Manticore

Yan Ivnitskiy
Trail of Bits
@yan

Introduction

- **Manticore as a ..**
    - Basic emulator
    - Basic symbolic execution tool
    - Python library
- **Intro to symbolic execution**
- **Hacking on Manticore**

What is Manticore?

… is a prototyping tool for dynamic binary analysis, with support for symbolic execution, taint analysis, and binary instrumentation.

- An open source system emulator implemented 100% in Python
- Major component of Trail of Bits' Cyber Grand Challenge entry
- Toolkit for symbolic execution experiments

What is Manticore?

- Run a Linux binary with Manticore
- The entire environment is emulated in software
  - CPU (x86/64, ARMv7)
  - MMU
  - Linux
- Emulates Linux, runs on Linux[1]

[1] might run on macOS

What is Manticore?

- Before continuing, everyone should have Manticore installed

```
# pip install manticore
```

or

```
$ git checkout https://github.com/trailofbits/manticore.git
$ cd manticore
$ pip install --user -e .[dev]
```

What is Manticore?

- Standalone tool

```
user@ubuntu /m/h/p/manticore> gcc -static examples/linux/helloworld.c -o hello
user@ubuntu /m/h/p/manticore> manticore ./hello
2017-11-01 15:50:43,716: [3859] m.manticore:INFO: Loading program ./hello
2017-11-01 15:50:44,725: [3859] m.manticore:INFO: Starting 1 processes.
2017-11-01 15:50:50,015: [3859] m.manticore:INFO: Generated testcase No. 0 - Program finished
with exit status: 0L
2017-11-01 15:50:50,022: [3859] m.manticore:INFO: Results in manticore/mcore_pqe1ZV
2017-11-01 15:50:50,022: [3859] m.manticore:INFO: Total time: 5.29676604271
```

What is Manticore?

- Standalone tool

```
user@ubuntu /m/h/p/manticore> ls -lA mcore_pqe1ZV/
total 524
-rw-r--r-- 1 501 user     39 Nov  1 15:50 command.sh
-rw-r--r-- 1 501 user   1224 Nov  1 15:50 test_00000000.messages
-rw-r--r-- 1 501 user      0 Nov  1 15:50 test_00000000.net
-rw-r--r-- 1 501 user 423243 Nov  1 15:50 test_00000000.pkl
-rw-r--r-- 1 501 user      0 Nov  1 15:50 test_00000000.smt
-rw-r--r-- 1 501 user      0 Nov  1 15:50 test_00000000.stderr
-rw-r--r-- 1 501 user      0 Nov  1 15:50 test_00000000.stdin
-rw-r--r-- 1 501 user     14 Nov  1 15:50 test_00000000.stdout
-rw-r--r-- 1 501 user     92 Nov  1 15:50 test_00000000.syscalls
-rw-r--r-- 1 501 user  59191 Nov  1 15:50 test_00000000.trace
-rw-r--r-- 1 501 user   1036 Nov  1 15:50 test_00000000.txt
-rw-r--r-- 1 501 user  48526 Nov  1 15:50 visited.txt
```

What is Manticore?

- A Python library

```
user@ubuntu /m/h/p/manticore> python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from manticore import Manticore
>>> m = Manticore('./hello')
>>> m.verbosity(1)
>>> m.run()
2017-11-01 15:56:40,786: [4035] m.manticore:INFO: Starting 1 processes.
2017-11-01 15:56:46,044: [4035] m.manticore:INFO: Generated testcase No. 0 - Program finished
with exit status: 0L
2017-11-01 15:56:46,051: [4035] m.manticore:INFO: Results in manticore/mcore_hpqdRo
2017-11-01 15:56:46,051: [4035] m.manticore:INFO: Total time: 5.26445698738
>>>
```

Intro to Manticore API

Manticore API

- **Manticore provides a very simple Python API**
  - Inspired by Flask
  - Revolves around a single object (`Manticore`)
- **Just three things you need to know to get started:**
  - The Manticore class
  - Setting hooks
  - Symbolicating/concretizing values
- **http://manticore.readthedocs.io/en/stable/**

Manticore API

```
from manticore import Manticore

m = Manticore.linux("basic")

def main_reached(state):
    print "Reached ", state.cpu.PC

m.add_hook(0x4009ec, main_reached)

m.run(procs=4)
```

Manticore API

```
from manticore import Manticore

m = Manticore.linux("basic")

@m.hook(0x4009ec)
def main_reached(state):
    print "Reached ", state.cpu.PC

m.run(procs=4)
```

Manticore API

- Every hook receives a `state` object as a parameter
- Entire emulated state is accessible through its methods and properties
- Properties
  - `mem`
  - `cpu`
  - `platform`
- Symbolic support
  - `new_symbolic_buffer` / `new_symbolic_value` / `symbolicate_buffer`
  - `solve_n` / `solve_one`
  - `constrain`
- Testcases
  - `save()`
  - `generate_testcase()`
  - `abandon()`

Instrumentation with Manticore

- **Basic DBI example with hello**
  - Hook main and confirm by printing something
  - Hook `write` and display one or more arguments
  - Make `main`() return a different value

What is Manticore?

- Symbolic Exploration

```
user@ubuntu /m/h/p/manticore> gcc -static examples/linux/basic.c -o basic
user@ubuntu /m/h/p/manticore> manticore ./basic
m.manticore:INFO: Loading program ./basic
m.manticore:INFO: Starting 1 processes.
m.manticore:INFO: Generated testcase No. 0 - Program finished with exit status: 0L
m.manticore:INFO: Generated testcase No. 1 - Program finished with exit status: 0L
m.manticore:INFO: Results in /mnt/hgfs/projects/manticore/other/mcore_hME3MC
m.manticore:INFO: Total time: 7.64563584328
```

Simple Example

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

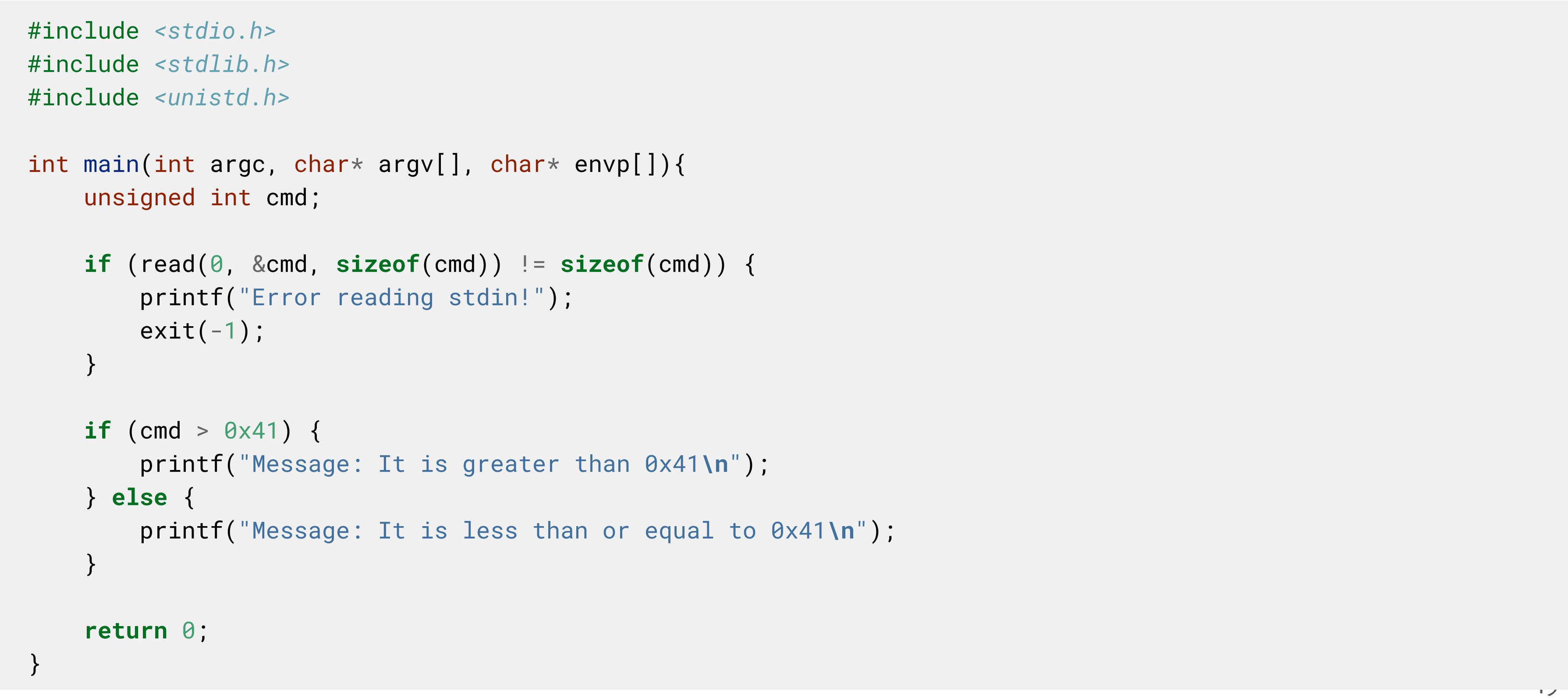
Simple Example

```
user@ubuntu /m/h/p/m/o/mcore_hME3MC> ls -lA
total 1037
-rw-r--r-- 1 501 user    44 Nov  1 16:15 command.sh
-rw-r--r-- 1 501 user  1239 Nov  1 16:15 test_00000000.messages
-rw-r--r-- 1 501 user     0 Nov  1 16:15 test_00000000.stderr
-rw-r--r-- 1 501 user     4 Nov  1 16:15 test_00000000.stdin
-rw-r--r-- 1 501 user    42 Nov  1 16:15 test_00000000.stdout
-rw-r--r-- 1 501 user   538 Nov  1 16:15 test_00000000.syscalls
-rw-r--r-- 1 501 user 69179 Nov  1 16:15 test_00000000.trace
-rw-r--r-- 1 501 user  1033 Nov  1 16:15 test_00000000.txt
-rw-r--r-- 1 501 user  1239 Nov  1 16:15 test_00000001.messages
-rw-r--r-- 1 501 user     0 Nov  1 16:15 test_00000001.stderr
-rw-r--r-- 1 501 user     4 Nov  1 16:15 test_00000001.stdin
-rw-r--r-- 1 501 user    33 Nov  1 16:15 test_00000001.stdout
-rw-r--r-- 1 501 user   493 Nov  1 16:15 test_00000001.syscalls
-rw-r--r-- 1 501 user 68981 Nov  1 16:15 test_00000001.trace
-rw-r--r-- 1 501 user  1033 Nov  1 16:15 test_00000001.txt
-rw-r--r-- 1 501 user 59774 Nov  1 16:15 visited.txt
```

Simple Example

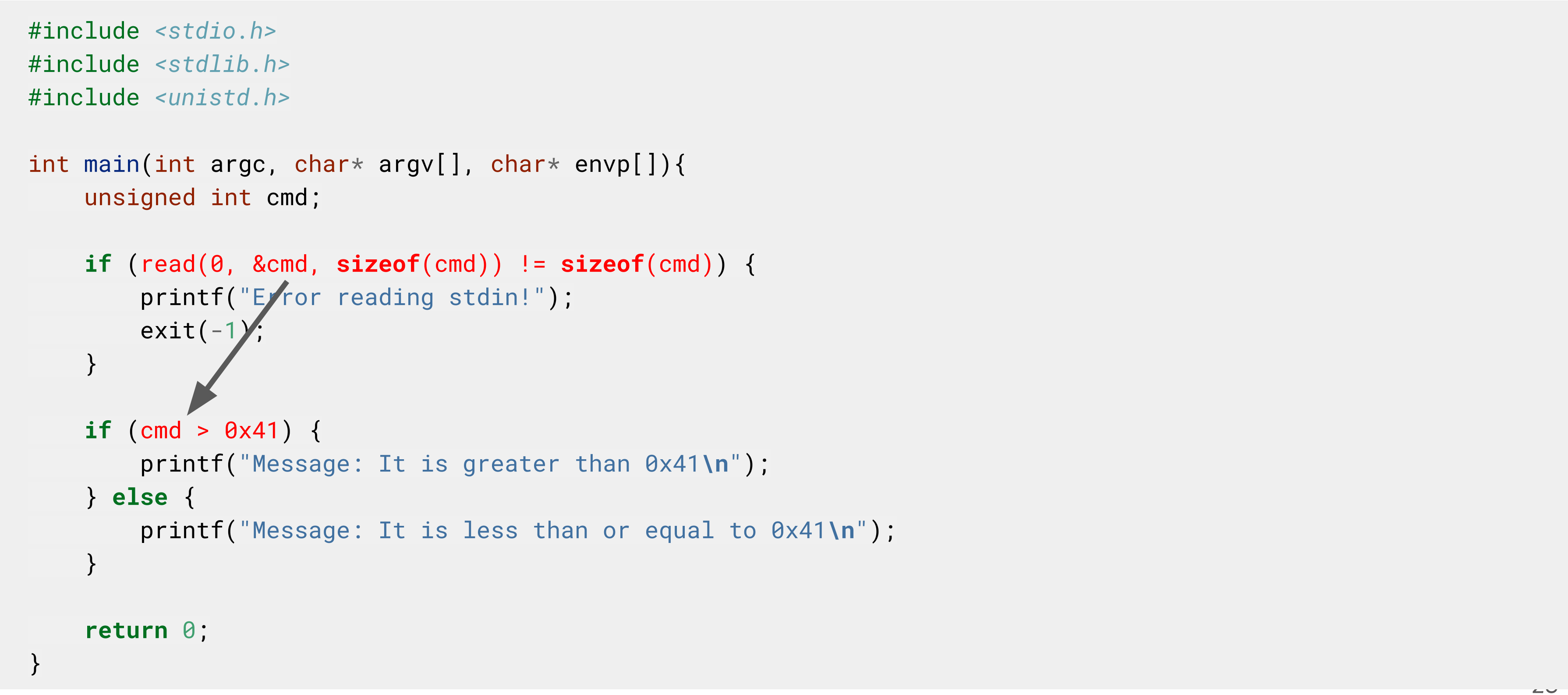
```
user@ubuntu /m/h/p/m/o/mcore_hME3MC> hexdump -C test_00000000.stdin
00000000  41 00 00 00                                       |A...|
00000004
user@ubuntu /m/h/p/m/o/mcore_hME3MC> hexdump -C test_00000001.stdin
00000000  00 80 00 20                                       |... |
00000004
```

What is Manticore?



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

What is Manticore?



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Intro to Symbolic Execution

Symbolic Execution

- **A program analysis technique that helps reason about the states a program can be in.**
- **Instead of executing a program *concretely*, it considers *all possible* executions**
- **Can answer some relevant questions**
  - What input can get me to this point in the program?
  - What values can this variable hold?
  - Can this buffer ever be accessed out-of-bounds?
  - Will this divisor ever be 0?
  - Will unmapped memory ever be accessed?

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Item	Value

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Item	Value
cmd	undef
val	0

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Item	Value
cmd	{0 <= cmd <= MAX_UINT}
val	0

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {  ←
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Item	Value
cmd	{0 <= cmd <= MAX_UINT}
val	0

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0;
}
```

Item	Value
cmd	{0 <= cmd <= 0x41}
val	0

Item	Value
cmd	{cmd > 0x41}
val	33

Symbolic Execution

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

int main(int argc, char* argv[], char* envp[]){
    unsigned int cmd, val = 0;

    if (read(0, &cmd, sizeof(cmd)) != sizeof(cmd)) {
        printf("Error reading stdin!");
        exit(-1);
    }

    if (cmd > 0x41) {
        val = printf("Message: It is greater than 0x41\n");
    } else {
        printf("Message: It is less than or equal to 0x41\n");
    }

    return 0; ←
}
```

Item	Value
cmd	{0 <= cmd <= 0x41}
val	0

Item	Value
cmd	{cmd > 0x41}
val	33

Symbolic Execution

Item	Value
`cmd`	`{0 <= cmd <= 0x41}`
`val`	`0`

Item	Value
`cmd`	`{cmd > 0x41}`
`val`	`33`

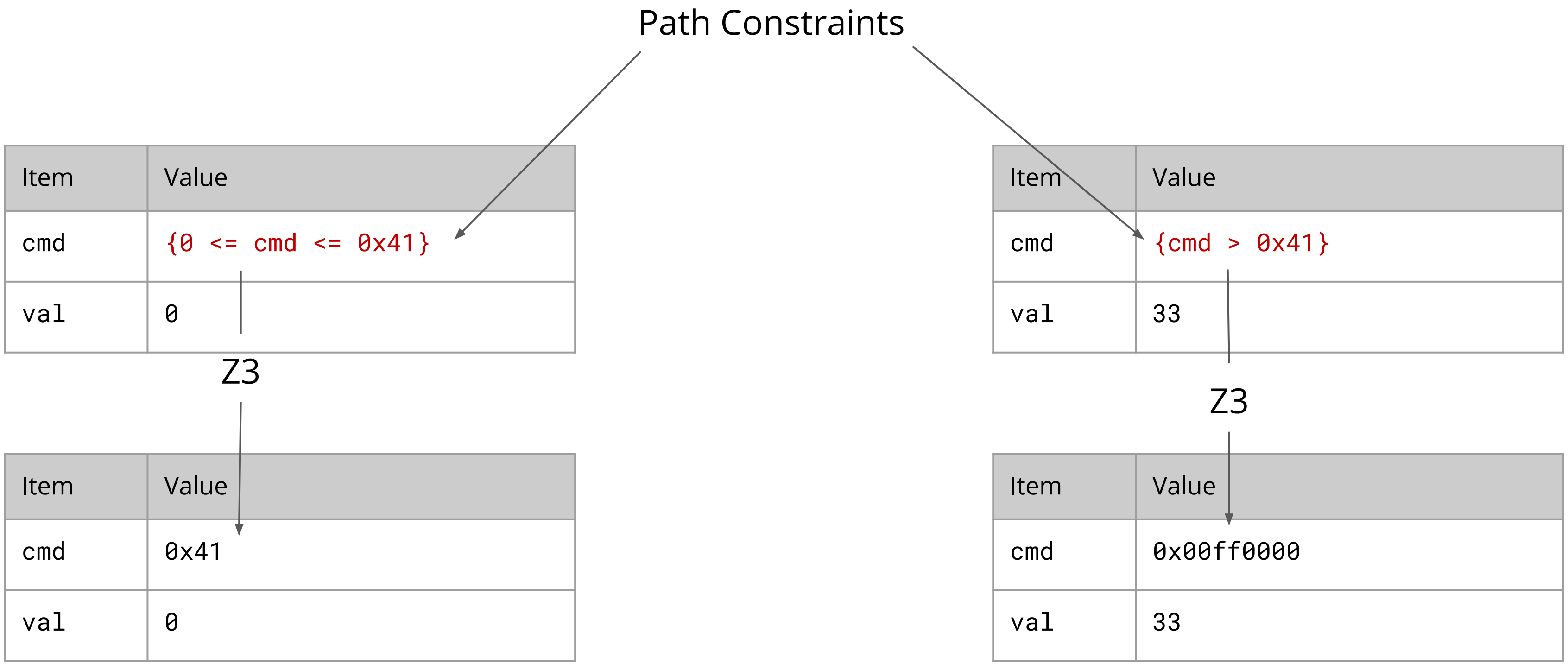
Symbolic Execution

Path Constraints

Item	Value
cmd	{0 <= cmd <= 0x41}
val	0

Item	Value
cmd	{cmd > 0x41}
val	33

Symbolic Execution



Path Constraints

Item	Value
cmd	{0 <= cmd <= 0x41}
val	0

Z3

Item	Value
cmd	0x41
val	0

Item	Value
cmd	{cmd > 0x41}
val	33

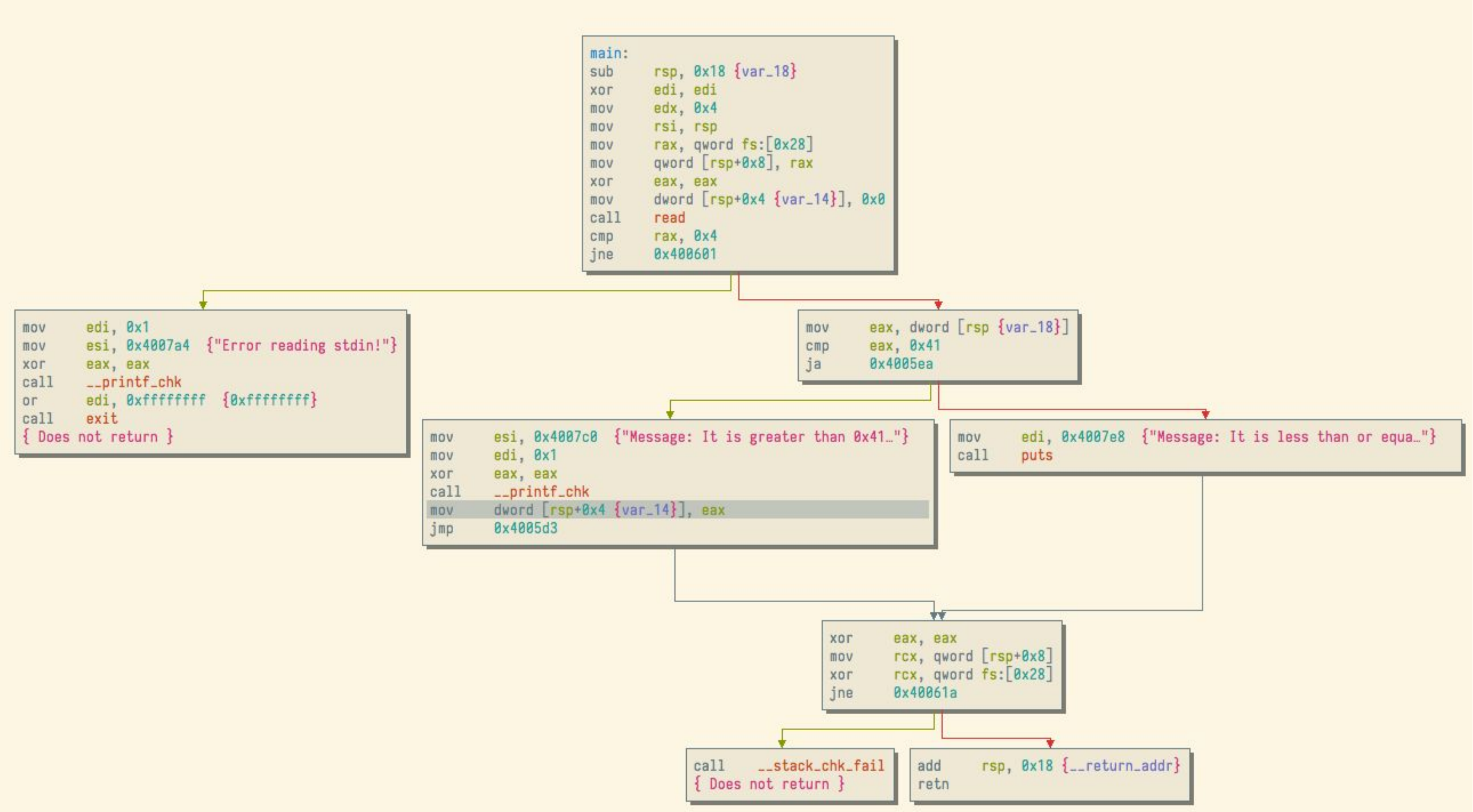
Z3

Item	Value
cmd	0x00ff0000
val	33

Symbolic Execution

- **Solver (Z3)**
  - Given a formula, determine satisfiability and solve for variables
  - Different solvers are better at different formulas
- **“Are the accumulated constraints satisfiable?”**
  - I.e. “Can we reach this point in the program?”
- **“Are the accumulated and current constraints satisfiable?”**
  - I.e. “Can this variable ever be a specific value?”
- **“Solve for a variable for current constraints”**
  - I.e. “What’s a possible value for this register/memory location”

Symbolic Execution of Native Code



```
main:
sub     rsp, 0x18 {var_18}
xor     edi, edi
mov     edx, 0x4
mov     rsi, rsp
mov     rax, qword fs:[0x28]
mov     qword [rsp+0x8], rax
xor     eax, eax
mov     dword [rsp+0x4 {var_14}], 0x0
call    read
cmp     rax, 0x4
jne     0x400601
```

```
mov     edi, 0x1
mov     esi, 0x4007a4  {"Error reading stdin!"}
xor     eax, eax
call    __printf_chk
or      edi, 0xffffffff  {0xffffffff}
call    exit
{ Does not return }
```

```
mov     eax, dword [rsp {var_18}]
cmp     eax, 0x41
ja      0x4005ea
```

```
mov     esi, 0x4007c0  {"Message: It is greater than 0x41…"}
mov     edi, 0x1
xor     eax, eax
call    __printf_chk
mov     dword [rsp+0x4 {var_14}], eax
jmp     0x4005d3
```

```
mov     edi, 0x4007e8  {"Message: It is less than or equa…"}
call    puts
```

```
xor     eax, eax
mov     rcx, qword [rsp+0x8]
xor     rcx, qword fs:[0x28]
jne     0x40061a
```

```
call    __stack_chk_fail
{ Does not return }
```

```
add     rsp, 0x18 {__return_addr}
retn
```

Symbolic Execution of Native Code

- Possible locations for data* to reside
  - Registers
  - Memory
- Must include support for symbolic expressions everywhere data can be stored
- How do we know when to fork?

Symbolic Execution

- **Caveats**
  - Quickly becomes intractable
  - Naive symbolic exploration fails at some basic structures
    - Jump tables
    - Loops
  - Determining what to analyze next is not trivial
- **Rarely used alone**
  - Typically joined with fuzzing or other concrete execution techniques
  - “Concolic” execution

Symbolic Execution

```
size_t index;
uint8_t storage[1024];
read(0, &index, sizeof(index));
storage[index] = kVisited;
```

Symbolic Execution

```
struct {
  int len;
  uint8_t *buffer;
} packet;

read(0, &packet.len, sizeof(packet.len));
for (int i = 0; i < packet.len; i++)
  read(0, &packet.buffer[i], sizeof(uint8_t));
```

Using symbolic execution support

- Introduce symbolic state by creating an expression
  - stdin
  - new_symbolic_buffer
  - new_symbolic_value
- Constrain it
- Write it to memory
- Solve for it
  - solve_n
  - solve_one

Exercise - `rand`

- Goal: Determine which value makes `verify()` succeed

Exercise - `rand`

- **Goal: Determine which value makes `verify()` succeed**
- **Hint, see:**
    - `state.new_symbolic_value()`
    - `state.solve_one()`

Exercise - hash

- Goal: Determine which argument makes `auth` print ‘Access granted!’

Exercise - hash

- Goal: Determine which argument makes `auth` print ‘Access granted!’
- Hint:
  - `state.new_symbolic_buffer()`
  - `state.constrain()`
    - `state.constrain(c > 5)`
    - `state.constrain(c < 50)`

Manticore

- **Some useful classes and methods**
  - `issymbolic`
  - State's `abandon()`
  - Cpu's `read_string()` / `write_string()`
- **Use Manticore's `shared_context()` for storing data when using multiple processes**

Exercise - heap

- Goal: Find input to trigger heap overflow to execute `win()`

“Advanced” Manticore

Manticore Plugins

- **A few notable high level classes**
  - Manticore
  - Executor
  - State
    - Cpu
    - Memory
    - Platform

Manticore Internals

- Can augment almost any aspect of Manticore’s behavior with plugins
- Various components emit events
- Plugins can implement handlers for any event
- See `ExamplePlugin`

Search Policies

- Selecting next state to explore is a non-trivial problem
- Can subclass the `Policy` class to provide your own
    - (manticore/core/executor.py)

Binary Ninja Plugins

- Ships with Binary Ninja plugins to visualize traces
- Has (very) experimental Binary Ninja IL support

Function models

- Some functions are too expensive to symbolically explore
    - Scanf, string handling functions
- Manticore provides the ability to model a function in Python
- Abstracts all ABI details
- strlen example

Hacking on Manticore

- **Extending Manticore**
    - Adding system calls
        - manticore/platforms/linux.py
    - Adding instructions, adding architectures
        - manticore/core/cpu/arm.py
    - Adding plugins
        - manticore/core/plugin.py

Other Symbolic Execution Engines

- Angr
- KLEE
- S2E
- Triton
- Many others
    - https://en.wikipedia.org/wiki/Symbolic_execution

Hacking on Manticore

- Check our project:
  - https://github.com/trailofbits/manticore
- Submit issues!
- Submit PRs!
  - Easy issues are labeled **help-wanted**
- Join our Slack!

Questions

- Questions
- Comments
- Anything?

- (Thanks)