Reverse Engineering

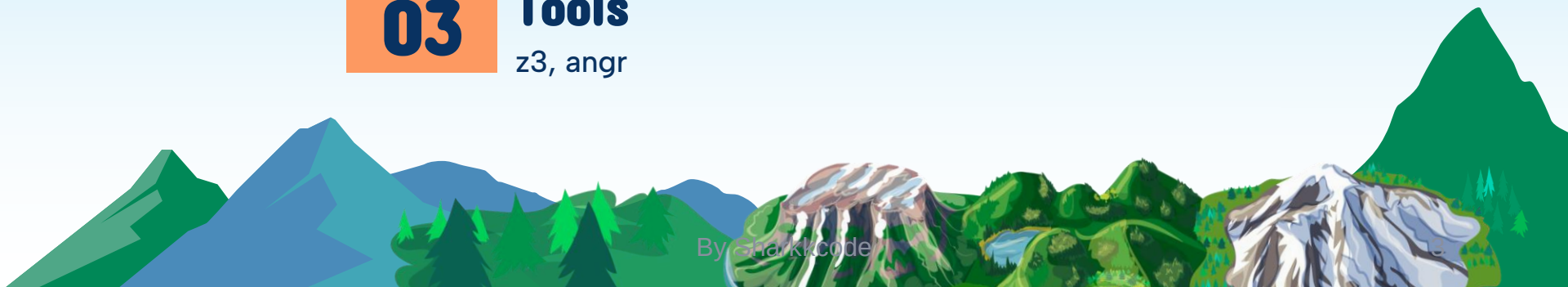
Symbolic Execution, Tools

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

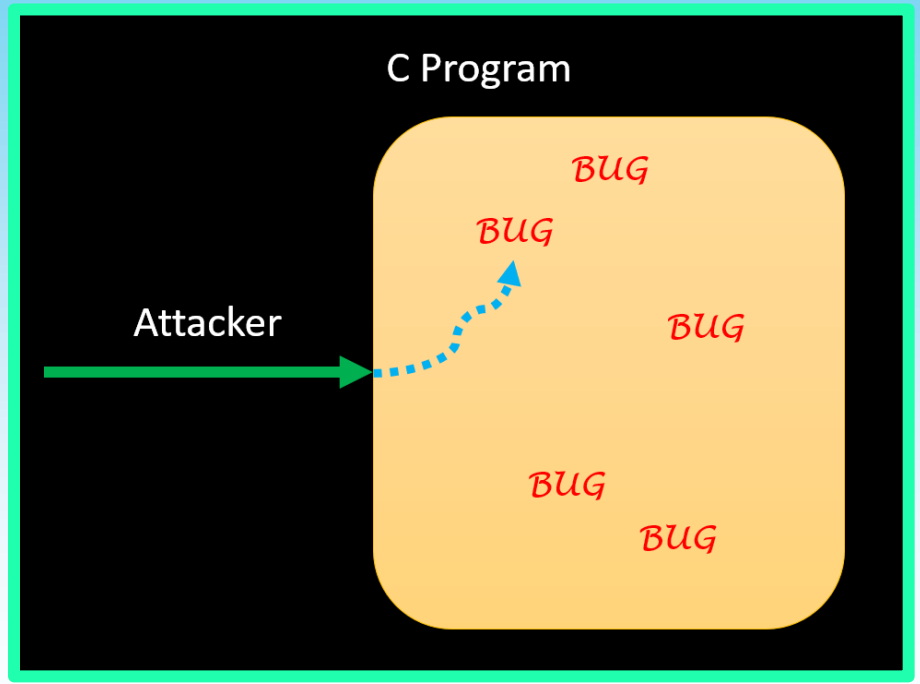


01

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 = ?

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

SAT/SMT -- SAT

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

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

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 = ?

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

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 = ?

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

Tools -- z3

- z3 Example 1 (Solution) :

```
1 from z3 import *
2
3 A, B, C = Bool('A'), Bool('B'), Bool('C')
4
5 s = Solver()
6
7 cond1 = Or(A, B)
8 cond2 = Or(Not(A), C)
9 cond3 = Or(Not(B), Not(C))
10
11 s.add(And(And(cond1, cond2), cond3) == True)
12
13 print("check :", s.check())
14 print("model :", s.model())
15
~
```

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 = ?

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

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

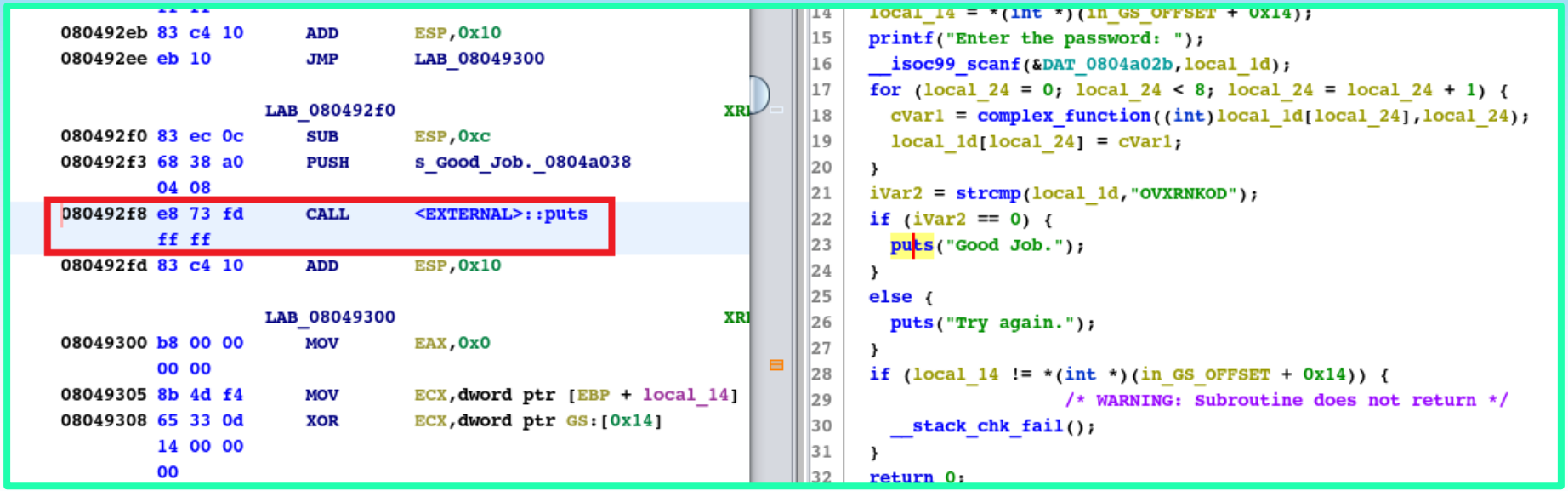
- angr Example 1 (Solution) -- Generate the executable file of `00_angr_find.c.jinja` with seed 12345 :
  - `python3 ./generate.py 12345 00_angr_find`

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):

```
13
14  def run_angr(file_name_str):
15
16      p = angr.Project(file_name_str)
17
18      # initial state
19      init_state = p.factory.entry_state()
20
21      # simulation execute
22      sm = p.factory.simulation_manager(init_state)
23
24      sm.explore(find=0x080492f8)
25
26      found_count = len(sm.found)
27      print("FOUND [ " + str(found_count) + " ] INPUT(s)")
28
```

Tools -- angr

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

```
28
29      print("<<<START>>>")
30      if found_count > 0:
31          found_counter = 1
32          for found_state in sm.found:
33
34              # print counter
35              print(str(found_counter) + ". ", end="")
36
37              # input to go to this state
38              print(found_state.posix.dumps(0))
39
40              found_counter = found_counter + 1
41
42      print("<<<END>>>")
43
44  # run angr
45  run_angr(sys.argv[1])
46
```

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

Tools -- angr

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

THANKS!

Q & A