CS 527

Lab 2: Symbolic Execution

Presenter: Sajad Meisami

25 Oct 2022

angr

angr is a platform-agnostic binary analysis framework.

angr is a suite of Python 3 libraries that let you load a binary and do a lot of cool things to it:

- Disassembly and intermediate-representation lifting
- Program instrumentation
- Symbolic execution
- Control-flow analysis
- Data-dependency analysis
- Value-set analysis (VSA)
- Decompilation

How to install *angr:*

- https://github.com/angr/angr
- **Install the environment:**
- https://docs.angr.io/introductory-errata/install
- **Tutorial:**
- https://blog.notso.pro/2019-03-20-angr-introduction-part0/

```
sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~$ cd Desktop
sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$ workon angr
(angr) sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$ python3 lab2.py
```

To get started:

a) Download the binary.

b) Download Angr and configure it up. Please refer to the tutorial on how to install.

Task 1: Control-flow graph generation

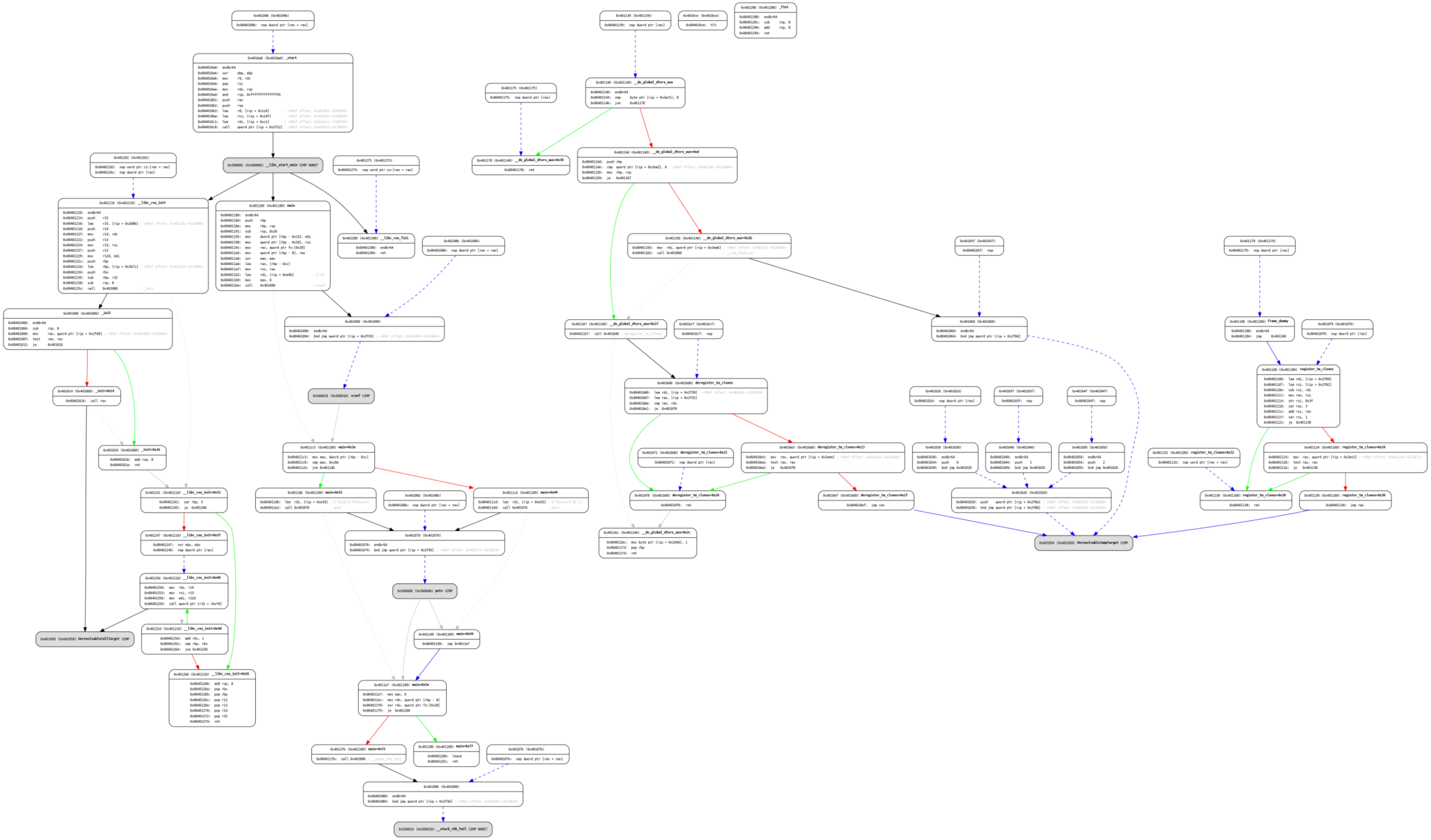
Given a binary, your job is to output the interprocedural control-flow graph for the entire binary into a **dot** format file.

Moreover, you need to print out the following numbers:

1) number of nodes in the graph
2) number of edges in the graph
3) number of different instruction types

```
import angr
import os
import argparse
import angrutils, subprocess

binary_input = angr.Project("test", load_options={"auto_load_libs": False})

cfg = binary_input.???.???(data_references=True, normalize=True)

nodelist = list(???)
edgelist = list(???)

nodelist1 = [node for node in list(cfg.graph.???) if node.block !=None]
print("number of nodes in the graph:",???)
print("number of edges in the graph:",???)

allIns = set()
for node in nodelist1:
    for ins in ((node.block.disassembly.insns)):
      …// create cfg in .dot format
print("number of different instruction types:", len(???))
```

```
sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~$ cd Desktop
sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$ workon angr
(angr) sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$ python3 lab2.py
WARNING | 2022-10-24 16:00:04,316 | cle.loader | The main binary is a position-ind
ependent executable. It is being loaded with a base address of 0x400000.
number of nodes in the graph: [redacted]
number of edges in the graph: [redacted]
{'test', 'sar', [redacted]
', 'sub', 'je',
'}
number of different instruction types: [redacted]
(angr) sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$
```



Task 2: Symbolic Execution

Given a binary, your job is to write a script to:

- 1) find addresses for all '**put**' functions
- 2) feed the addresses as targets to the symbolic execution engine
- 3) perform symbolic execution to generate correct inputs to trigger these 'put' functions.

```
(angr) sajad@sajad-HP-Pavilion-Gaming-Desktop-TG01-2xxx:~/Desktop$ objdump -d test

test:     file format elf64-x86-64


Disassembly of section .init:

0000000000001000 <_init>:
    1000:       f3 0f 1e fa             endbr64
    1004:       48 83 ec 08             sub    $0x8,%rsp
    1008:       48 8b 05 d9 2f 00 00    mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start
    100f:       48 85 c0                test   %rax,%rax
    1012:       74 02                   je     1016 <_init+0x16>
    1014:       ff d0                   call   *%rax
    1016:       48 83 c4 08             add    $0x8,%rsp
    101a:       c3                      ret

Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:       ff 35 8a 2f 00 00       push   0x2f8a(%rip)        # 3fb0 <_GLOBAL_OFFSET_TA
    1026:       f2 ff 25 8b 2f 00 00    bnd jmp *0x2f8b(%rip)        # 3fb8 <_GLOBAL_OFFSET_
    102d:       0f 1f 00                nopl   (%rax)
    1030:       f3 0f 1e fa             endbr64
    1034:       68 00 00 00 00          push   $0x0
    1039:       f2 e9 e1 ff ff ff       bnd jmp 1020 <.plt>
    103f:       90                      nop
```

```
    11ab:       48 8d 45 f4             lea    -0xc(%rbp),%rax
    11af:       48 89 c6                mov    %rax,%rsi
    11b2:       48 8d 3d 4b 0e 00 00    lea    0xe4b(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    11b9:       b8 00 00 00 00          mov    $0x0,%eax
    11be:       e8 cd fe ff ff          call   1090 <scanf@plt>
    11c3:       8b 45 f4                mov    -0xc(%rbp),%eax
    11c6:       3d 8e 0c 00 00          cmp    $0xc8e,%eax
    11cb:       75 0e                   jne    11db <main+0x52>
    11cd:       48 8d 3d 33 0e 00 00    lea    0xe33(%rip),%rdi        # 2007 <_IO_stdin_used+0x7>
    11d4:       e8 97 fe ff ff          call        <puts@plt>
    11d9:       eb 0c                   jmp    11e7 <main+0x5e>
    11db:       48 8d 3d 34 0e 00 00    lea    0xe34(%rip),%rdi        # 2016 <_IO_stdin_used+0x16>
    11e2:       e8 89 fe ff ff          call        <puts@plt>
    11e7:       b8 00 00 00 00          mov    $0x0,%eax
    11ec:       48 8b 55 f8             mov    -0x8(%rbp),%rdx
    11f0:       64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
    11f7:       00 00
    11f9:       74 05                   je     1200 <main+0x77>
    11fb:       e8 80 fe ff ff          call   1080 <__stack_chk_fail@plt>
    1200:       c9                      leave
    1201:       c3                      ret
    1202:       66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
    1209:       00 00 00
    120c:       0f 1f 40 00             nopl   0x0(%rax)
```

```
import angr
import sys

proj = angr.Project("test", load_options={'auto_load_libs': False})
cfg = proj.analyses.???()

nodelist1 = list(cfg.graph.???)
edgelist1 = list(cfg.graph.???)

for node in ???:
    if node.block is None:
        continue
    for insn in node.block.capstone.insns:
        mne = insn.mnemonic
        if mne == 'call':
            if insn.op_str.endswith("???"):
                addr_target = insn.address
                # print(hex(addr_target))

              //execute binary symbolically
                ...

                print(solution_state.posix.dumps(sys.stdin.fileno()).decode())
```