Using SMT solvers for binary analysis and exploitation

A primer on SMT, SMT solvers, Z3 & angr

Carl Svensson

August 29, 2018

Nixucon 2018

About me

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

Reverse engineering in 15 seconds?

- Take stuff, e.g. software, apart
- Understand how it works
- Many possible goals
  - How can I reach a specific state?

What is SMT?

- Satisfiability modulo theories, SMT
- A bunch of variables
- A bunch of theories
  - Theory = A bunch of rules
- A bunch of formulas
- Can we find values for all values s.t. all formulas are satisifed?

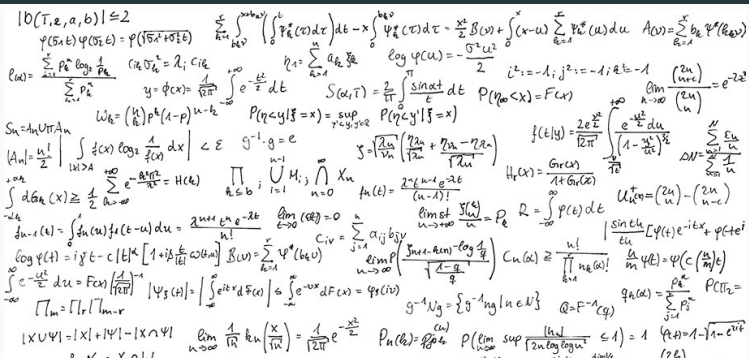
SMT: Example 1

$$x + 13 = 37$$

SMT: Example 2

$$x + y + 13 = 37 - z$$
$$x - 2 \cdot y + 10 = 10 \cdot z$$
$$4 \cdot x - z + 13 = 37 + y$$

SMT: Example 3



Microsoft to the rescue

- Can we automate? Yes!
- Microsoft Research
- Z3 Theorem Prover
  - General purpose
  - Own language
  - Bindings for several languages
  - Open source & cross platform

Using Z3 in Python

```python
from z3 import *

x = Real('x')
y = Real('y')

s = Solver()
s.add(x + y > 5)
s.add(x > 1, y > 1)

print(s.check())
print(s.model())

"""
> python z3_example.py
sat
[y = 3/2, x = 4]
"""
```

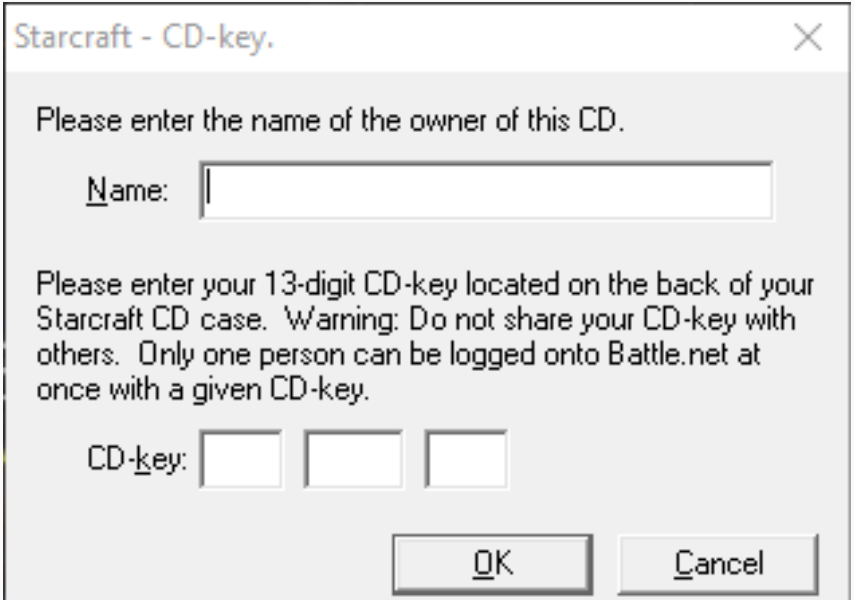
Throwback Thursday: Starcraft

Throwback Thursday: Starcraft

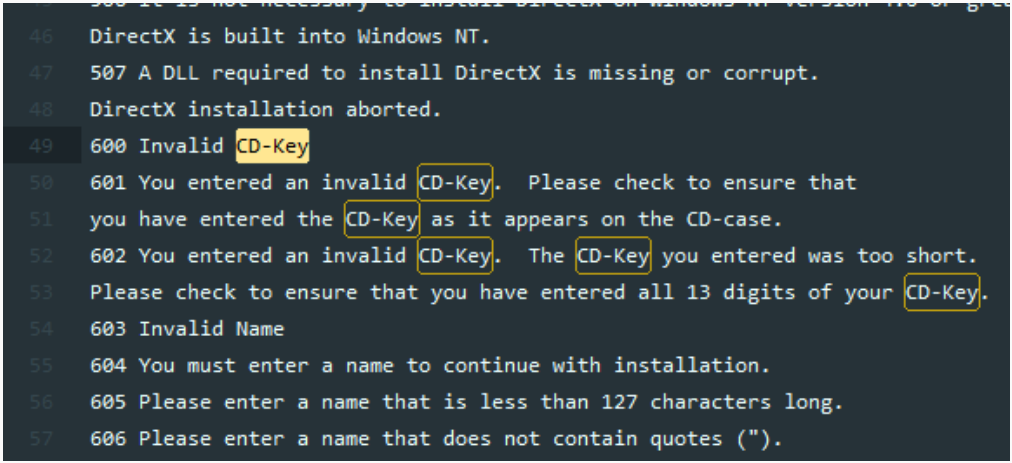
- Commercial software
- Released in 1998
  - Simple protections
  - Good starting point
- Requires a serial key
- Can we create our own?

Getting to the core: Installer

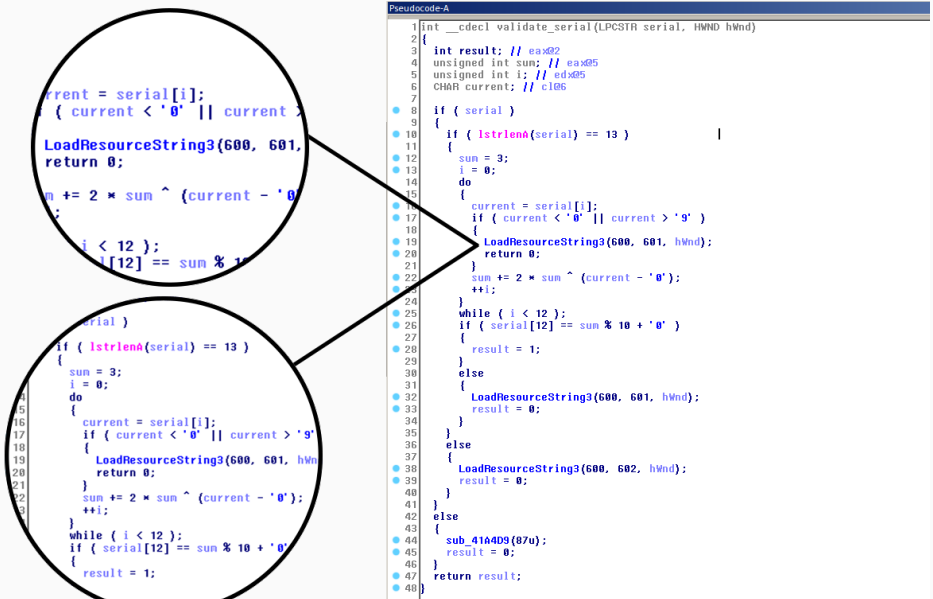
Getting to the core: Serial key input



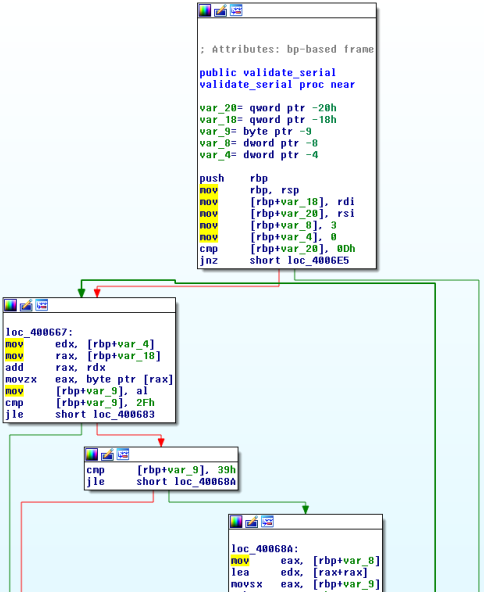
Getting to the core: Resource strings



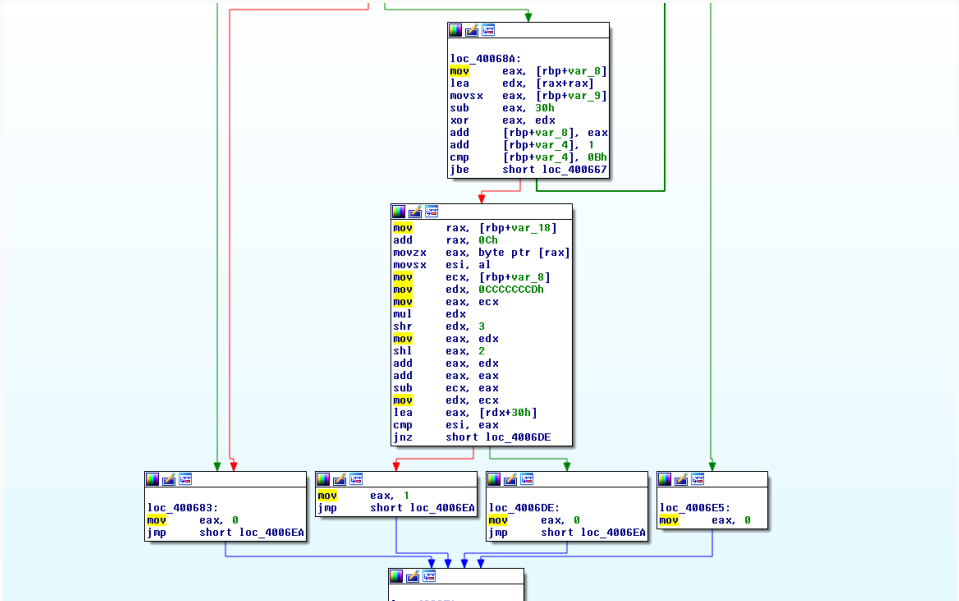
Getting to the core: Decompilation



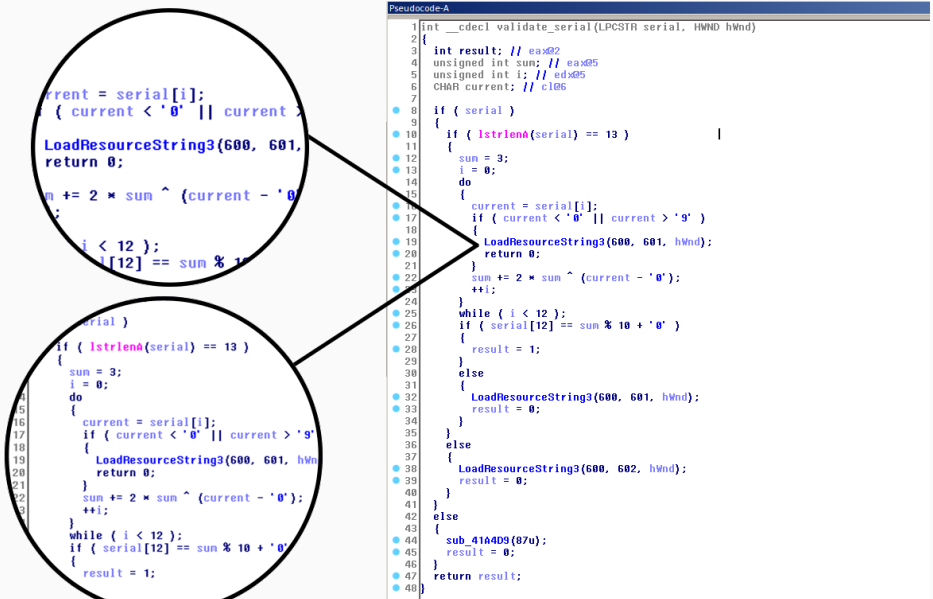
Getting to the core: Call graph



Getting to the core: Call graph



Getting to the core: Decompilation



Z3: Formulating formulas

solve.py

```
from z3 import *

s = Solver()

# Serial is 13 digits
serial = [BitVec('c%d' % i, 32) for i in range(13)]
for c in serial:
    s.add(c >= 0)
    s.add(c < 10)

# Partial sum
partials = [3]
for i in range(len(serial)-1):
    p = BitVec('p%d' % i, 32)
    s.add(p == partials[-1] + ((2*partials[-1]) ^ (serial[i])))
    partials.append(p)
```

Z3: Formulating formulas

```
17
18  # Final check
19  s.add(serial[-1] == (partials[-1] % 10))
20
21  # Print model
22  if s.check() == sat:
23      m = s.model()
24      res = map(lambda s: m[s].as_long(), serial)
25      res = map(lambda n: chr(n+ord('0')), res)
26      print(''.join(res))
27
```

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

Angr management: Extracting the code

validate.c

```
int __cdecl validate_serial(LPCSTR serial, HWND hWnd)
{
  int result; // eax@2
  unsigned int v3; // eax@5
  unsigned int v4; // edx@5
  CHAR v5; // cl@6

  if ( serial )
  {
    if ( lstrlenA(serial) == 13 )
    {
      v3 = 3;
      v4 = 0;
      do
      {
        v5 = serial[v4];
        if ( v5 < '0' || v5 > '9' )
        {
          LoadResourceString3(600, 601, hWnd);
          return 0;
        }
        v3 += 2 * v3 ^ (v5 - '0');
        ++v4;
      }
      while ( v4 < 12 );
      if ( serial[12] == v3 % 10 + '0' )
      {
        result = 1;
      }
      else
      {
        LoadResourceString3(600, 601, hWnd);
        result = 0;
      }
    }
    else
    {
      LoadResourceString3(600, 602, hWnd);
      result = 0;
    }
  }
  else
  {
    sub_41A4D9(0x57u);
    result = 0;
  }
  return result;
}
```

Angr management: Minimizing the code

validate_clean.c

```c
#include <stdio.h>
#include <string.h>

int validate_serial(char *serial, size_t len)
{
  int result;
  unsigned int sum = 3;
  unsigned int i = 0;
  char current;

  if ( len == 13 )
  {
    do
    {
      current = serial[i];
      if ( current < '0' || current > '9' )
      {
        return 0;
      }
      sum += 2 * sum ^ (current - '0');
      ++i;
    }
    while ( i < 12 );
    if ( serial[12] == sum % 10 + '0' )
    {
      return 1;
    }
    else
    {
      return 0;
    }
  }
  else
  {
    return 0;
  }
}

int main(int argc, char **argv) {
  char serial[1024];
  scanf("%s", serial);
  printf("Serial: %s\nValid: %d\n", serial, validate_serial(serial, strlen(serial)));

  return 0;
}
```

Angr management: Writing the explorer

solve_angr.py

```
#!/usr/bin/python

import angr

def main():
    p = angr.Project('./validator2', load_options={"auto_load_libs": False})
    pg = p.factory.path_group()

    pg.explore(find=(0x4006d7,), avoid=(0x400683,0x4006de,0x4006e5,))

    found = pg.found[0]
    return found.state.posix.dumps(0).split('\0')[0]

if __name__ == '__main__':
    print(main())
```

Can we use even less effort?

- Extracting code is cumbersome
- Can't we use the code in place?
- "Call" directly into validator
- Symbolic argument
- Patch away irrelevant parts

Full fury: Writing the explorer

```
int __cdecl validate_serial(LPCSTR lpString, HWND hWnd)
{
  int result; // eax
  unsigned int sum; // eax
  unsigned int i; // edx
  CHAR cur; // cl

  if ( lpString )
  {
    if ( lstrlenA(lpString) == 13 )
    {
      sum = 3;
      i = 0;
      do
      {
        cur = lpString[i];
        if ( cur < 48 || cur > 57 )
        {
          LoadResourceString(600, 601, hWnd);
          return 0;
        }
        sum += (cur - 48) ^ 2 * sum;
        ++i;
      }
      while ( i < 0xC );
      if ( lpString[12] == sum % 0xA + 48 )
      {
        result = 1;
      }
      else
      {
        LoadResourceString(600, 601, hWnd);
        result = 0;
      }
    }
    else
    {
      LoadResourceString(600, 602, hWnd);
      result = 0;
    }
  }
  else
  {
    SetError(0x57u);
    result = 0;
  }
  return result;
}
```

Full fury: Writing the explorer

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

"""
> file INSTALL.EXE
INSTALL.EXE: PE32 executable (GUI) Intel 80386, for MS Windows
> sha256sum INSTALL.EXE
ba155a30ca11a57a2ea917cb4f25715f79bee7397ebb16db4816e7725395e58d  INSTALL.EXE
"""

ADDR_VALIDATE = 0x0040F8A0
ADDR_VALIDATE_OK = 0x0040F93F
ADDR_VALIDATE_BAD = [
    0x0040F8B5,
    0x0040F8DD,
    0x0040F937,
    0x0040F95B,
]

ADDR_RESOURCE_STRING = 0x00402DD0
ADDR_SET_ERROR = 0x0041696A

# This takes a long time
loader = angr.cle.Loader("INSTALL.EXE", auto_load_libs=False, use_system_libs=False)
print('Binary loaded')
```

Full fury: Writing the explorer

```
28  proj = angr.Project(loader)
29
30  # Skip irrelevant functions
31  stub_func = angr.SIM_PROCEDURES['stubs']['ReturnUnconstrained']
32  proj.hook(ADDR_RESOURCE_STRING, stub_func())
33  proj.hook(ADDR_SET_ERROR, stub_func())
34
35  # Setup the symbolic input
36  serial_size = 32
37  sym_serial = claripy.BVS("sym_serial", serial_size * 8)
38
39  # HWND pointer can be NULL as all uses are hooked
40  state = proj.factory.call_state(ADDR_VALIDATE, sym_serial, 0)
41  simgr = proj.factory.simulation_manager(state)
42  simgr.explore(find=ADDR_VALIDATE_OK, avoid=ADDR_VALIDATE_BAD)
43
44  found = simgr.found[0] # A state that reached the find condition from explore
45  val_serial = found.solver.eval(sym_serial, cast_to=str) # Return a concrete string value for the sym arg to reach this state
46  val_serial = val_serial.strip('\x00') # Cleanup
47
48  print('Serial key: %s' % val_serial)
49
```

What about exploitation?

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

```
000000000000F466 ; ---------------------------------------------------------------------------
000000000000F466
000000000000F466 loc_F46:                                ; CODE XREF: main+8C↑j
000000000000F466                mov     eax, [rbp+choice]
000000000000F469                sar     eax, 1Fh
000000000000F46C                mov     ecx, eax
000000000000F46E                xor     ecx, [rbp+choice]
000000000000F471                sub     ecx, eax
000000000000F473                mov     edx, 92492493h
000000000000F478                mov     eax, ecx
000000000000F47A                imul    edx
000000000000F47C                lea     eax, [rdx+rcx]
000000000000F47F                sar     eax, 2
000000000000F482                mov     edx, eax
000000000000F484                mov     eax, ecx
000000000000F486                sar     eax, 1Fh
000000000000F489                sub     edx, eax
000000000000F48B                mov     eax, edx
000000000000F48D                mov     [rbp+var_C], eax
000000000000F490                mov     edx, [rbp+var_C]
000000000000F493                mov     eax, edx
000000000000F495                shl     eax, 3
000000000000F498                sub     eax, edx
000000000000F49A                sub     ecx, eax
000000000000F49C                mov     eax, ecx
000000000000F49E                mov     [rbp+var_C], eax
000000000000F4A1                lea     rax, endptr
000000000000F4A8                mov     rax, [rax]
000000000000F4AB                mov     rdx, rax
000000000000F4AE                lea     rax, nptr        ; "0"
000000000000F4B5                sub     rdx, rax
000000000000F4B8                mov     rax, rdx
000000000000F4BB                mov     rdx, rax         ; n
000000000000F4BE                mov     esi, 0           ; c
000000000000F4C3                lea     rax, nptr        ; "0"
000000000000F4CA                mov     rdi, rax         ; s
000000000000F4CD                call    memset
000000000000F4D2                lea     rax, func_table
000000000000F4D9                mov     edx, [rbp+var_C]
000000000000F4DC                movsxd  rdx, edx
000000000000F4DF                mov     rdx, (func_table - 2030A0h)[rax+rdx*8]
000000000000F4E3                lea     rax, endptr
000000000000F4EA                mov     rax, [rax]
000000000000F4ED                lea     rcx, [rax+1]
000000000000F4F1                lea     rax, endptr
000000000000F4F8                mov     [rax], rcx
000000000000F4FB                lea     rax, endptr
000000000000F502                mov     rax, [rax]
000000000000F505                mov     rdi, rax
000000000000F508                mov     eax, 0
000000000000F50D                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)
```

Even deobfuscation?!

Obfuscation

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

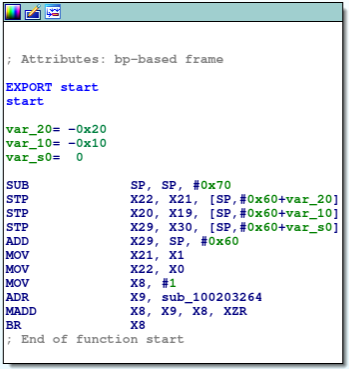
Deobfuscation in general

- 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

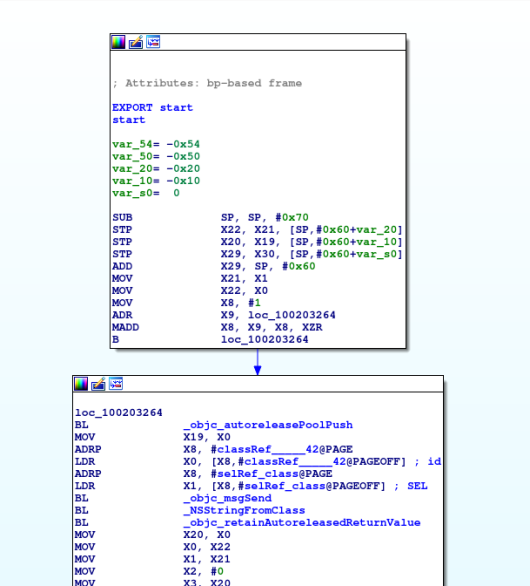
Example: indirect jmp deobfuscator

```
def try_get_reg_value(proj, node, addr, reg):
    state = proj.factory.blank_state(addr=node.addr)
    simgr = proj.factory.simgr(state)

    # Find call location and eval target
    simgr.explore(find=addr)
    if len(simgr.found) == 0:
        print("Unconstrained")
        return False
    s = simgr.found[0]
    target_addr = s.solver.eval_upto(getattr(s.regs, reg), 10)
    if len(target_addr) <= 1:
        print('Jump addr: %016x' % target_addr[0])
        return target_addr[0]
    else:
        print('Non-unique addr: %016x' % target_addr[0])
        return False

```

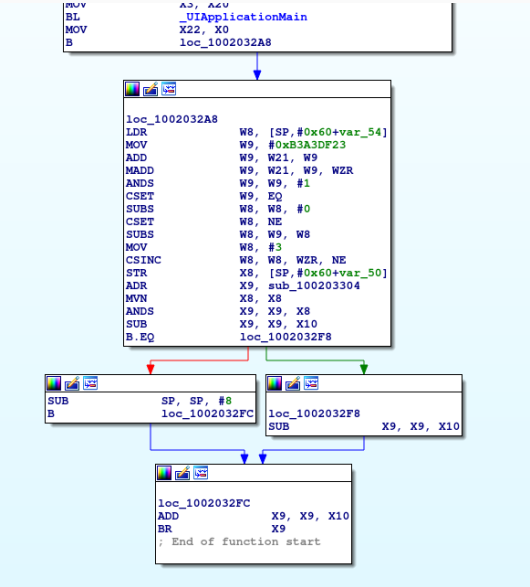
Example: indirect jmp deobfuscator

```
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



Example: indirect jmp deobfuscator



Thanks for listening!