Reverse Engineering

Symbolic Execution, Tools

2023/5/21, Sharkkcode

About

Sharkkcode
CCU IM → NTHU IS
CCU ISC
Reverse, PWN
https://github.com/Sharkkcode
https://sharkkcode.github.io/

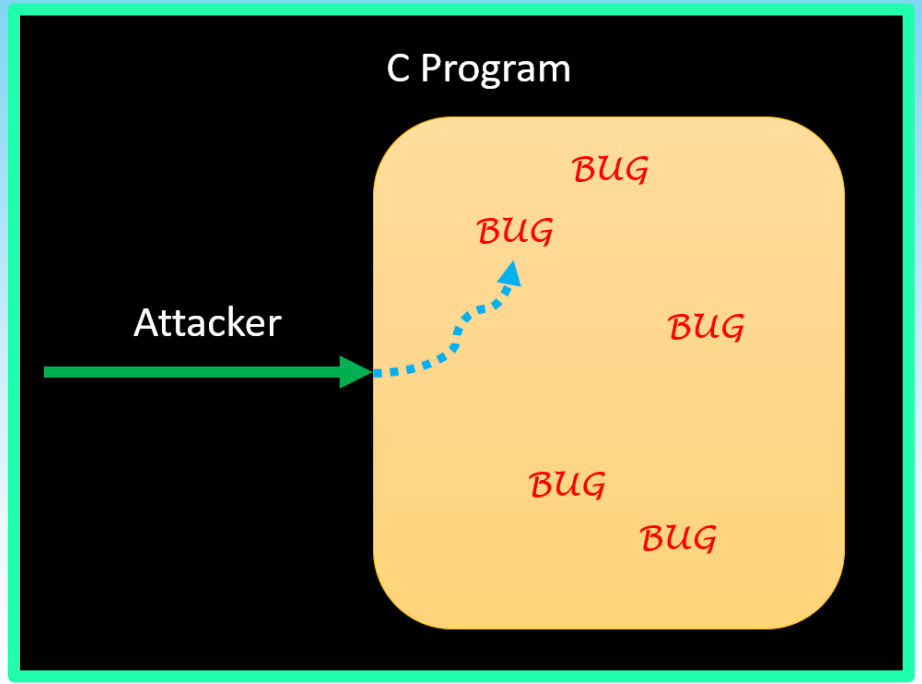
Table of Contents

01

Symbolic Execution

Symbolic Execution

- Symbolic Execution is useful for detecting ***bugs*** and ***vulnerabilities*** in programs, as it can explore all possible execution paths and identify inputs that cause the program to behave unexpectedly or violate security policies.



Symbolic Execution

- Symbolic Execution is a program analysis technique that uses ***symbolic values*** instead of concrete inputs to systematically explore program execution paths.

Examples :

Symbolic Values → x

Concrete Values → 100

Symbolic Execution

- The symbolic values are manipulated based on the program's operations and conditions, generating a set of constraints that must be satisfied for each path.

Symbolic Execution

Example:

```
1. read x, y
2. if x > y:
3.   x = y
4. if x < y:
5.   x = x + 1
6. if x + y == 7
7.   error()
```

Symbolic Execution

Example:

```
1. read x, y
2. if x > y:
3.   x = y
4. if x < y:
5.   x = x + 1
6. if x + y == 7
7.   error()
```

```
assert(x + y != 7)
```

Symbolic Execution

Example:

```
1. read x, y
2. if x > y:
3.   x = y
4. if x < y:
5.   x = x + 1
6. assert(x + y != 7)
```

Symbolic Execution

Example:

```
1. read x, y
2. if x > y:
3.   x = y
4. if x < y:
5.   x = x + 1
6. assert(x + y != 7)
```

Example Path 1 :

Line	x (a)	y (b)	Path Condition {}
1.	a	b	{}
2t.	a	b	{a > b}
3.	b	b	{a > b}
4.	No Fork	No Fork	No Fork
6.	No Fork	No Fork	No Fork

Symbolic Execution

Example:

```
1. read x, y
2. if x > y:
3.   x = y
4. if x < y:
5.   x = x + 1
6. assert(x + y != 7)
```

Example Path 2 :

Line	x (a)	y (b)	Path Condition {}
1.	a	b	{}
2f.	a	b	{a <= b}
4t.	a	b	{a < b}
5.	a + 1	b	{a < b}
6.	a + 1	b	{a + 1 + b == 7}

One solution : a == 2 && b == 4

Symbolic Execution

- Symbolic Execution can also be used to generate test cases that achieve high code coverage, as it can explore paths that are difficult to reach with random or manual testing.

Symbolic Execution

- However, Symbolic Execution can be computationally expensive, as it requires solving complex constraint systems.
  - EX: Path explosion…

- To mitigate this, various techniques have been developed, such as ***constraint-solving optimizations***, ***path pruning***, and ***concolic execution*** (which combines concrete and symbolic execution).

02

SAT/SMT

SAT/SMT -- SAT

- Boolean **SAT**isfiability Problem
- SAT is a decision problem in computer science that asks whether a given boolean formula can be satisfied by assigning boolean values to its variables.

SAT/SMT -- SAT

- SAT Example :
  - A = ?
  - B = ?
  - C = ?

$$\boxed{(A\ OR\ B)\ AND\ (NOT\ A\ OR\ C)\ AND\ (NOT\ B\ OR\ NOT\ C) = 1}$$

SAT/SMT -- SAT

- SAT Example (One solution) :
  - A = 0
  - B = 1
  - C = 0

$$\boxed{(A\ OR\ B)\ AND\ (NOT\ A\ OR\ C)\ AND\ (NOT\ B\ OR\ NOT\ C) = 1}$$

SAT/SMT -- SMT

- Satisfiability Modulo Theories
- SAT is focused on boolean formulas, while SMT extends SAT to formulas that involve variables from different theories.

SAT/SMT -- SMT

- SAT Example :
  - x = ?
  - y = ?
  - z = ?

$$\boxed{\begin{cases} 3x + 8y - z = 6 \\ -2x + 5y - 9z \leq 4 \\ -7x + 2y - 10z > 1 \end{cases}}$$

SAT/SMT -- SMT

- SAT Example (One Solution) :
  - x = -78
  - y = 37
  - z = 56

$$\begin{cases} 3 \times (-78) + 8 \times 37 - 56 = 6 \\ -2 \times (-78) + 5 \times 37 - 9 \times 56 = -163 \leq 4 \\ -7 \times (-78) + 2 \times 37 - 10 \times 56 = 60 > 1 \end{cases}$$

03

Tools

z3, angr

Tools -- z3

- Theorem prover developed by Microsoft Research for automatically solving logical formulas. It's widely used in automated reasoning, program analysis, verification, and security.

Tools -- z3

- z3 Example 1 :
  - A = ?
  - B = ?
  - C = ?

$$\boxed{(A\ OR\ B)\ AND\ (NOT\ A\ OR\ C)\ AND\ (NOT\ B\ OR\ NOT\ C) = 1}$$

Tools -- z3

- z3 Example 1 (Solution) :

```python
from z3 import *

A, B, C = Bool('A'), Bool('B'), Bool('C')

s = Solver()

cond1 = Or(A, B)
cond2 = Or(Not(A), C)
cond3 = Or(Not(B), Not(C))

s.add(And(And(cond1, cond2), cond3) == True)

print("check :", s.check())
print("model :", s.model())

```

Tools -- z3

- z3 Example 1 (Solution) :

```
check : sat
model : [A = False, B = True, C = False]
```

Verification:

```
>>>
>>> (not False or True) and (not False or False) and (not True or not False) == True
True
>>>
```

Tools -- z3

- z3 Example 2 :
  - x = ?
  - y = ?
  - z = ?

$$\boxed{\begin{cases} 3x + 8y - z = 6 \\ -2x + 5y - 9z \leq 4 \\ -7x + 2y - 10z > 1 \end{cases}}$$

Tools -- z3

- z3 Example 2 (Solution) :

```python
from z3 import *

x, y, z = Int('x'), Int('y'), Int('z')

s = Solver()

s.add(3 * x + 8 * y - 1 * z == 6)
s.add((-2) * x + 5 * y - 9 * z <= 4)
s.add((-7) * x + 2 * y - 10 * z > 1)

print("check :", s.check())
print("model :", s.model())
```

Tools -- z3

- z3 Example 2 (Solution) :

```
check : sat
model : [x = -78, y = 37, z = 56]
```

Verification:

```
>>>
>>> (3)*(-78)+(8)*(37)+(-1)*(56) == 6
True
>>> (-2)*(-78)+(5)*(37)+(-9)*(56) <= 4
True
>>> (-7)*(-78)+(2)*(37)+(-10)*(56) > 1
True
>>>
```

Tools -- angr

- Binary analysis framework developed by MIT for automating binary analysis tasks such as vulnerability discovery, exploit generation, and malware analysis. It provides a Python-based interface for analyzing binaries, and includes a wide range of analysis tools and techniques.

Tools -- angr

- angr Example 1 :
  - https://github.com/jakespringer/angr_ctf/tree/master/00_angr_find
  - Use seed `12345`

Tools -- angr

- angr Example 1 (Solution) -- Generate the executable file of `00_angr_find.c.jinja` with seed 12345 :
  - `python3 ./generate.py 12345 00_angr_find`

Tools -- angr

- angr Example 1 (Solution) -- `00_angr_find` overview :
  - `file ./00_angr_find && checksec ./00_angr_find`
  - Information :
    - ELF 32-bit LSB executable
    - dynamically linked
    - not stripped
    - Arch: i386-32-little
    - RELRO: Partial RELRO
    - Stack: Canary found
    - NX: NX enabled
    - PIE: No PIE (0x8048000)

Tools -- angr

- angr Example 1 (Solution) -- Reverse :

```
080492eb 83 c4 10      ADD        ESP,0x10
080492ee eb 10         JMP        LAB_08049300

                 LAB_080492f0                                   XRI
080492f0 83 ec 0c      SUB        ESP,0xc
080492f3 68 38 a0      PUSH       s_Good_Job._0804a038
         04 08
080492f8 e8 73 fd      CALL       <EXTERNAL>::puts
         ff ff
080492fd 83 c4 10      ADD        ESP,0x10

                 LAB_08049300                                   XRI
08049300 b8 00 00      MOV        EAX,0x0
         00 00
08049305 8b 4d f4      MOV        ECX,dword ptr [EBP + local_14]
08049308 65 33 0d      XOR        ECX,dword ptr GS:[0x14]
         14 00 00
         00
```

```
14  local_14 = *(int *)(in_GS_OFFSET + 0x14);
15  printf("Enter the password: ");
16  __isoc99_scanf(&DAT_0804a02b,local_1d);
17  for (local_24 = 0; local_24 < 8; local_24 = local_24 + 1) {
18    cVar1 = complex_function((int)local_1d[local_24],local_24);
19    local_1d[local_24] = cVar1;
20  }
21  iVar2 = strcmp(local_1d,"OVXRNKOD");
22  if (iVar2 == 0) {
23    puts("Good Job.");
24  }
25  else {
26    puts("Try again.");
27  }
28  if (local_14 != *(int *)(in_GS_OFFSET + 0x14)) {
29                    /* WARNING: Subroutine does not return */
30    __stack_chk_fail();
31  }
32  return 0;
```

Tools -- angr

- angr Example 1 (Solution) -- `get_flag.py` (1/3) :

```python
import sys

def argv_check(input_argv):
    if len(input_argv) != 2:
        print("Usage: python3 <script name> <file name>")
        print("Usage: ./<script name> <file name>")
        sys.exit()

# check argv
argv_check(sys.argv)

import angr

```

Tools -- angr

- angr Example 1 (Solution) -- `get_flag.py` (2/3) :

```python

def run_angr(file_name_str):

    p = angr.Project(file_name_str)

    # initial state
    init_state = p.factory.entry_state()

    # simulation execute
    sm = p.factory.simulation_manager(init_state)

    sm.explore(find=0x080492f8)

    found_count = len(sm.found)
    print("FOUND [ " + str(found_count) + " ] INPUT(s)")

```

Tools -- angr

- angr Example 1 (Solution) -- `get_flag.py` (3/3):

```
    print("<<<START>>>")
    if found_count > 0:
        found_counter = 1
        for found_state in sm.found:

            # print counter
            print(str(found_counter) + ". ", end="")

            # input to go to this state
            print(found_state.posix.dumps(0))

            found_counter = found_counter + 1

    print("<<<END>>>")

# run angr
run_angr(sys.argv[1])
```

Tools -- angr

- angr Example 1 (Solution) -- `get_flag.py` output :

```
FOUND [ 1 ] INPUT(s)
<<<START>>>
1. b'OSRIBVWI'
<<<END>>>
```

Tools -- angr

- angr Example 2 :
  - https://github.com/jakespringer/angr_ctf/tree/master/01_angr_avoid
  - Use seed `12345`

Tools -- angr

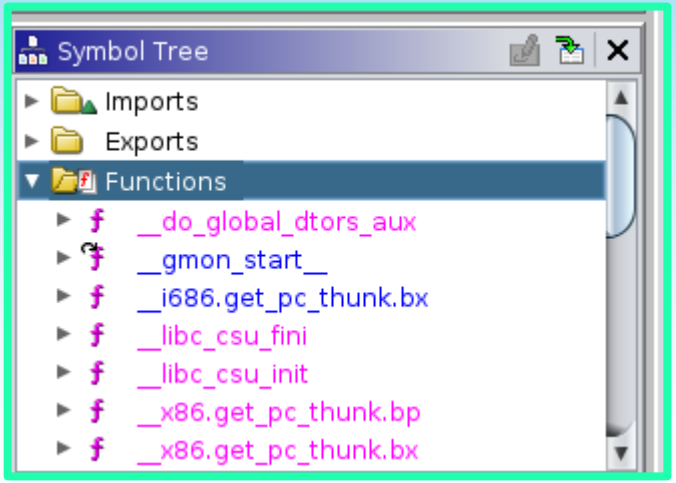
- angr Example 2 (Solution) -- Generate the executable file of `01_angr_avoid.c.jinja` with seed 12345 :
  - `python3 ./generate.py 12345 ./01_angr_avoid`

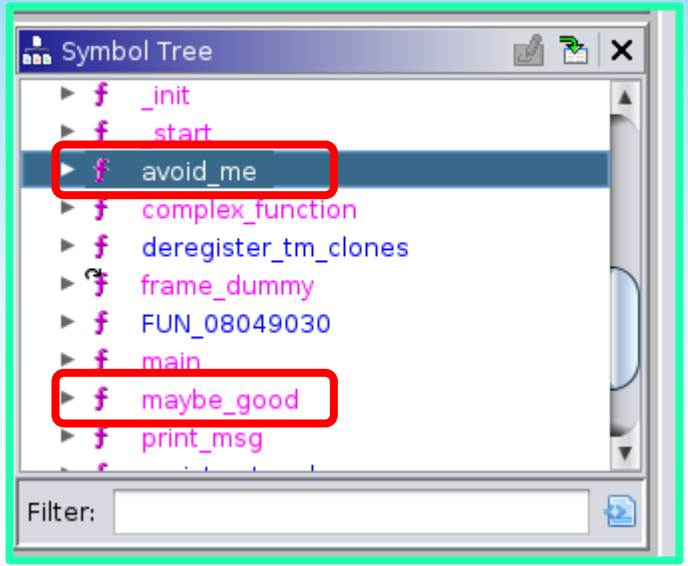
Tools -- angr

- angr Example 2 (Solution) -- `01_angr_avoid` overview :
  - `file ./01_angr_avoid && checksec ./01_angr_avoid`
  - Information :
    - ELF 32-bit LSB executable
    - dynamically linked
    - not stripped
    - Arch: i386-32-little
    - RELRO: Partial RELRO
    - Stack: Canary found
    - NX: NX enabled
    - PIE: No PIE (0x8048000)

Tools -- angr

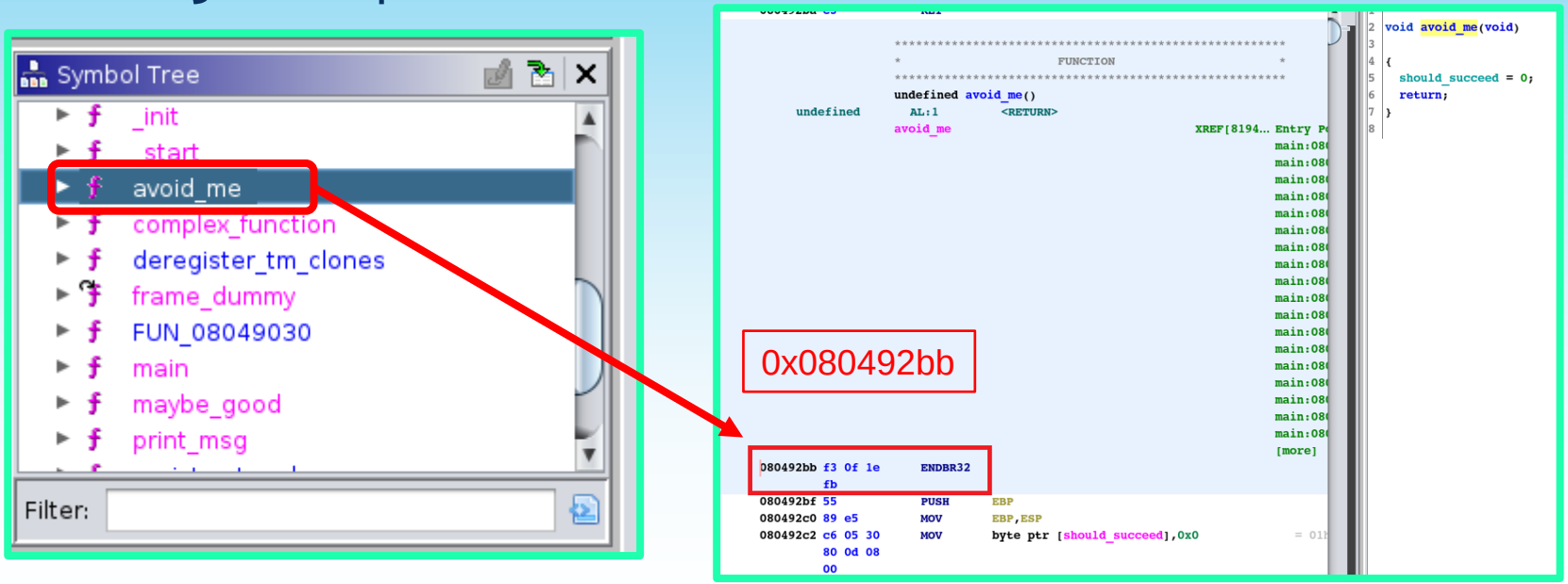
- angr Example 2 (Solution) -- Reverse (1/3) :





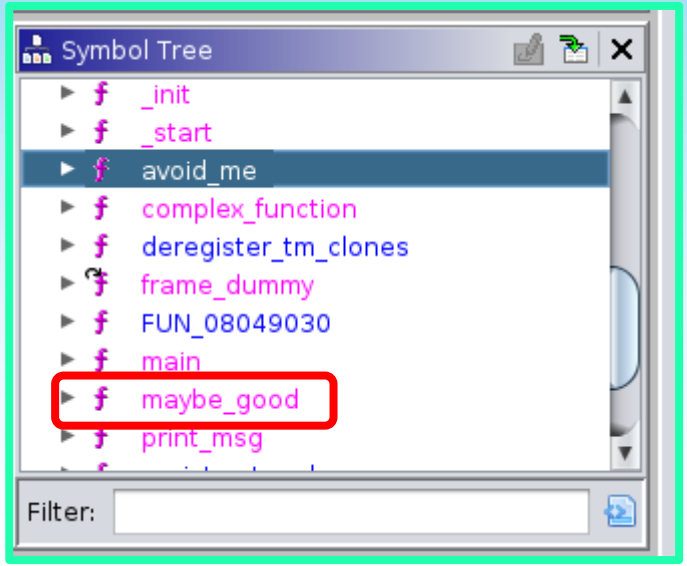
Tools -- angr

- angr Example 2 (Solution) -- Reverse (2/3) :

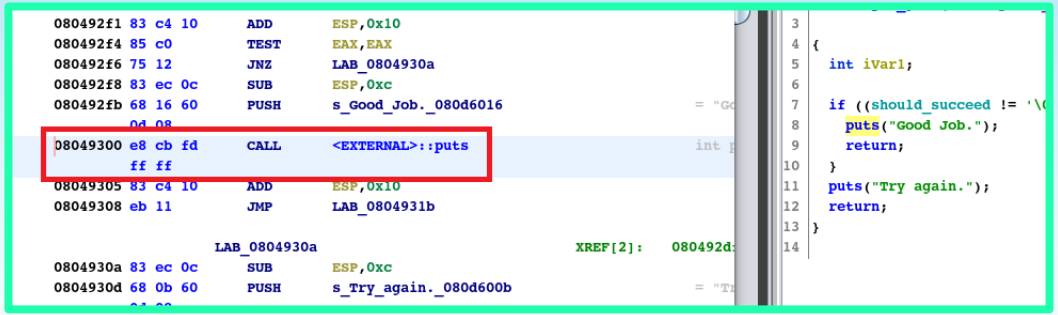


Tools -- angr

- angr Example 2 (Solution) -- Reverse (3/3) :



0x08049300 CALL puts → “Good Job.”



Tools -- angr

- angr Example 2 (Solution) -- `get_flag.py` (1/3):

```python
import sys

def argv_check(input_argv):
    if len(input_argv) != 2:
        print("Usage: python3 <script name> <file name>")
        print("Usage: ./<script name> <file name>")
        sys.exit()

# check argv
argv_check(sys.argv)

import angr

```

Tools -- angr

- angr Example 2 (Solution) -- `get_flag.py` (2/3) :

```python

def run_angr(file_name_str):

    p = angr.Project(file_name_str)

    # initial state
    init_state = p.factory.entry_state()

    # simulation execute
    sm = p.factory.simulation_manager(init_state)

    sm.explore(find=0x08049300, avoid=0x080492bb)

    found_count = len(sm.found)
    print("FOUND [ " + str(found_count) + " ] INPUT(s)")

```

Tools -- angr

- angr Example 2 (Solution) -- `get_flag.py` (3/3):

```python

    print("<<<START>>>")
    if found_count > 0:
        found_counter = 1
        for found_state in sm.found:

            # print counter
            print(str(found_counter) + ". ", end="")

            # input to go to this state
            print(found_state.posix.dumps(0))

            found_counter = found_counter + 1

    print("<<<END>>>")

# run angr
run_angr(sys.argv[1])

```

Tools -- angr

- angr Example 2 (Solution) -- `get_flag.py` output :

```
FOUND [ 1 ] INPUT(s)
<<<START>>>
1. b'MFPOIYZC'
<<<END>>>
```

Tools -- angr

- angr Example 3 :
  - https://github.com/jakespringer/angr_ctf/tree/master/02_angr_find_condition
  - Use seed `12345`

Tools -- angr

- angr Example 3 (Solution) -- Generate the executable file of `02_angr_find_condition.c.jinja` with seed 12345 :
  - `python3 ./generate.py 12345 02_angr_find_condition`

Tools -- angr

- angr Example 3 (Solution) -- `02_angr_find_condition` overview :
  - `file ./02_angr_find_condition && checksec ./02_angr_find_condition`
  - Information :
    - ELF 32-bit LSB executable
    - dynamically linked
    - not stripped
    - Arch: i386-32-little
    - RELRO: Partial RELRO
    - Stack: Canary found
    - NX: NX enabled
    - PIE: No PIE (0x8048000)

Tools -- angr

- angr Example 3 (Solution) -- Reverse :
  - Good Job.
  - Try again.

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` (1/5):

```python
import sys

def argv_check(input_argv):
    if len(input_argv) != 2:
        print("Usage: python3 <script name> <file name>")
        print("Usage: ./<script name> <file name>")
        sys.exit()

# check argv
argv_check(sys.argv)

import angr

```

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` (2/5):

```python

def run_angr(file_name_str):

    p = angr.Project(file_name_str)

    # initial state
    init_state = p.factory.entry_state()

    # simulation execute
    sm = p.factory.simulation_manager(init_state)

```

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` (3/5):

```python

def is_good(state):
    return True if b"Good Job." in state.posix.dumps(1) else False

def is_bad(state):
    return True if b"Try again." in state.posix.dumps(1) else False

```

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` (4/5):

```
29
30      sm.explore(find=is_good, avoid=is_bad)
31
32      found_count = len(sm.found)
33      print("FOUND [ " + str(found_count) + " ] INPUT(s)")
34
```

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` (5/5):

```python

    print("<<<START>>>")
    if found_count > 0:
        found_counter = 1
        for found_state in sm.found:

            # print counter
            print(str(found_counter) + ". ", end="")

            # input to go to this state
            print(found_state.posix.dumps(0))

            found_counter = found_counter + 1

    print("<<<END>>>")

# run angr
run_angr(sys.argv[1])

```

Tools -- angr

- angr Example 3 (Solution) -- `get_flag.py` output :

```
FOUND [ 1 ] INPUT(s)
<<<START>>>
1. b'YRBAUKLO'
<<<END>>>
```

Tools -- angr

- angr Example 4 :
  - https://github.com/jakespringer/angr_ctf/tree/master/03_angr_symbolic_registers
  - Use seed `12345`

Tools -- angr

- angr Example 4 (Solution) -- Generate the executable file of `03_angr_symbolic_registers.c.jinja` with seed 12345 :
  - `python3 ./generate.py 12345 03_angr_symbolic_registers`

Tools -- angr

- angr Example 4 (Solution) -- `03_angr_symbolic_registers` overview :
  - `file ./03_angr_symbolic_registers && checksec ./03_angr_symbolic_registers`
  - Information :
    - ELF 32-bit LSB executable
    - dynamically linked
    - not stripped
    - Arch: i386-32-little
    - RELRO: Partial RELRO
    - Stack: Canary found
    - NX: NX enabled
    - PIE: No PIE (0x8048000)

Tools -- angr

- angr Example 4 (Solution) -- Reverse (1/3) :

Decompile: get_user_input - (03_angr_symbolic_registers)

```
1
2  undefined8 get_user_input(void)
3
4  {
5    int in_GS_OFFSET;
6    undefined4 local_1c;
7    undefined local_18 [4];
8    undefined4 local_14;
9    int local_10;
10
11   local_10 = *(int *)(in_GS_OFFSET + 0x14);
12   __isoc99_scanf("%x %x %x",&local_1c,local_18,&local_14);
13   if (local_10 != *(int *)(in_GS_OFFSET + 0x14)) {
14                    /* WARNING: Subroutine does not return */
15     __stack_chk_fail();
16   }
17   return CONCAT44(local_14,local_1c);
18 }
19
```

Tools -- angr

- angr Example 4 (Solution) -- Reverse (2/3) :

Listing: 03_angr_symbolic_registers

```
080495c1 f3 0f 1e        ENDBR32
         fb
080495c5 55              PUSH       EBP
080495c6 89 e5           MOV        EBP,ESP
080495c8 83 ec 18        SUB        ESP,0x18
080495cb 65 a1 14        MOV        EAX,GS:[0x14]
         00 00 00
080495d1 89 45 f4        MOV        dword ptr [EBP + local_10],EAX
080495d4 31 c0           XOR        EAX,EAX
080495d6 8d 4d f0        LEA        ECX=>local_14,[EBP + -0x10]
080495d9 51              PUSH       ECX
080495da 8d 4d ec        LEA        ECX=>local_18,[EBP + -0x14]
080495dd 51              PUSH       ECX
080495de 8d 4d e8        LEA        ECX=>local_1c,[EBP + -0x18]
080495e1 51              PUSH       ECX
080495e2 68 0b a0        PUSH       s_%x_%x_%x_0804a00b                    = "%x %x %x"
         04 08
080495e7 e8 e4 fa        CALL       <EXTERNAL>::__isoc99_scanf             undefined __isoc99_scanf(
         ff ff
080495ec 83 c4 10        ADD        ESP,0x10
080495ef 8b 45 e8        MOV        EAX,dword ptr [EBP + local_1c]
080495f2 8b 55 ec        MOV        EDX,dword ptr [EBP + local_18]
080495f5 89 d3           MOV        EBX,EDX
080495f7 8b 55 f0        MOV        EDX,dword ptr [EBP + local_14]
080495fa 90              NOP
080495fb 8b 4d f4        MOV        ECX,dword ptr [EBP + local_10]
080495fe 65 33 0d        XOR        ECX,dword ptr GS:[0x14]
         14 00 00
         00
08049605 74 05           JZ         LAB_0804960c
08049607 e8 94 fa        CALL       <EXTERNAL>::__stack_chk_fail           undefined __stack_chk_fai
         ff ff
```

Decompile: get_user_input - (03_angr_symbolic_registers)

```
1
2  undefined8 get_user_input(void)
3
4  {
5    int in_GS_OFFSET;
6    undefined4 local_1c;
7    undefined local_18 [4];
8    undefined4 local_14;
9    int local_10;
10
11   local_10 = *(int *)(in_GS_OFFSET + 0x14);
12   __isoc99_scanf("%x %x %x",&local_1c,local_18,&local_14);
13   if (local_10 != *(int *)(in_GS_OFFSET + 0x14)) {
14                    /* WARNING: Subroutine does not return */
15     __stack_chk_fail();
16   }
17   return CONCAT44(local_14,local_1c);
18 }
19
```

get_user_input

Store user input in → EAX, EBX, EDX

Tools -- angr

- angr Example 4 (Solution) -- Reverse (3/3) :

0x08049638

main

EAX, EBX, EDX → Stack (argument of x86 32 bit function)

```
        ff ff
08049638 89 45 ec     MOV     dword ptr [EBP + local_1c],EAX
0804963b 89 5d f0     MOV     dword ptr [EBP + local_18],EBX
0804963e 89 55 f4     MOV     dword ptr [EBP + local_14],EDX
08049641 83 ec 0c     SUB     ESP,0xc
08049644 ff 75 ec     PUSH    dword ptr [EBP + local_1c]
08049647 e8 cc fb     CALL    complex_function_1          undefined complex_functio
        ff ff
0804964c 83 c4 10     ADD     ESP,0x10
0804964f 89 45 ec     MOV     dword ptr [EBP + local_1c],EAX
08049652 83 ec 0c     SUB     ESP,0xc
08049655 ff 75 f0     PUSH    dword ptr [EBP + local_18]
```

```
10
11  printf("Enter the password: ");
12  uVar4 = get_user_input();
13  iVar1 = complex_function_1((int)uVar4);
14  iVar2 = complex_function_2(unaff_EBX);
15  iVar3 = complex_function_3((int)((ulonglong)uVar4 >> 0x20));
16  if (((iVar1 == 0) && (iVar2 == 0)) && (iVar3 == 0)) {
17    puts("Good Job.");
18  }
19  else {
20    puts("Try again.");
21  }
```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` (1/5):

```python
import sys
import claripy

def argv_check(input_argv):
    if len(input_argv) != 2:
        print("Usage: python3 <script name> <file name>")
        print("Usage: ./<script name> <file name>")
        sys.exit()

# check argv
argv_check(sys.argv)

import angr

```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` (2/5):

```python

def run_angr(file_name_str):

    p = angr.Project(file_name_str)

    # initial state
    # blank state
    init_addr = 0x08049638
    init_state = p.factory.blank_state(addr=init_addr)

```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` (3/5) :

```python

    # create symbol vector
    # 1 byte = 8 bit
    register_len_byte = 4
    eax_vec = claripy.BVS("eax_vec", register_len_byte * 8)
    ebx_vec = claripy.BVS("ebx_vec", register_len_byte * 8)
    edx_vec = claripy.BVS("edx_vec", register_len_byte * 8)

    # set symbol vector to register
    init_state.regs.eax = eax_vec
    init_state.regs.ebx = ebx_vec
    init_state.regs.edx = edx_vec

```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` (4/5):

```python
# simulation execute
sm = p.factory.simulation_manager(init_state)

def is_good(state):
    return True if b"Good Job." in state.posix.dumps(1) else False

def is_bad(state):
    return True if b"Try again." in state.posix.dumps(1) else False

sm.explore(find=is_good, avoid=is_bad)

found_count = len(sm.found)
print("FOUND [ " + str(found_count) + " ] INPUT(s)")

```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` (5/5) :

```python

    print("<<<START>>>")
    if found_count > 0:
        found_counter = 1
        for found_state in sm.found:

            # print counter
            print(str(found_counter) + ". ", end="")

            # get value
            ans1 = str(hex(found_state.solver.eval(eax_vec))).replace("0x", "")
            ans2 = str(hex(found_state.solver.eval(ebx_vec))).replace("0x", "")
            ans3 = str(hex(found_state.solver.eval(edx_vec))).replace("0x", "")
            print(ans1 + " " + ans2 + " " + ans3)

            found_counter = found_counter + 1

    print("<<<END>>>")

# run angr
run_angr(sys.argv[1])
```

Tools -- angr

- angr Example 4 (Solution) -- `get_flag.py` output :

```
FOUND [ 1 ] INPUT(s)
<<<START>>>
1. a89f88f0 4290f2bc bb1e433e
<<<END>>>
```

THANKS!

Q & A