0b              mov al, 0xb
        | EAX: 0000000b
100002e: cd 80              int 0x80
[SYSCALL(INT 0x80)] SYS_EXECV (Path: '', Arg: '')
```

```
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue loadstate
========================================
        | EAX: 0000000b
        | EBX: 011fffe4
        | EDX: 00000019
        | ESP: 011fffd4
1000023: 6e                 outsb dx, byte ptr [esi]
        | Memory (RD): addr 0, size: 1, value: 0
        | ESI: 00000001
1000024: 31 db              xor ebx, ebx
        | EBX: 00000000
1000026: 89 e3              mov ebx, esp
        | EBX: 011fffd4
1000028: 50                 push eax
        | Memory (WR): addr 11fffd0, size: 4, value: b
        | ESP: 011fffd0
1000029: 54                 push esp
        | Memory (WR): addr 11fffcc, size: 4, value: 11fffd0
        | ESP: 011fffcc
100002a: 53                 push ebx
        | Memory (WR): addr 11fffc8, size: 4, value: 11fffd4
        | ESP: 011fffc8
100002b: 50                 push eax
        | Memory (WR): addr 11fffc4, size: 4, value: b
        | ESP: 011fffc4
100002c: b0 0b              mov al, 0xb
100002e: cd 80              int 0x80
[SYSCALL(INT 0x80)] SYS_EXECV (Path: '', Arg: '')
String Byte (filename): (11fffd4)
forensics@forensics-VirtualBox:~/unicorn/bindings/python$ python3 emul.py continue loadstate
```