Fun with symbolic execution

Exploit development and deobfuscation

Carl Svensson

September 13, 2018

SEC-T 2018

About me

- Carl Svensson, 27
- MSc in Computer Science, KTH
- Head of Security, Kry
- CTF-player, HackingForSoju
- calle.svensson@zeta-two.com
- @zetatwo
- https://zeta-two.com

KRY

Symbolic execution

- Symbols vs. concrete values
- Pro: Explore ”all” paths
- Con: Exponential complexity

Once again, with fee... angr

- ”python framework for analyzing binaries”
- ”both static and dynamic symbolic (concolic)”
- Computer Security Lab at UC Santa Barbara
- Uses Z3 internally

Exploitation

- IP control
- Satisfy condition

Exploitation with angr

- Find execution path
- Constrain execution
- Satisfy condition

Example from Security Fest CTF

- Function pointer lookup
- Index OOB
- Hook messy function

angr exploitation example

```
1  void __fastcall __noreturn main(__int64 argc, char **argv, char **envp)
2  {
3    void (__fastcall *func_ptr)(); // rdx
4    int choice; // [rsp+0h] [rbp-10h]
5
6    setvbuf(stdin, 0LL, 2, 0LL);
7    setvbuf(stdout, 0LL, 2, 0LL);
8    alarm(0x3Cu);
9    print_welcome();
10
11   while ( 1 )
12   {
13     choice = get_choice();
14     if ( choice == -1 )
15     {
16       printf("\x1B[31;1merror:\x1B[0m not a number: %s\n", nptr);
17     }
18     else
19     {
20       memset(nptr, 0, endptr - nptr);
21       func_ptr = func_table[abs(choice) % 7];
22       ++endptr;
23       func_ptr();
24     }
25     print_menu();
26   }
27 }
```

angr exploitation example

```python
#!/usr/bin/env python
import angr

BASE_ADDR = 0x400000
def pie_addr(addr):
    return BASE_ADDR + addr

def hook_nop(state):
    state.regs.rax = 0

# Setup project and patch call
proj = angr.Project('dist/bowrain_581bbadaafd23051a25ccb4adc80b670', load_options={'auto_load_libs': False})
proj.hook(pie_addr(0xFAD), hook_nop, length = 5) # The memset call does nothing of importance and messes up angr (why?)

```

angr exploitation example

```
0000000000000F46 ; ---------------------------------------------------------------------------
0000000000000F46
0000000000000F46 loc_F46:                                ; CODE XREF: main+8C↑j
0000000000000F46                 mov     eax, [rbp+choice]
0000000000000F49                 sar     eax, 1Fh
0000000000000F4C                 mov     ecx, eax
0000000000000F4E                 xor     ecx, [rbp+choice]
0000000000000F51                 sub     ecx, eax
0000000000000F53                 mov     edx, 92492493h
0000000000000F58                 mov     eax, ecx
0000000000000F5A                 imul    edx
0000000000000F5C                 lea     eax, [rdx+rcx]
0000000000000F5F                 sar     eax, 2
0000000000000F62                 mov     edx, eax
0000000000000F64                 mov     eax, ecx
0000000000000F66                 sar     eax, 1Fh
0000000000000F69                 sub     edx, eax
0000000000000F6B                 mov     eax, edx
0000000000000F6D                 mov     [rbp+var_C], eax
0000000000000F70                 mov     edx, [rbp+var_C]
0000000000000F73                 mov     eax, edx
0000000000000F75                 shl     eax, 3
0000000000000F78                 sub     eax, edx
0000000000000F7A                 sub     ecx, eax
0000000000000F7C                 mov     eax, ecx
0000000000000F7E                 mov     [rbp+var_C], eax
0000000000000F81                 lea     rax, endptr
0000000000000F88                 mov     rax, [rax]
0000000000000F8B                 mov     rdx, rax
0000000000000F8E                 lea     rax, nptr        ; "0"
0000000000000F95                 sub     rdx, rax
0000000000000F98                 mov     rax, rdx
0000000000000F9B                 mov     rdx, rax         ; n
0000000000000F9E                 mov     esi, 0           ; c
0000000000000FA3                 lea     rax, nptr        ; "0"
0000000000000FAA                 mov     rdi, rax         ; s
0000000000000FAD                 call    memset
0000000000000FB2                 lea     rax, func_table
0000000000000FB9                 mov     edx, [rbp+var_C]
0000000000000FBC                 movsxd  rdx, edx
0000000000000FBF                 mov     rdx, (func_table - 2030A0h)[rax+rdx*8]
0000000000000FC3                 lea     rax, endptr
0000000000000FCA                 mov     rax, [rax]
0000000000000FCD                 lea     rcx, [rax+1]
0000000000000FD1                 lea     rax, endptr
0000000000000FD8                 mov     [rax], rcx
0000000000000FDB                 lea     rax, endptr
0000000000000FE2                 mov     rax, [rax]
0000000000000FE5                 mov     rdi, rax
0000000000000FE8                 mov     eax, 0
0000000000000FED                 call    rdx
```

angr exploitation example

```
14
15  # Create state and setup symbolic variable
16  state = proj.factory.blank_state(addr=pie_addr(0x000F46))
17  ADDR_CHOICE = state.regs.rbp - 0x10
18  state.mem[ADDR_CHOICE:].dword = state.solver.BVS('choice', 32)
19
20  # Find funcptr lookup instruction
21  sm = proj.factory.simgr(state)
22  sm.explore(find=pie_addr(0xFBF))
23
24  # Pick out the state and add constraint
25  find_state = sm.found[0].state
26  find_state.solver.add(find_state.solver.Or(find_state.regs.rdx < 0, find_state.regs.rdx > 7))
27
28  # Extract and display results
29  choice = find_state.regs.rbp - 0x10
30  print('Choice: %0d' % find_state.solver.eval(find_state.mem[choice:].int64_t.resolved, cast_to=int))
31  print('RDX: %08x' % find_state.solver.eval(find_state.regs.rdx, cast_to=int))
32
```

angr exploitation example

```
> python exploit_angr.py
Choice: 2147483648
RDX: fffffffffffffffe

> ./bowrain_581bbadaafd23051a25ccb4adc80b670
...
: 2147483648
[1] 17059 segmentation fault (core dumped)
```

Deobfuscation

Obfuscation

- Make code hard to read
  - for humans
  - for computers
- Control flow flattening
- Packer
- Dropper
- VM
- Dead code

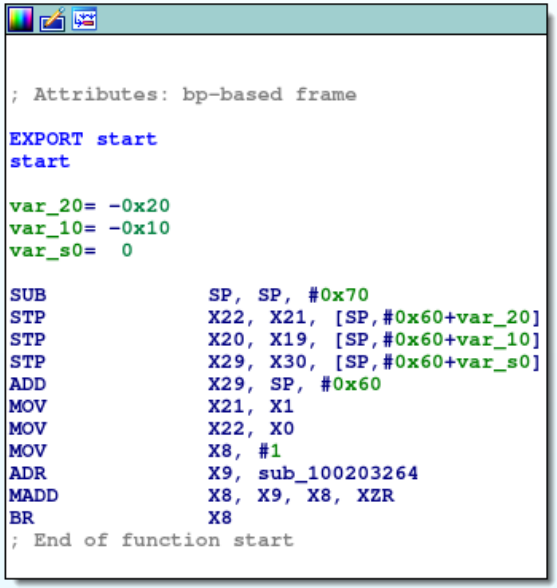
Deobfuscation in general

- Undo the mess
- Hard problem

Deobfuscation of dead code with angr

- Prove that dead code is dead
- Prove uniqueness of value

Example: indirect jmp deobfuscator



```
; Attributes: bp-based frame

EXPORT start
start

var_20= -0x20
var_10= -0x10
var_s0=  0

SUB             SP, SP, #0x70
STP             X22, X21, [SP,#0x60+var_20]
STP             X20, X19, [SP,#0x60+var_10]
STP             X29, X30, [SP,#0x60+var_s0]
ADD             X29, SP, #0x60
MOV             X21, X1
MOV             X22, X0
MOV             X8, #1
ADR             X9, sub_100203264
MADD            X8, X9, X8, XZR
BR              X8
; End of function start
```

Example from mobile app

- Find ”jmp reg”
- Search callgraph backwards
- Search forward
- Simplify expression
- Replace code

Example: indirect jmp deobfuscator

```
19  def try_get_reg_value(proj, node, addr, reg):
20      state = proj.factory.blank_state(addr=node.addr)
21      simgr = proj.factory.simgr(state)
22
23      # Find call location and eval target
24      simgr.explore(find=addr)
25      if len(simgr.found) == 0:
26          print("Unconstrained")
27          return False
28      s = simgr.found[0]
29      target_addr = s.solver.eval_upto(getattr(s.regs, reg), 10)
30      if len(target_addr) <= 1:
31          print('Jump addr: %016x' % target_addr[0])
32          return target_addr[0]
33      else:
34          print('Non-unique addr: %016x' % target_addr[0])
35          return False
36
```

Example: indirect jmp deobfuscator

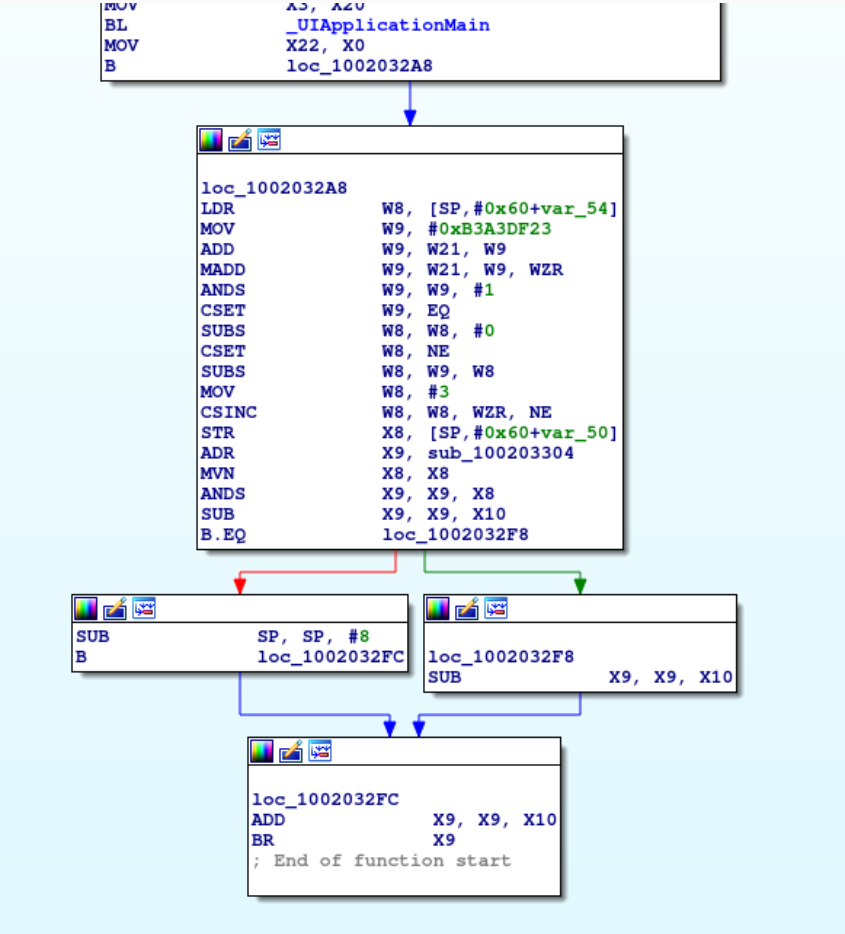
```
61  def get_reg_value(proj, cfg, addr):
62      current_function = get_target_function(cfg, addr)
63      current_node = cfg.get_any_node(addr, anyaddr=True)
64
65      reg = get_block_call_operand(current_node.block)
66      if not reg:
67          print('ERROR: Does not end with br')
68
69      while True:
70          target_addr = try_get_reg_value(proj, current_node, addr, reg)
71          if target_addr:
72              return reg, target_addr
73          current_node = bfs_back_to_function(current_node, current_function)
74          if not current_node:
75              return reg, False
76
```

Example: indirect jmp deobfuscator

```
; Attributes: bp-based frame

EXPORT start
start

var_54= -0x54
var_50= -0x50
var_20= -0x20
var_10= -0x10
var_s0=  0

SUB             SP, SP, #0x70
STP             X22, X21, [SP,#0x60+var_20]
STP             X20, X19, [SP,#0x60+var_10]
STP             X29, X30, [SP,#0x60+var_s0]
ADD             X29, SP, #0x60
MOV             X21, X1
MOV             X22, X0
MOV             X8, #1
ADR             X9, loc_100203264
MADD            X8, X9, X8, XZR
B               loc_100203264
```

```
loc_100203264
BL              _objc_autoreleasePoolPush
MOV             X19, X0
ADRP            X8, #classRef_____42@PAGE
LDR             X0, [X8,#classRef_____42@PAGEOFF] ; id
ADRP            X8, #selRef_class@PAGE
LDR             X1, [X8,#selRef_class@PAGEOFF] ; SEL
BL              _objc_msgSend
BL              _NSStringFromClass
BL              _objc_retainAutoreleasedReturnValue
MOV             X20, X0
MOV             X0, X22
MOV             X1, X21
MOV             X2, #0
MOV             X3, X20
BL              _UIApplicationMain
MOV             X22, X0
```

Example: indirect jmp deobfuscator



Thanks for listening!