

Intro to angr and concolic analysis



[Foo-Manroot](#)



<https://foo-manroot.github.io/>

Agenda

- Theory
 - What is symbolic execution?
 - Concrete + Symbolic = Concolic
 - angr: platform and use cases
- (BREAK !)
- Hands-on
 - Sample CTF binaries

What is symbolic execution?

What is symbolic execution?

Different approaches to analyze a program: **concrete values**

```
int whatever (int a)
{
    int b = 3;

    if (a < b)
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43)
    {
        a += 1234;
    }

    printf ("kalm");

    return a - b;
}
```

- Goal: find inputs so that `whatever ()` returns “1000” or above
- Approach 1: random (*concrete*) testing.
 - Execute the function with different values and check the output.
 - We can’t be 100% sure that we covered all paths.
 - We missed the `if (a == 43)` branch.

Provided value	Output	Return value
<code>whatever (0);</code>	“PANIK!”	<i>(fail)</i>
<code>whatever (12);</code>	“kalm”	9
<code>whatever (-1);</code>	“PANIK!”	<i>(fail)</i>

What is symbolic execution?

Different approaches to analyze a program: **symbolic values**

```
int whatever (int a)
{
    int b = 3;

    if (a < b)
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43)
    {
        a += 1234;
    }

    printf ("kalm");

    return a - b;
}
```

- Goal: find inputs so that `whatever ()` returns "1000" or above
- Approach 2: *symbolic* testing.
 1. Replace unknown values (`int a`) by an arbitrary *symbol* (α).
 2. Gather all constraints for each branch.
 3. Solve the equations to obtain possible input values to reach that specific branch.

(or, at least, that's the *very* simplified explanation)

What is symbolic execution?

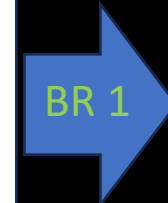
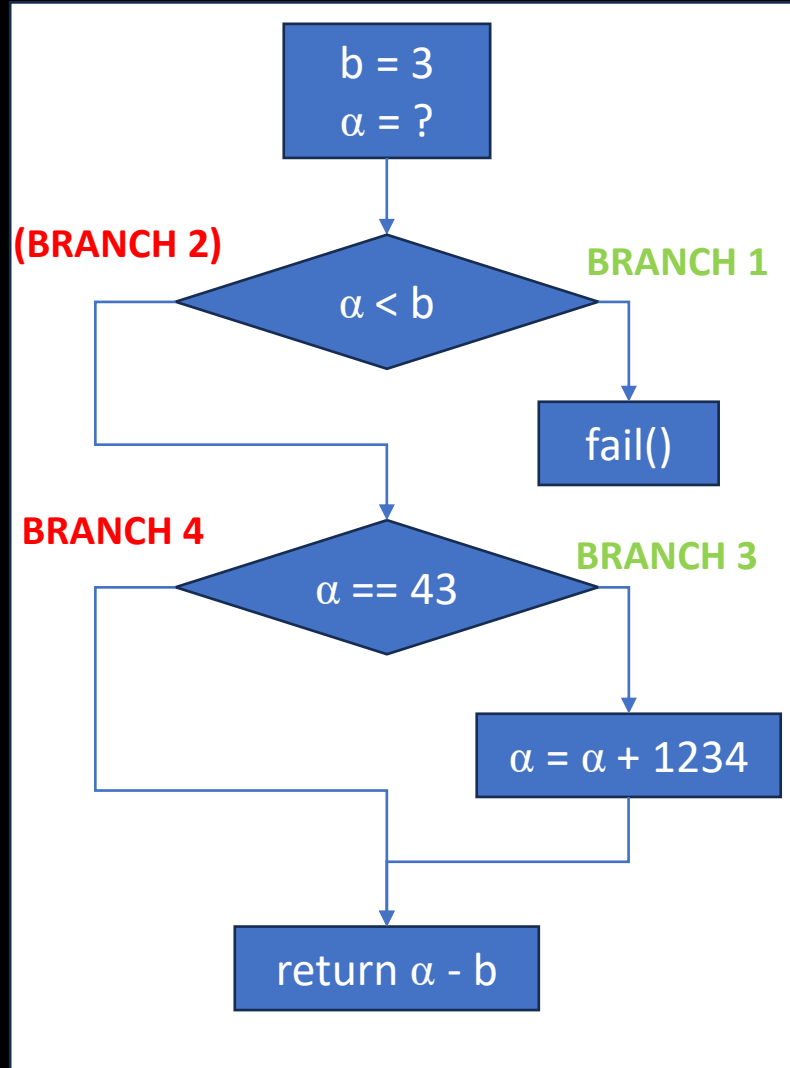
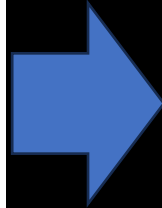
Get constraints for each branch

```
int whatever (int a)
{
    int b = 3;

    if (a < b)
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43)
    {
        a += 1234;
    }

    printf ("kalm");

    return a - b;
}
```



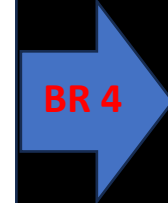
$\alpha < 3$



$\alpha \geq 3$

$\alpha == 43$

return = $\alpha + 1234 - 3$



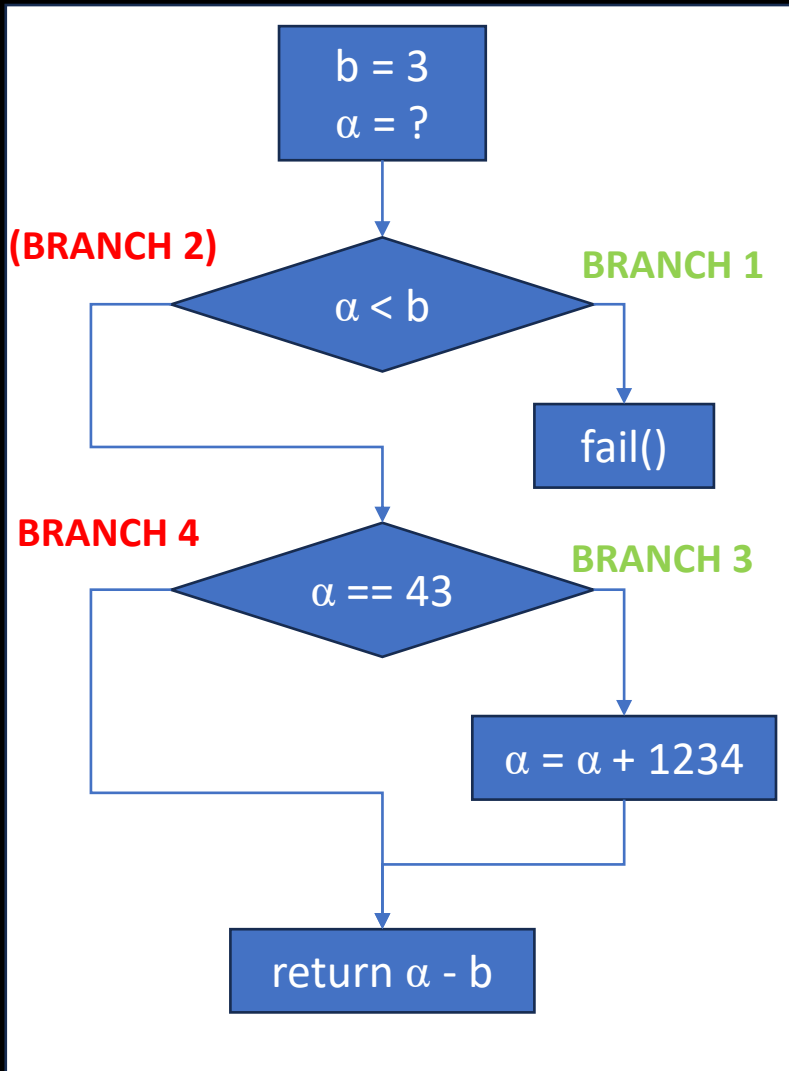
$\alpha > 3$

$\alpha \neq 43$

return = $\alpha - 3$

What is symbolic execution?

Solve the constraints (SMT solver)



- Example constraints: execute **Branch 3**

$$\begin{cases} \alpha \geq 3 \\ \alpha == 43 \end{cases}$$

- Solving this equations gives us a concretized value for α : **43**

- Execute **Branch 4**

$$\begin{cases} \alpha \geq 3 \\ \alpha \neq 43 \end{cases}$$

- Concrete value for α : **17**
- There are multiple solutions, but we only need one

What is symbolic execution?

Limitations

- Solving the equations (**SMT**) is an **NP-hard** problem.
 - Domain-specific reasoning can be combined with Boolean solvers (**SAT**) to speed-up execution. Some problems are **still insolvable** (e.g.: *hashing*)
- Taking all branches means an **explosion in possible states** to keep track of.
 - Search **can be optimised**:
 - Avoid certain paths (e.g.: “finish exploration when reaching `fail()`”)
 - Depth-First search
 - ...
- Modelling **environment**
 - Normal programs open **sockets**, read **files**, allocate **memory**, ...

What is symbolic execution?

Advantages

- **Full coverage** of all branches.
 - We're exploring all possible branches.
- **Platform-independent.**
 - An ELF can be analysed from a Windows machine.
- **Traceability.**
 - There's no difficulty reproducing the state: we know the constraints required to trigger it (e.g.: `random_func()` must return 1337 to crash).

Concrete + Symbolic = Concolic

Concolic execution

Overview

- Using concrete and symbolic execution to support each other
 - Concrete execution **avoids** the need to model the **environment** (API calls, file operations, ...)
 - **Symbolic** execution helps to **find concrete values** that will cover all paths
- Advantages:
 - **No more paths explosions** (drawback from symbolic exec.)
 - **Better code coverage** (drawback from concrete exec.)

Concolic execution

Practical example: branch 1

```
int whatever (int a) ←  
{  
    int b = 3;  
  
    if (a < b)  
    {  
        printf ("PANIK!");  
        fail ();  
    }  
    else if (a == 43)  
    {  
        a += 1234;  
    }  
  
    printf ("kalm");  
  
    return a - b;  
}
```

Concrete values	Symbolic values	Path conditions
a = 0	a = α	-

Concolic execution

Practical example: branch 1

```
int whatever (int a)
{
    int b = 3;

    if (a < b) ←
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43)
    {
        a += 1234;
    }

    printf ("kalm");

    return a - b;
}
```

Concrete values	Symbolic values	Path conditions
$a = 0$	$a = \alpha$	-
$a = 0$	$a = \alpha$	$\alpha < b$

Concolic execution

Practical example

- The concrete execution found a branch and reached its end: `fail()`
- Now, the concolic engine:
 1. Negates one of the path conditions:
 1. $\neg(\alpha < b) \rightarrow (\alpha \geq b)$
 2. Asks the SMT solver to find new concrete values
 3. Executes the function again, to follow the other branch

Concolic execution

Practical example: branch 2

```
int whatever (int a) ←  
{  
    int b = 3;  
  
    if (a < b)  
    {  
        printf ("PANIK!");  
        fail ();  
    }  
    else if (a == 43)  
    {  
        a += 1234;  
    }  
  
    printf ("kalm");  
  
    return a - b;  
}
```

Concrete values	Symbolic values	Path conditions
a = 123	a = α	-

Concolic execution

Practical example: branch 2

```
int whatever (int a)
{
    int b = 3;

    if (a < b) ←
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43)
    {
        a += 1234;
    }

    printf ("kalm");

    return a - b;
}
```

Concrete values	Symbolic values	Path conditions
a = 123	a = α	-
a = 123	a = α	$\neg(a < b)$

Concolic execution

Practical example: branch 2

```
int whatever (int a)
{
    int b = 3;

    if (a < b)
    {
        printf ("PANIK!");
        fail ();
    }
    else if (a == 43) ←
    {
        a += 1234;
    }

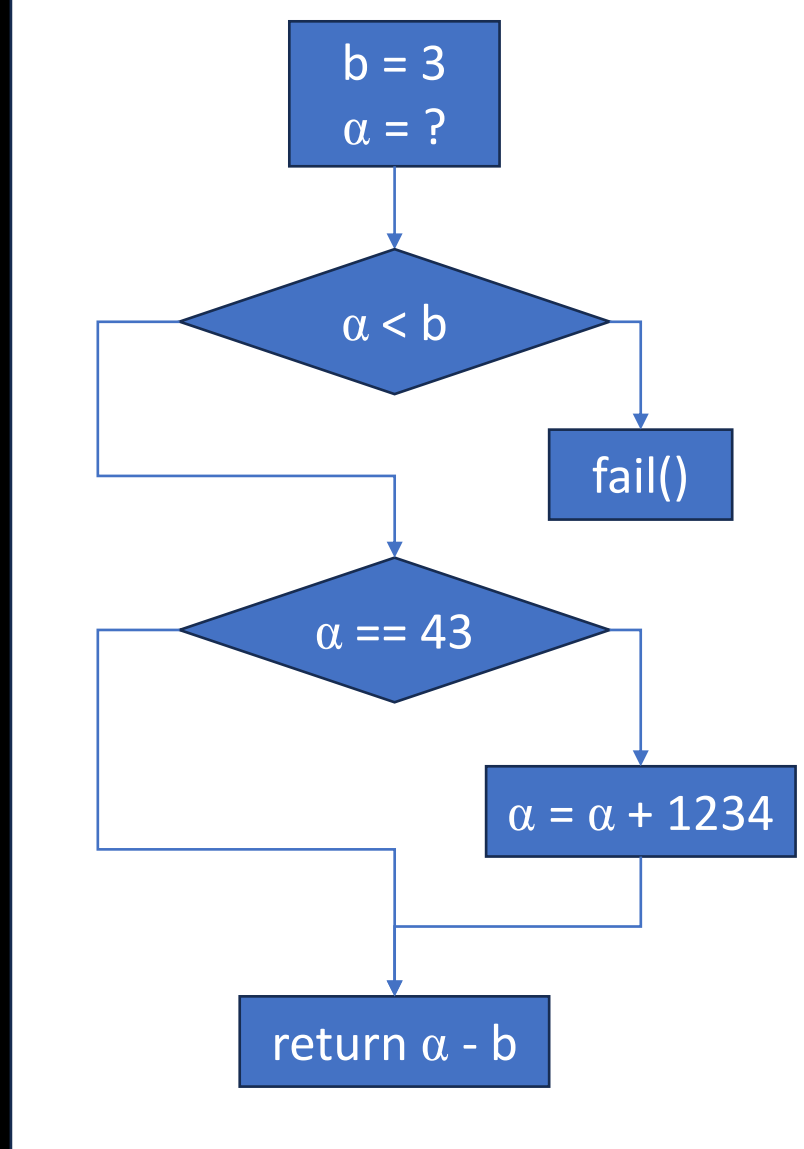
    printf ("kalm");

    return a - b;
}
```

Concrete values	Symbolic values	Path conditions
a = 123	a = α	-
a = 123	a = α	$\neg(a < b)$
a = 123	a = α	$\neg(a < b) \wedge (a \neq 43)$

Concolic execution

Multiple executions lead to full coverage



Execution 1
Execution 2
Execution 3

angr: platform and use cases

About angr

Scope

- Binary **concolic** analysis platform
- **Emulation** of multiple architectures (arm64, x86, mips, JVM bytecode, ...) and executable formats (PE, ELF, Mach-O, ...)
- **Multi-platform** (all the Python fanboys raise your hand)
- Applications:
 - CFG generation (IDA-like) // <https://github.com/angr/angr-management>
 - Search for vulnerabilities // <https://github.com/angr/angr>
 - Exploit generation // <http://github.com/shellphish/rex>
 - ...

About angr

Quick Start

```
C:\ > pip install angr
(...)
C:\ > python
>>> import angr
>>> proj = angr.Project ("fauxware")
>>> proj.arch
<Arch AMD64 (LE)>
>>> proj.loader.main_object
<ELF Object fauxware, maps [0x400000:0x60105f]>

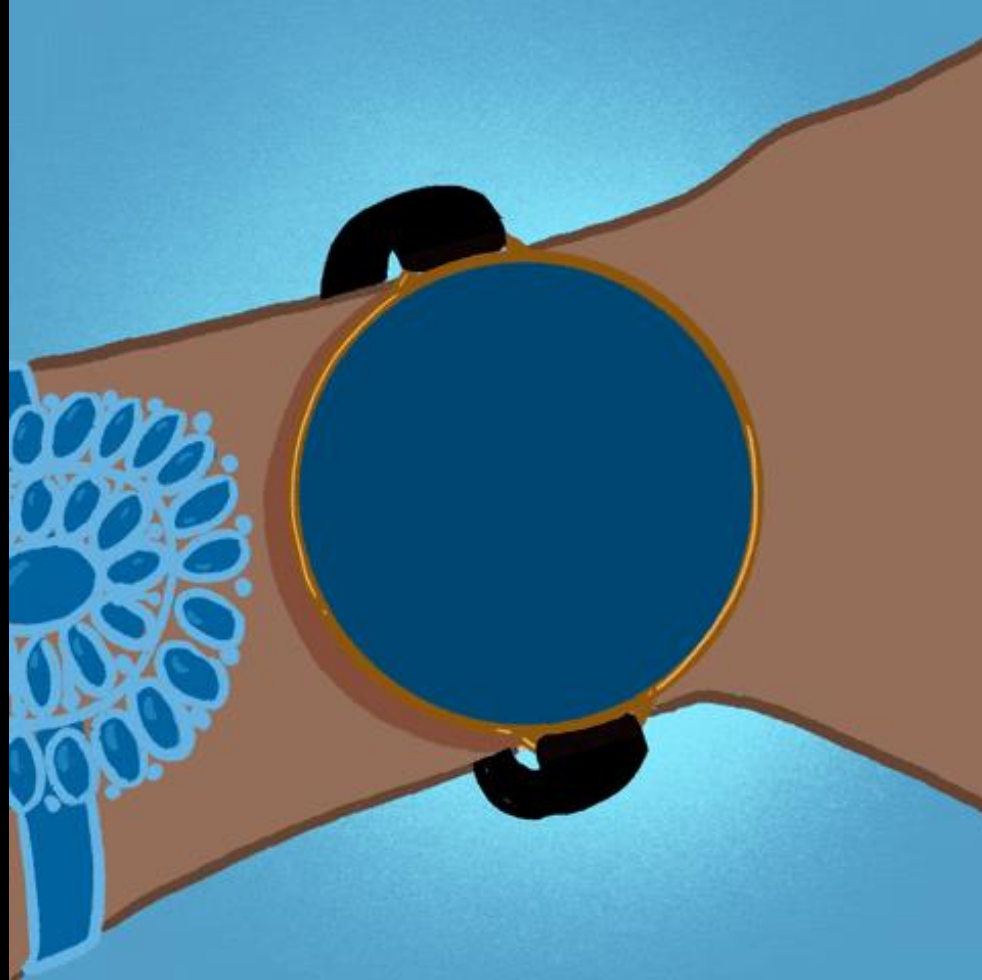
>>> # Create a specific state starting at the entrypoint
>>> state = proj.factory.entry_state ()
>>> state.regs.rip
<BV64 0x400580>
```

```
>>> # Constraint solver
>>> x = state.solver.BVS ("x", 64)
>>> y = state.solver.BVS ("y", 64)
>>> state.solver.add (x - y >= 4)
[<Bool x_3_64 - y_4_64 >= 0x4>]
>>> state.solver.add (y > 0)
[<Bool y_4_64 > 0x0>]
>>> state.solver.eval (x) # Concretize a value
3
>>> state.solver.eval (y)
18446744073709551615

>>> # Simulations are controlled by the Simulation Manager
>>> simgr = proj.factory.simulation_manager (state)
>>> simgr.step () #
<SimulationManager with 1 active>
>>> simgr.active[0].regs.rip
<BV64 0x400540>
```

15' Break

(+ get **angr**, **IDA**/Ghidra/whatever and **angr_ctf** ready)



Learning on-the-go with CTFs

angr CTF

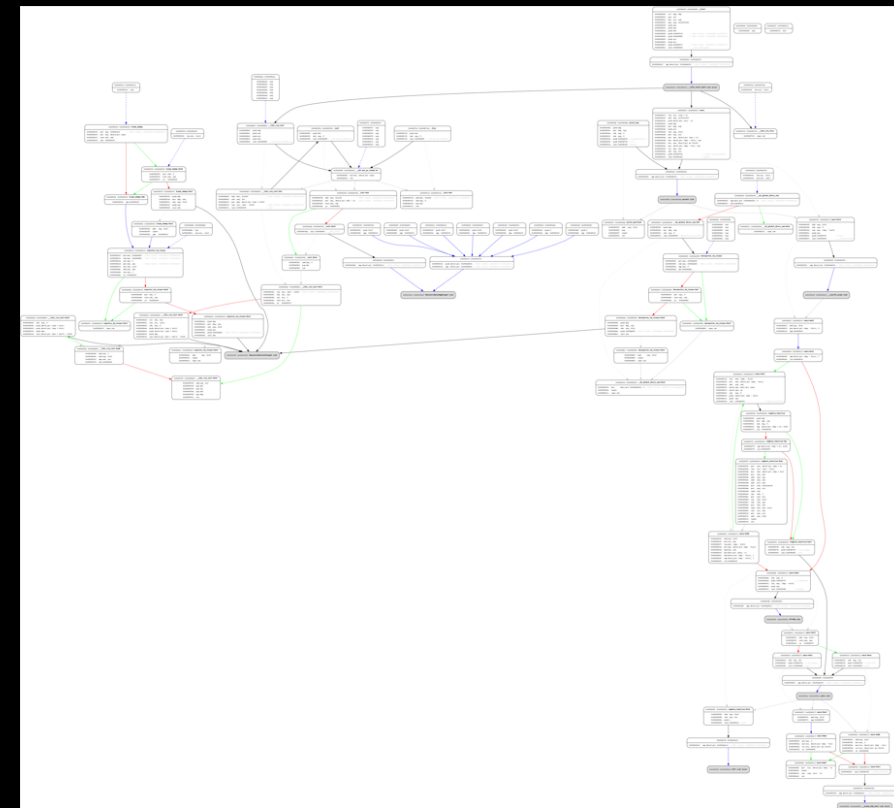
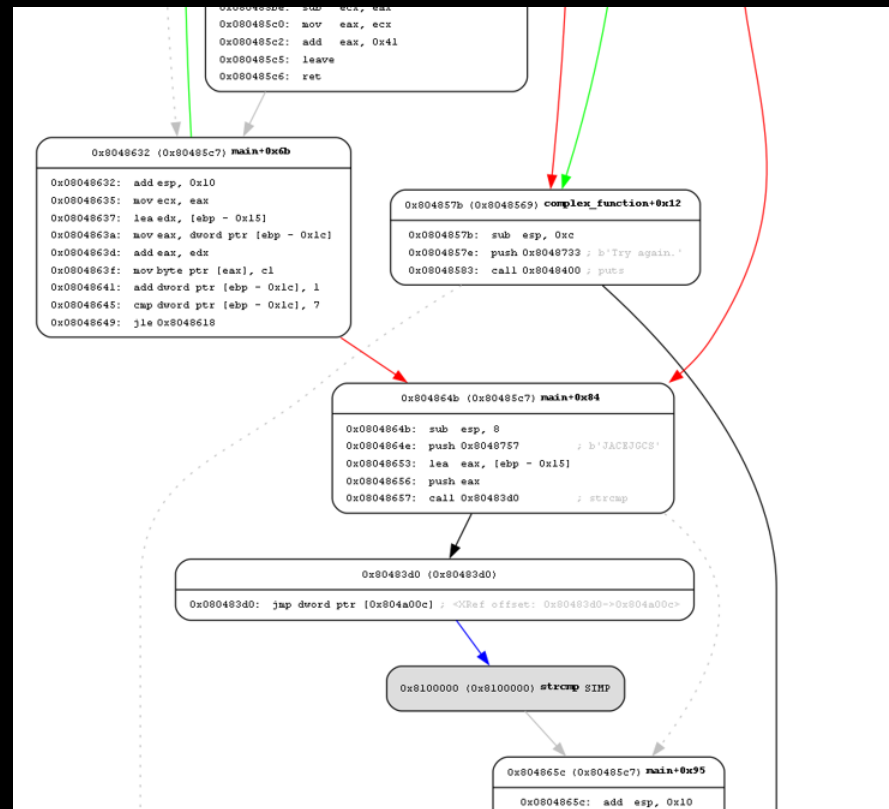
00_angr_find // Option 1: Manually create CFG

```
C:\> pip install angr-utils
C:\> python
>>> import angr
>>> from angrutils import *
>>> proj = angr.Project ("..\00_angr_find", load_options={'auto_load_libs': False})
>>> cfg = proj.analyses.CFG ()
>>> plot_cfg (cfg, "ctf_00-CFG", asminst = True)
```

Requires

<https://graphviz.org/download/>

to paint the CFG



angr CTF

00_angr_find // Option 2: Use angr-management

C:\ > pip install angr-management
C:\ > angr-management

angr management

File View Analyze Plugins Help

Load a new binary... Ctrl+O

Load a trace file... Ctrl+Shift+T

Load recent

Load angr database... Ctrl+L

Save angr database... Ctrl+S

Save angr database as... Ctrl+Shift+S

Save patched binary as...

Load a new trace...

Preferences... Ctrl+,

Exit

New simulation

CFG

The screenshot displays the angr-management interface. At the top, a blue arrow points to the 'New simulation' button. Below it, the 'CFG' (Control Flow Graph) is visible, showing a graph of instructions and their flow. The main window shows the disassembly of the 'main' function (00485c7). The console at the bottom shows logs from angr-management, including error messages about memory operands and a successful job completion.

Name	Tags	Address	Binary
_init		8048394	00_angr_fi
sub_80483c0		80483c0	00_angr_fi
strcmp		80483d0	00_angr_fi
printf		80483e0	00_angr_fi
__stack_chk_fail		80483f0	00_angr_fi
puts		8048400	00_angr_fi
exit		8048410	00_angr_fi
__libc_start_main		8048420	00_angr_fi
__isoc99_scanf		8048430	00_angr_fi
__gmon_start__		8048440	00_angr_fi
_start		8048450	00_angr_fi
sub_8048471		8048471	00_angr_fi
__x86_get_pc_th...		8048480	00_angr_fi
deregister_tm_c...		8048490	00_angr_fi
register_tm_clo...		80484c0	00_angr_fi
__do_global_..._dt...		8048500	00_angr_fi
frame_dummy		8048520	00_angr_fi
print_msg		804854b	00_angr_fi
complex_functi...		8048569	00_angr_fi
main		80485c7	00_angr_fi
__libc_csu_init		8048660	00_angr_fi
__libc_csu_fini		8048710	00_angr_fi
_fini		8048714	00_angr_fi

```
08048618 lea     edx, [ebp-0x15] {s_19}
08048619 j     ptr [ebp-0x1c] {s_20}
0804861a ptr [eax]
0804861b [ebp-0x1c] {s_20}
0804861c junction
0804861d -0x15] {s_19}
0804861e j     ptr [ebp-0x1c] {s_20}
0804861f [eax], cl
08048620 [ebp-0x1c] {s_20}, 0x1

loc_0x804864b:
0804864b sub     esp, 0x8
0804864e push  eax, [ebp-0x15] {s_19}
08048653 lea   eax, [ebp-0x15] {s_19}
08048656 push  eax
08048657 call  strcmp
0804865c add   esp, 0x10
0804865f test  eax, eax
08048661 je    0x8048675

loc_0x8048663:
08048663 sub     esp, 0xc
08048666 push  0x8048733 "Try again."
0804866b call  puts
08048670 add   esp, 0x10
08048673 jmp   0x8048685

loc_0x8048675:
08048675 sub     esp, 0xc
08048678 push  0x8048760 "Good Job."
0804867d call  puts
08048682 add   esp, 0x10

loc_0x8048685:
08048685 mov   eax, 0x0
0804868a mov   ecx, dword ptr [ebp-0xc] {s_10}
0804868d xor   ecx, dword ptr gs:[0x14]
08048694 je    0x804869b
```

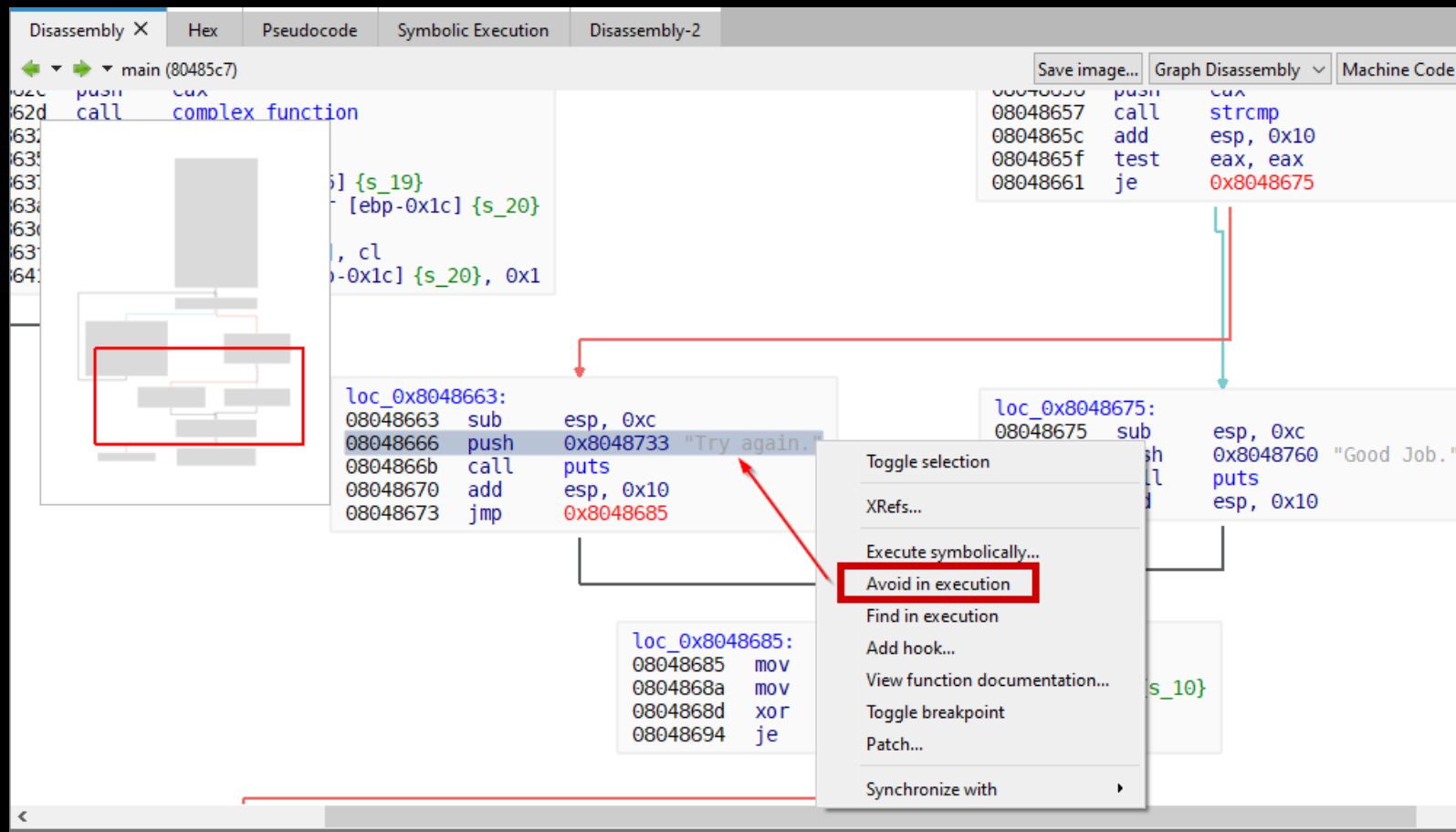
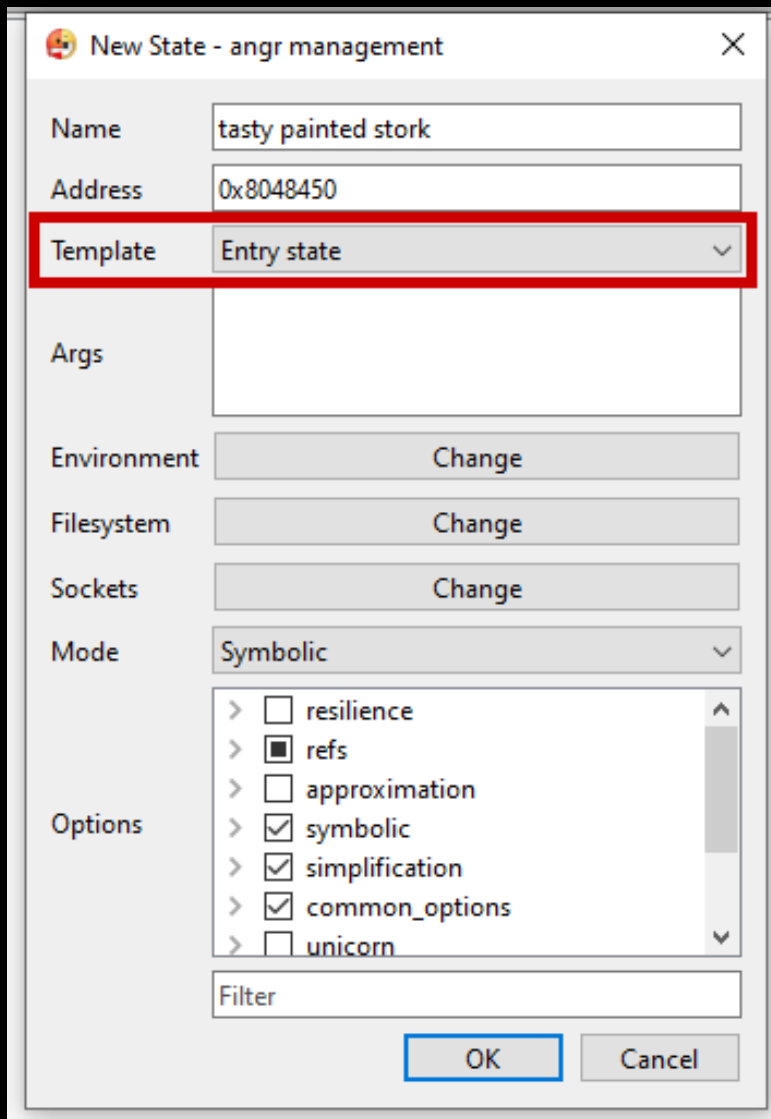
Timestamp	Source	Content
15:59:15	angrmanageme...	Instruction 0x8048629 has two memory operands. Please report it on GitHub.
15:59:15	angrmanageme...	Instruction 0x804863a has two memory operands. Please report it on GitHub.
15:59:15	angrmanageme...	Job "Variable Recovery" completed after 0.69 seconds
15:59:15	angrmanageme...	Instruction 0x8048641 has two memory operands. Please report it on GitHub.

Functions

Console

angr CTF

00_angr_find // Option 2: Use angr-management // Create simulation



angr CTF

00_angr_find // Option 2: Use angr-management // Manage simulation

The screenshot displays the angr Symbolic Execution interface. The main window shows a state transition diagram with three nodes:

- State 0x8048569
Function: complex_function+0
- State 0x8048575
Function: complex_function+c
- State 0x804857b
Function: complex_function+12

Red arrows indicate a transition from the top state to both the bottom-left and bottom-right states. A red rectangle highlights the two bottom states. The right-hand panel, titled "SimulationManagers", shows the "late red-winged blackbird" simulation manager. The "active (2)" section lists two active states: <SimState @ 0x804857b> and <SimState @ 0x8048575>. The "Step actives until branch" button is highlighted with a red rectangle.

??? It doesn't work (or I don't know how to use it)

<https://docs.angr.io/en/latest/getting-started/helpwanted.html#angr-management>

angr CTF

00_angr_find // Option 3: Ghidra + angr CLI

Command Prompt - python

```
>>> import angr
>>> project = angr.Project("dist\\00_angr_find")
>>> project.loader.all_objects
[<ELF Object 00_angr_find, maps [0x8048000:0x804a03f]>, <ExternObject Object cle##externs, maps [0x8100000:0x8100018]>, <ExternObject Object cle##externs, maps [0x8200000:0x8207fff]>, <ELFTLSObjectV2 Object cle##tls, maps [0x8300000:0x8314807]>, <KernelObject Object cle##kernel, maps [0x8400000:0x8407fff]>]
>>> project.loader.find_symbol("main")
<Symbol "main" in 00_angr_find at 0x80485c7>
>>>
```

Function Graph [CodeBrowser: angr_ctf:/00_angr_find]

Function Graph - main - 9 vertices (00_angr_find)

080485c7 - main

```
undefined main(undefined1 param_1, undefined4 param_2)
  undefined AL:1 <RETURN>
  undefined1 Stack[0x4]:1 param_1
  undefined4 Stack[0x8]:4 param_2
  undefined4 Stack[0x0]:4 local_res0
  undefined4 Stack[-0xc]:4 local_c
  undefined4 Stack[-0x14]:4 local_14
  undefined1 Stack[-0x1d]:1 local_1d
  undefined4 Stack[-0x24]:4 local_24
  undefined4 Stack[-0x34]:4 local_34
  main
...85c7 LEA ECX=>param_1,[ESP + 0x4]
...85cb AND ESP,0xffffffff0
...85ce PUSH dword ptr [ECX + local_res...
...85d1 PUSH EBP
...85d2 MOV EBP,ESP
...85d4 PUSH ECX
...85d5 SUB ESP,0x34
...85d8 MOV EAX,ECX
...85da MOV EAX,dword ptr [EAX + param...
...85dd MOV dword ptr [EBP + local_34]...
...85e0 MOV EAX,GS:[0x14]
...85e6 MOV dword ptr [EBP + local_14]...
...85e9 XOR EAX,EAX
...85eb SUB ESP,0xc
...85ee PUSH s Enter the password: 0804...
```

angr CTF

00_angr_find // Option 3: Ghidra + angr CLI

```
undefined main(undefined param_1, undefined param_2)
undefined AL:1 <RETURN>
undefined Stack[0x4]:1 param_1
undefined Stack[0x8]:4 param_2
undefined Stack[0x0]:4 local_res0
undefined Stack[-0xc]:4 local_c
undefined Stack[-0x14]:4 local_14
undefined Stack[-0x1d]:4 local_1d
undefined Stack[-0x24]:4 local_24
undefined Stack[-0x34]:4 local_34
main
...85c7 LEA ECX=>param_1,[ESP + 0x4]
...85cb AND ESP,0xffffffff0
...85ce PUSH dword ptr [ECX + local_res-
...85d1 PUSH EBP
...85d2 MOV EBP,ESP
...85d4 PUSH ECX
...85d5 SUB ESP,0x34
...85d8 MOV EAX,ECX
...85da MOV EAX,dword ptr [EAX + param..
...85dd MOV dword ptr [EBP + local_34...
...85e0 MOV EAX,GS:[0x14]
...85e6 MOV dword ptr [EBP + local_14...
...85e9 XOR EAX,EAX
...85eb SUB ESP,0xc
...85ee PUSH s_Enter_the_password:_0804
...85f3 CALL <EXTERNAL>::printf
...85f8 ADD ESP,0x10
...85fb SUB ESP,0x8
...85fe LEA EAX=>local_1d[EBP + -0x15]
...8601 PUSH EAX
...8602 PUSH DAT_08048753
...8607 CALL <EXTERNAL>::__isoc99_scanf
...860c ADD ESP,0x10
...860f MOV dword ptr [EBP + local_24...
...8616 JMP LAB_08048645

0804864b
...864b SUB ESP,0x8
...864e PUSH s_JACEJGCS_08048757
...8653 LEA EAX=>local_1d[EBP + -0x15]
...8656 PUSH EAX
...8657 CALL <EXTERNAL>::strcmp
...865c ADD ESP,0x10
...865f TEST EAX,EAX
...8661 JZ LAB_08048675

08048645 - LAB_08048645
LAB_08048645
...8645 CMP dword ptr [EBP + local_24...
...8649 JLE LAB_08048618

08048618 - LAB_08048618
LAB_08048618
...8618 LEA EDX=>local_1d[EBP + -0x15]
...861b MOV EAX,dword ptr [EBP + local_3...
...861e ADD EAX,EDX
...8620 MOVZX EAX,byte ptr [EAX]
...8623 MOVSB EAX,AL
...8626 SUB ESP,0x8
...8629 PUSH dword ptr [EBP + local_24...
...862c PUSH EAX
...862d CALL complex_function
...8632 ADD ESP,0x10
...8635 MOV ECX,EAX
...8637 LEA EDX=>local_1d[EBP + -0x15]
...863a MOV EAX,dword ptr [EBP + local_3...
...863d ADD EAX,EDX
...863f MOV byte ptr [EAX],CL
...8641 ADD dword ptr [EBP + local_24...

08048663
...8663 SUB ESP,0xc
...8666 PUSH s_Try_again_080487
...866b CALL <EXTERNAL>::puts
...8670 ADD ESP,0x10
...8673 JMP LAB_08048685

08048685 - LAB_08048685
LAB_08048685
...8685 MOV EAX,0x0
...868a MOV ECX,dword ptr [EBP + local_...
...868d XOR ECX,dword ptr GS:[0x14]
...8694 JZ LAB_0804869b

08048675 - LAB_08048675
LAB_08048675
...8675 SUB ESP,0xc
...8678 PUSH s_Good_Job_080487
...867d CALL <EXTERNAL>::puts
...8682 ADD ESP,0x10

0804869b - LAB_0804869b
LAB_0804869b
...869b MOV ECX,dword ptr [EBP + local_...
...869e LEAVE
...869f LEA ESP=>local_res0[ECX + -0x...
...86a2 RET
```

angr CTF

00_angr_find // Option 3: Ghidra + angr CLI // Ghidra analysis

- Important points discovered with Ghidra:
 1. The program gets some **user input** using **scanf()**
 2. A loop transforms that input using **complex_function()**
 3. The result is compared with the global variable **s_JACEJGCS**
 4. If the comparison **fails**, the program prints **“Try again”**
 5. If the comparison **succeeds**, the program prints **“Good job”**
- angr takes care of symbolizing this simple scanf
- We can ask angr to gather the constraints that must be satisfied to print **“Good Job”**

angr CTF

00_angr_find // Option 3: Ghidra + angr CLI // angr modelling and execution

```
>>> initial_state = project.factory.entry_state ()
>>> simulation = project.factory.simgr (initial_state)
>>> target_address = 0x8048675 # Address to print "Good Job"
>>> avoid_address = 0x8048663 # Not needed now, but good to optimise
>>> simulation.explore (find = target_address, avoid = avoid_address)
WARNING | (...)
<SimulationManager with 16 deadended, 1 found, 1 avoid>
>>> simulation.found
[<SimState @ 0x8048675>]
>>> solution.posix.dumps (0) # STDIN
b'JXWVXRKX'
>>> # Simulations that failed to meet the requirements
>>> simulation.deadended[12].posix.dumps (0)
b'AAAAAB\x00\x00'
>>> simulation.deadended[13].posix.dumps (0)
b'AAAAAA[\x00'
>>> simulation.deadended[14].posix.dumps (0)
b'AAAAAAC\x00'
>>> simulation.deadended[15].posix.dumps (0)
b'AAAAABA['
```

Different stashes depending on the simulation's state

Solution to this challenge

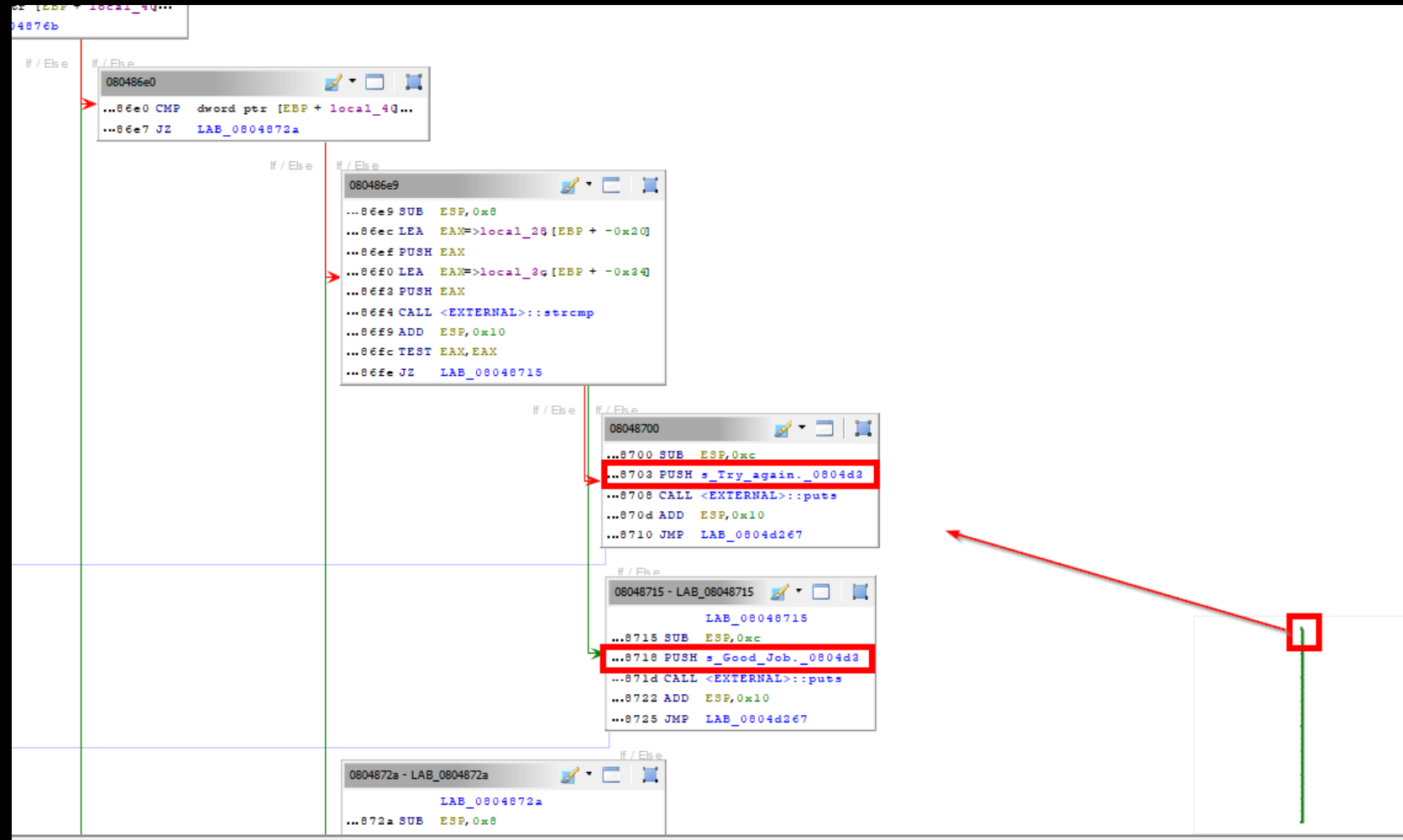
```
kali@kali: /tmp/tmp.xm5xs9m4TU$ ./00_angr_find
Enter the password: JXWVXRKX
Good Job.
```

angr CTF

02_angr_find_condition // Ghidra

Next example: 02_angr_find_condition

- Huge if-else tree
- All statements end on:
 - “Good Job”
 - or
 - “Try again”



angr CTF

02_angr_find_condition // angr

- angr lets us add conditions to avoid paths, aside from of hardcoded addresses
 - take the **current state** as a parameter
 - return **True** (the state fulfills the condition) or **False**
- We can also use STDIN or STDOUT for these conditions
- **QUIZ:** What should these functions return?

```
def is_successful (state):  
    stdout_output = state.posix.dumps (sys.stdout.fileno ())  
    return ???  
  
def should_abort (state):  
    stdout_output = state.posix.dumps (sys.stdout.fileno ())  
    return ???  
  
simulation.explore (find = is_successful, avoid = should_abort)
```

API docs: <https://docs.angr.io/en/latest/api.html#state>

angr CTF

02_angr_find_condition // angr // solution

```
>>> initial_state = project.factory.entry_state ()
>>> simulation = project.factory.simgr (initial_state)
>>> def is_successful (state):
...     stdout_output = state.posix.dumps (sys.stdout.fileno ())
...     return b'Good Job.' in stdout_output
>>> def should_abort (state):
...     stdout_output = state.posix.dumps (sys.stdout.fileno ())
...     return b'Try again.' in stdout_output
...
>>> simulation.explore (find = is_successful, avoid = should_abort)
<SimulationManager with 1 found, 17 avoid>
>>> simulation.found[0].posix.dumps (sys.stdin.fileno ())
b'HETOBRCU'
```

angr CTF

03_angr_symbolic_registers // Ghidra

Next example: 03_angr_symbolic_registers

- Complex scanf ()

```
get_user_input                                     XREF[4]:      Entry Point(*), main:0804897b
                                                    08048afc, 08048be0(*)

0804890c 55          PUSH      EBP
0804890d 89 e5       MOV      EBP,ESP
0804890f 83 ec 18    SUB      ESP,0x18
08048912 65 8b 0d    MOV      ECX,dword ptr GS:[0x14]
           14 00 00 00
08048919 89 4d f4    MOV      dword ptr [EBP + local_10],ECX
0804891c 31 c9       XOR      ECX,ECX
0804891e 8d 4d f0    LEA     ECX=>local_14,[EBP + -0x10]
08048921 51         PUSH     ECX
08048922 8d 4d ec    LEA     ECX=>local_18,[EBP + -0x14]
08048925 51         PUSH     ECX
08048926 8d 4d e8    LEA     ECX=>local_1c,[EBP + -0x18]
08048929 51         PUSH     ECX
0804892a 68 93 8a   PUSH     s_%x_%x_%x_08048a93          = "%x %x %x"
           04 08
0804892f e8 9c fa   CALL    <EXTERNAL>:::__isoc99_scanf   undefined __isoc99_scanf()
           ff ff
```

angr CTF

03_angr_symbolic_registers // angr // How to create a symbol

- We can manually recreate the state after `scanf()` with our symbolic values
- **EAX**, **EBX** and **EDX** contain the read values (`%x` → lowercase hex values)
- Symbols are handled by **claripy** (angr's solver engine)

- An example on how to create a symbol and assign it to EAX:

```
>>> import claripy
>>> eax = claripy.BVS('<symbol_name>', 32) # registers are 32-bit long
>>> initial_state.regs.eax = eax
```

angr CTF

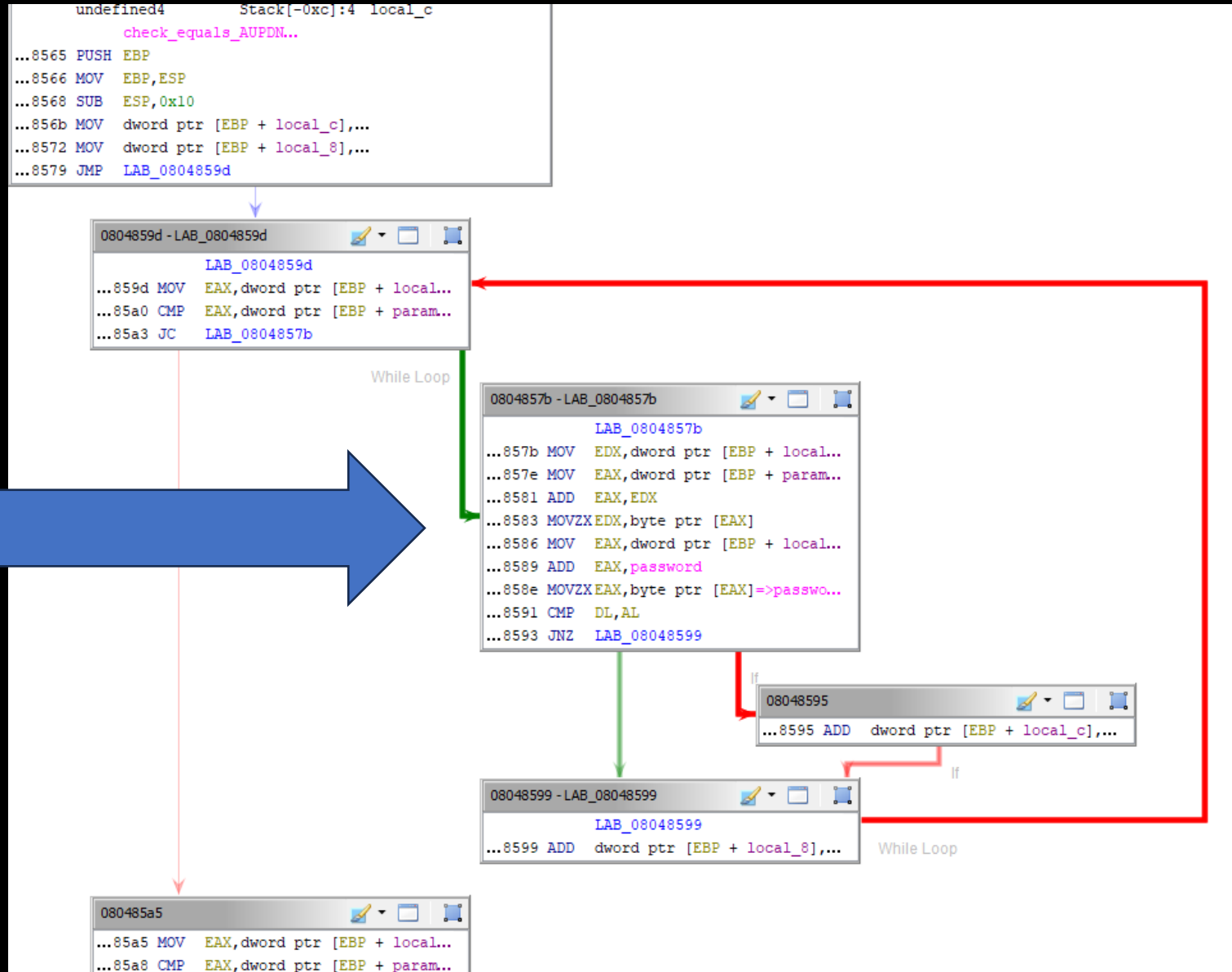
03_angr_symbolic_registers // angr // Solution

```
>>> start_address = 0x08048946 # Right after get_user_input()
>>> initial_state = project.factory.blank_state (addr = start_address)
>>> eax = claripy.BVS ('password0', 32) # registers are 32-bit long
>>> ebx = claripy.BVS ('password1', 32) # registers are 32-bit long
>>> edx = claripy.BVS ('password2', 32) # registers are 32-bit long
>>> initial_state.regs.eax = eax
>>> initial_state.regs.ebx = ebx
>>> initial_state.regs.edx = edx
>>> simulation = project.factory.simgr (initial_state)
>>> simulation.explore (find = 0x080489e9, avoid = 0x080489dc)
<SimulationManager with 1 found, 3 avoid>
>>> "%x" % simulation.found[0].solver.eval (eax)
'b9ffd04e'
>>> "%x" % simulation.found[0].solver.eval (ebx)
'ccf63fe8'
>>> "%x" % simulation.found[0].solver.eval (edx)
'8fd4d959'
```

angr CTF

08_angr_constraints // path explosion

- Comparing each character of the password leads to **path explosion**
- 2 branches to explore for each of the 16 characters: **$2^{16} = 65536$ states** to explore
- Solution: stop *before* this check and constraint it manually



angr CTF

08_angr_constraints // manual constraint

- Tip: the CPython REPL takes ages to finish
 - Better use a standalone script (or maybe another REPL - ?)

```
start_addr = 0x0804863C # Right after scanf()

initial_state = project.factory.blank_state (
    addr = start_addr,
    add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY }
)

passwd_size = 16 # 16 symbolic Bytes for the password
password = claripy.BVS ('password', 8 * passwd_size)
passwd_address = 0x804a040 # Made-up address to "store" the user input
initial_state.memory.store (passwd_address, password)

target_addr = 0x08048685 # After the loop, but before check_equals_()
simulation = project.factory.simgr (initial_state)
simulation.explore (find = target_addr)
# <SimulationManager with 2 active, 30 deadended, 1 found>
solution_state = simulation.found[0]

constrained_bv = solution_state.memory.load (passwd_address, passwd_size)
target_passwd = b'OSIWHBXIFOQVSBZB'

solution_state.add_constraints (constrained_bv == target_passwd)

solved_passwd = solution_state.solver.eval (password, cast_to = bytes)
print (solved_passwd)
```

angr CTF

10_angr_simprocedures // Same problem, different solution

- The same problem can also be solved by implementing the problematic function Python using **SimProcedures**
- Useful to avoid uninteresting or already tested functions
 - the less code we symbolize, the better
- angr already implements many common library functions as SimProcedures:
<https://github.com/angr/angr/tree/master/angr/procedures>

- Very easy to implement:

```
class Whatever (SimProcedure):  
    def run (self, arg):  
        return arg + 3  
  
project.hook_symbol (<target>, Whatever ())
```


angr CTF

10_angr_simprocedures // Running our SimProcedure

```
>>> project = angr.Project ("\\.\\dist\\10_angr_simprocedures")
>>> # This scanf is a simple and we don't need to manually constrain the password buffer
>>> initial_state = project.factory.entry_state ()
>>> class CustomSimproc (angr.SimProcedure):
...     def run (self, string_ptr, length):
...         string = self.state.memory.load (string_ptr, length)
...         return claripy.If ( # The expression can be symbolic, hence claripy being used here
...             string == b'ORSDDWXHZURJRBBDH',
...             claripy.BVV (1, 32), # 32-bit vector // TRUE
...             claripy.BVV (0, 32) # 32-bit vector // FALSE
...         )
...
...
>>> project.hook_symbol ('check_equals_ORSDDWXHZURJRBBDH', CustomSimproc ())
0x80485f5
>>> simulation = project.factory.simgr (initial_state)
>>> simulation.explore (find = is_successful, avoid = should_abort)
>>> simulation.found[0].posix.dumps (0)
b'MSWKNJNAVTTTOZMRY'
```

angr CTF

17_angr_arbitrary_jump // Finding a Buffer Overflow

- Theory: if a **jump destination is unconstrained**, that means we (maybe) overwrote it with **user input** → possible buffer overflow
- Unconstrained states are normally discarded by angr
- Let's practice with a basic buffer overflow:

```
2 void read_input(void)
3
4 {
5     undefined local_24 [32];
6
7     __isoc99_scanf("%s", local_24);
8     return;
9 }
```

angr CTF

17_angr_arbitrary_jump // Finding unconstrained jumps

```
>>> project = angr.Project (".\dist\17_angr_arbitrary_jump")
>>> symbolic_input = claripy.BVS ("input", 100 * 8) # Simulate a big user input
>>> initial_state = project.factory.entry_state (stdin = symbolic_input)
>>> simulation = project.factory.simgr ( # Manually create the needed stashes
...   initial_state,
...   save_unconstrained = True, # Needed to check for unconstrained EIP
...   stashes = {
...     'active' : [initial_state],
...     'unconstrained' : [],
...     'found' : [],
...     'not_needed' : []
...   }
... )
>>> # explore() doesn't work automatically anymore. We must do it manually
>>> while (simulation.active or simulation.unconstrained) and (not simulation.found):
...   for state in simulation.unconstrained:
...     simulation.move ('unconstrained', 'found')
...     simulation.step ()
...
<SimulationManager with 2 found>
```


Resources

- <https://docs.angr.io>
- <https://www.youtube.com/watch?v=yRVZPvHYHzw>
- https://github.com/jakespringer/angr_ctf
- <https://blog.notso.pro/2019-03-20-angr-introduction-part0/>
- <https://www.youtube.com/watch?v=TIEjgqSXYNE>

- angr API docs: <https://docs.angr.io/en/latest/api.html>
- claripy API docs:
<https://docs.angr.io/projects/claripy/en/latest/api.html>
- CLE API docs: <https://api.angr.io/projects/cle/en/latest/>