A Dozen Years of Shellphish

From DEFCON to the Cyber Grand Challenge

Antonio Bianchi antoniob@cs.ucsb.edu

Jacopo Corbetta jacopo@cs.ucsb.edu

Andrew Dutcher dutcher@cs.ucsb.edu

Chaos Communication Congress / December 29th, 2015

Shellphish

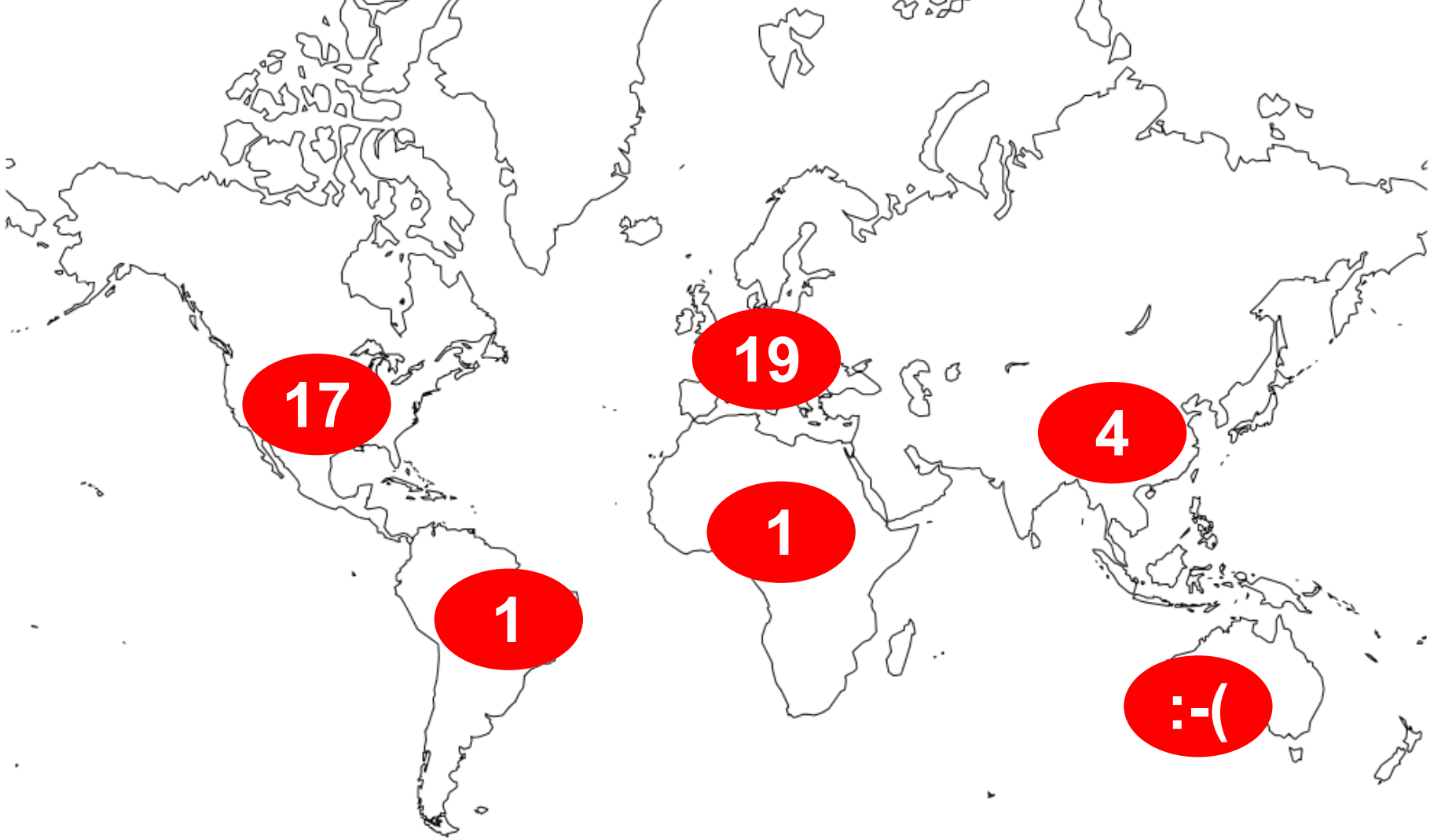
- Who are we?
  - A team of security enthusiasts
    - do research in System Security
    - play Capture the Flag competitions
    - released a couple of tools

Shellphish

- Started (in 2004) at:
  - SecLab: University of California, Santa Barbara

Shellphish

- expanded to:
  - Northeastern University: Boston
  - Eurecom: France
  - ...


17
19
4
1
1
:-(

Mini-primer: What's a CTF

- Security competition:
  - exploit a vulnerable service / website / device
  - reverse a binary
  - …
- Different formats
  - Jeopardy – Attack-Defense
  - Online – Live
  - ...
- Basic idea: find the secret, submit to organizers, ... *profit*

Shellphish

Shellphish

- We do not only play CTFs
- We also organize them!
  - UCSB iCTF
    - Attack-Defense format
    - every year, since 2002!
  - Try to innovate with a different style every year
    - Site: ictf.cs.ucsb.edu
    - Base: github.com/ucsb-seclab/ictf-framework
    - Vigna, et al., *"Ten years of ictf: The good, the bad, and the ugly."* 3GSE, 2014.

Why we're here


SHELLPHISH


SHELLFISH

Why we're here

- DARPA Cyber Grand Challenge (CGC)
  The (almost-)Million Dollar Baby
- Our Cyber Reasoning System (CRS)
  Fancy term for auto-playing a CTF
- Automated Vulnerability Discovery
- Live example using **angr**
  Open-source binary analysis framework
- Towards the Cyber Grand Challenge Finals (CFE)

Cyber Grand Challenge (CGC)

- 2004: DARPA Grand Challenge
  - Autonomous vehicles

Cyber Grand Challenge (CGC)

- 2014: DARPA **Cyber** Grand Challenge
  - Autonomous **hacking!**

Cyber Grand Challenge (CGC)

- Started in 2014
- Qualification event: June 3rd, 2015, online
  - ~70 teams → 7 qualified teams
- Final event: August 4th, 2016 @ DEFCON (Las Vegas)
  - Winning CRS will also play against humans!

cybergrandchallenge.com / cgc.darpa.mil

CGC Rules

- Attack-Defense CTF
- ~~Solving security challenges~~ → Developing a system that solves security challenges
- Develop a system that **automatically**
  - Exploit vulnerabilities in binaries
  - Patch binaries, removing the vulnerabilities
- No human intervention
- Think it's trivial? How would you play?
  - And how would you organize it?

CGC – Rules

- Exploits
  - “getting a flag” (how? where?)
  - For the quals: exploit = **crash**
- Defend
  - int main() { return 0; }
    - Functionality checks
  - SIGSEGV => exit(0)
    - No easy “out-of-band” error handling
  - QEMU-style interpreter: interrupts => exit(0)
    - Performance cost (CPU, memory, file size)

CGC Qualification Event – Rules

- Basic idea:
  - Real(istic) programs
  - No “extra” complications
    - Is modeling the entire POSIX API a good use of team resources? What about the file systems? Or horrible things like interruptible syscalls?

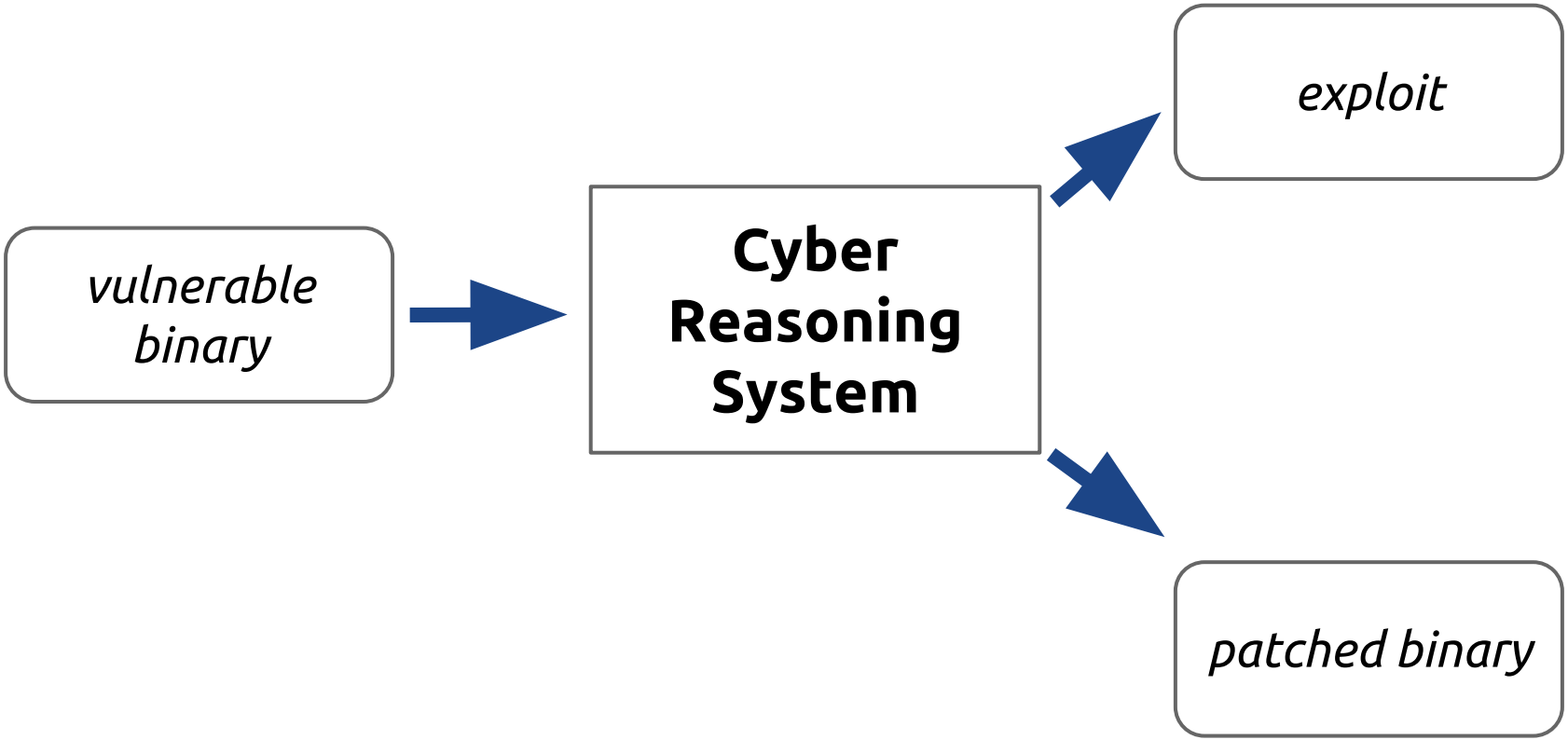
CGC Qualification Event – Rules

- Architecture: Intel x86, 32-bit
- OS: DECREE
  - Linux-like, but with 7 syscalls only
    - `transmit / receive / fdwait (≈select)`
    - `allocate / deallocate` (even executable!)
    - `random`
    - `_terminate`
  - no signals, threads, shared memory
- “Bring Your Own Defense” approach (and pay for it)
  - Not even “the usual”: stack is executable, no ASLR, …

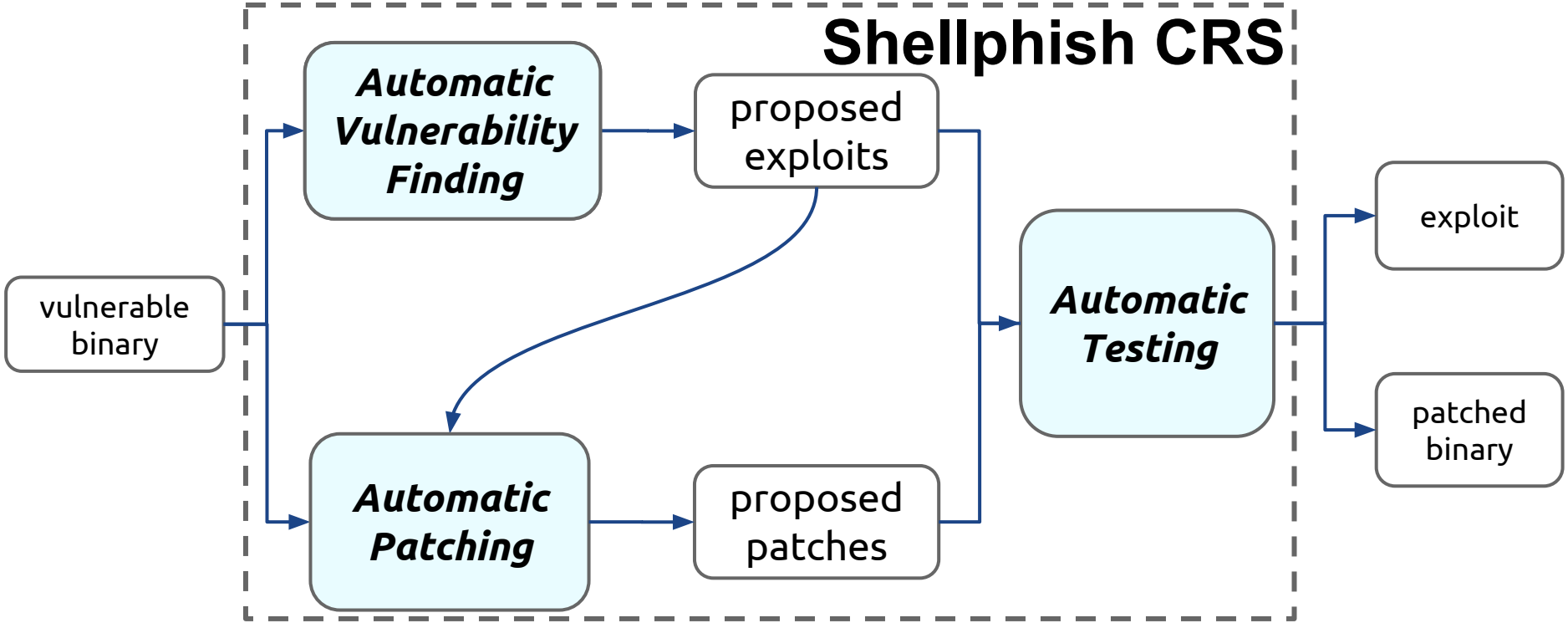
Coming up:

- DARPA Cyber Grand Challenge (CGC)
  The (almost-)Million Dollar Baby
- **Our Cyber Reasoning System (CRS)**
  Fancy term for auto-playing a CTF
- Automated Vulnerability Discovery
- Live example using **angr**
  Open-source binary analysis framework
- Towards the Cyber Grand Challenge Finals (CFE)

Shellphish CRS



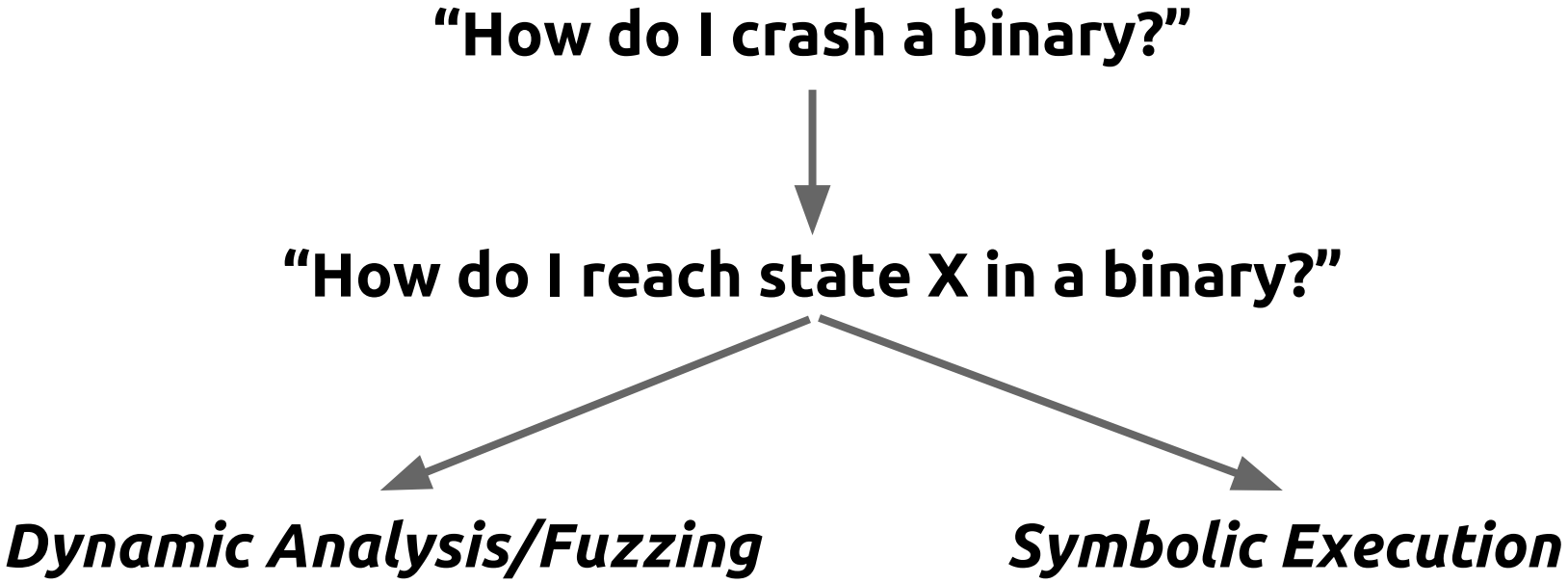
Shellphish CRS



Why we're here

- DARPA Cyber Grand Challenge (CGC)
  The (almost-)Million Dollar Baby
- Our Cyber Reasoning System (CRS)
  Fancy term for auto-playing a CTF
- **Automated Vulnerability Discovery**
- Live example using **angr**
  Open-source binary analysis framework
- Towards the Cyber Grand Challenge Finals (CFE)

Automatic Vulnerability Discovery



Dynamic Analysis/Fuzzing

- **How do I reach the state: “You win!” is printed?**

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

Dynamic Analysis/Fuzzing

- **How do I reach the state: “You win!” is printed?**

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

- Try “1” → *“You lose!”*
- Try “2” → *“You lose!”*
- …
- Try “10” → *“You win!”*

Dynamic Analysis/Fuzzing

- How did we use Fuzzing for CGC?
- Coverage-guided fuzzing
  - Looking for “crashing inputs”
  - Based on AFL lcamtuf.coredump.cx/afl/
- In general, it cannot work in some cases
  - e.g., “magic numbers”, computations, ...

Dynamic Analysis/Fuzzing

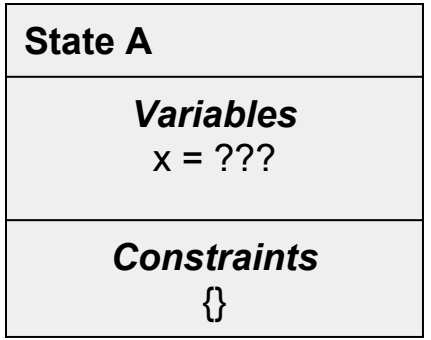
- **How do I reach the state: “You win!” is printed?**

```
x = int(input())
if x >= 10:
   if x^2 == 152399025:
      print "You win!"
   else:
      print "You lose!"
else:
   print "You lose!"
```

Symbolic Execution

- Interpret the binary code and replace user-input with symbolic variables

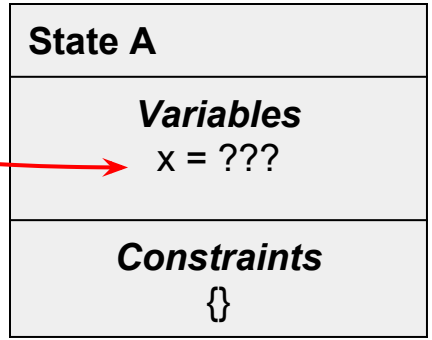
```
x = int(input())
if x >= 10:
   if x < 100:
      print "You win!"
   else:
      print "You lose!"
else:
   print "You lose!"
```



Symbolic Execution

- Interpret the binary code and replace user-input with symbolic variables

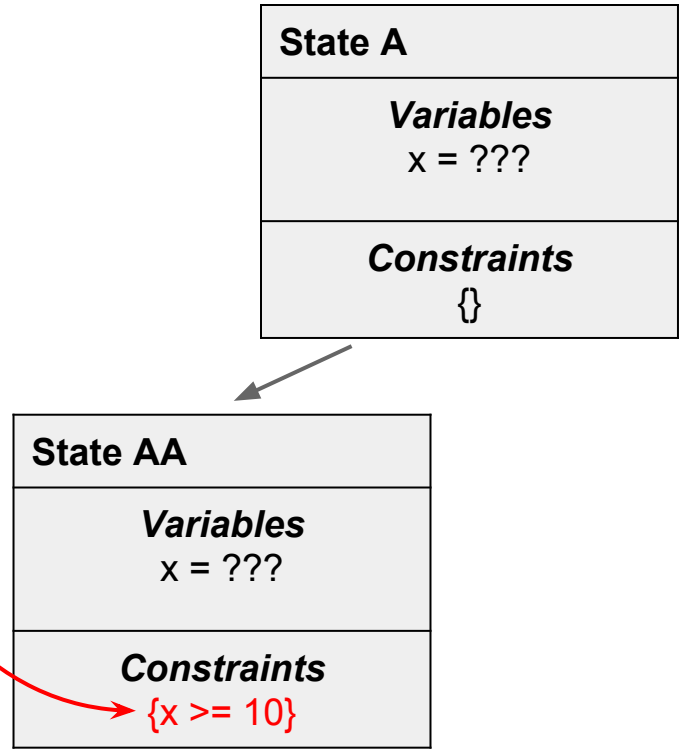
```
x = int(input())
if x >= 10:
   if x < 100:
      print "You win!"
   else:
      print "You lose!"
else:
   print "You lose!"
```



Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

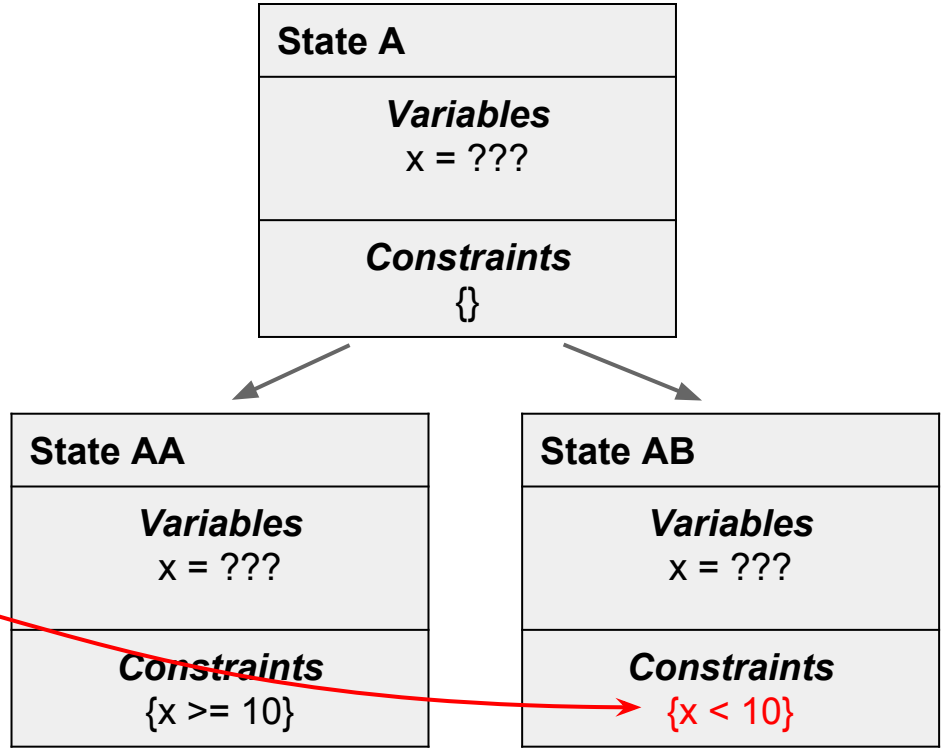
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

State AA
Variables x = ???
Constraints {x >= 10}

State AB
Variables x = ???
Constraints {x < 10}

Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

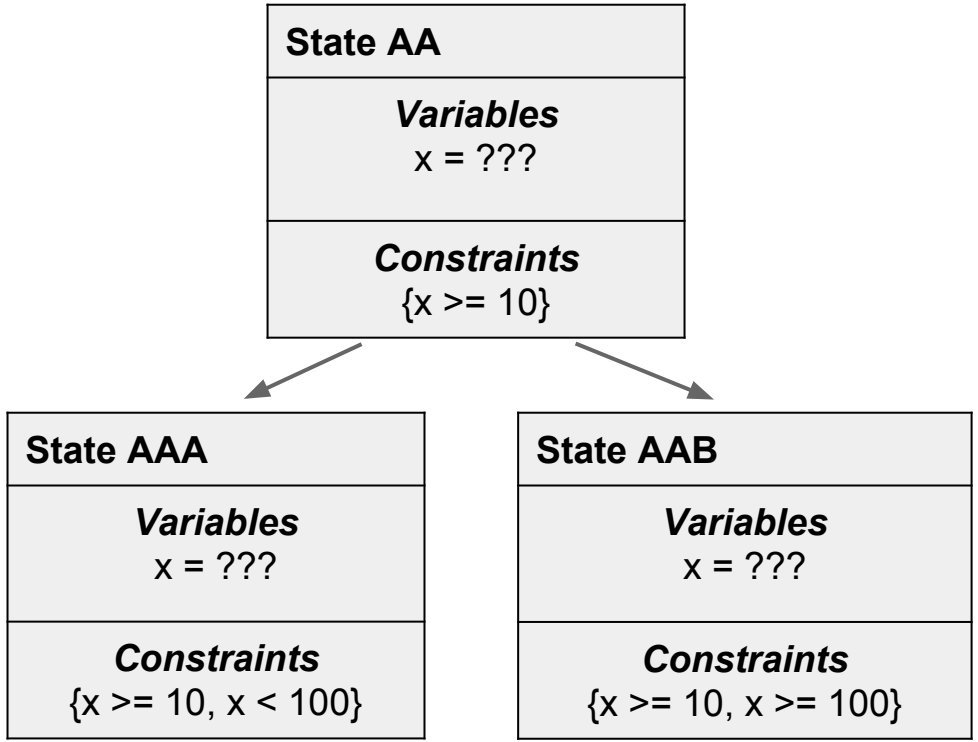
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```

State AA
Variables x = ???
Constraints {x >= 10}

Symbolic Execution

- Follow all feasible paths, tracking "constraints" on variables

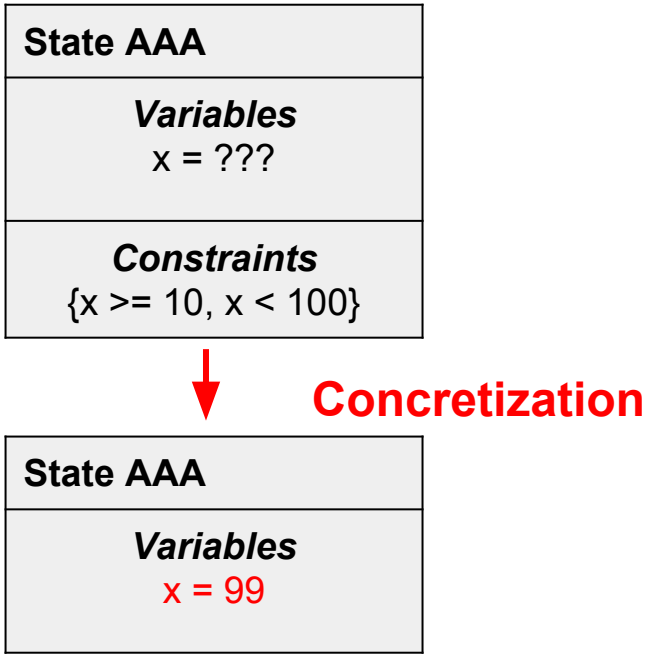
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Symbolic Execution

- Concretize the constraints on the symbolic variables

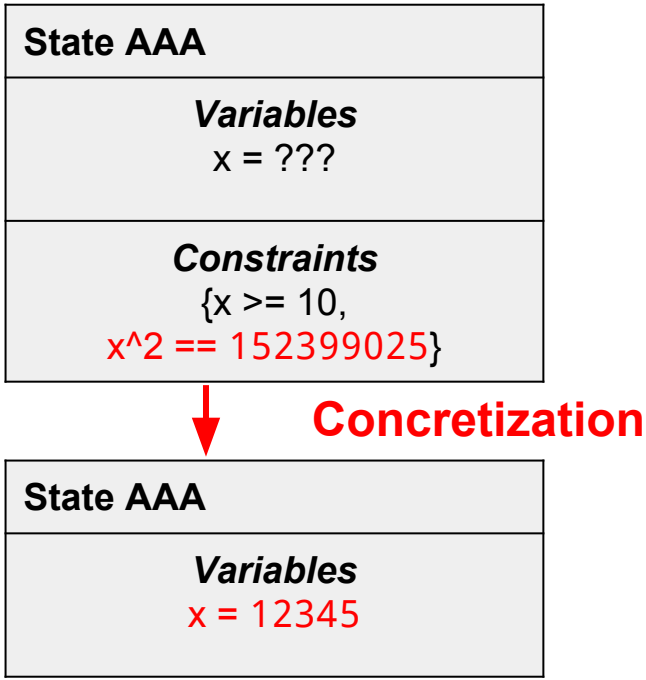
```
x = int(input())
if x >= 10:
    if x < 100:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Symbolic Execution

- Concretize the constraints on the symbolic variables

```
x = int(input())
if x >= 10:
    if x^2 == 152399025:
        print "You win!"
    else:
        print "You lose!"
else:
    print "You lose!"
```



Symbolic Execution

- How did we use Symbolic Execution for CGC?
  - We used the symbolic execution engine of **angr,** the binary analysis platform developed at UCSB

- Symbolically execute the binaries looking for
  1. Memory accesses outside allocated regions
  2. “Unconstrained” instruction pointer (e.g., controlled by user input)
     - `eax = <user input>`, `jmp eax`
- If either 1. or 2. is true
  → we found an input that will make the program crash

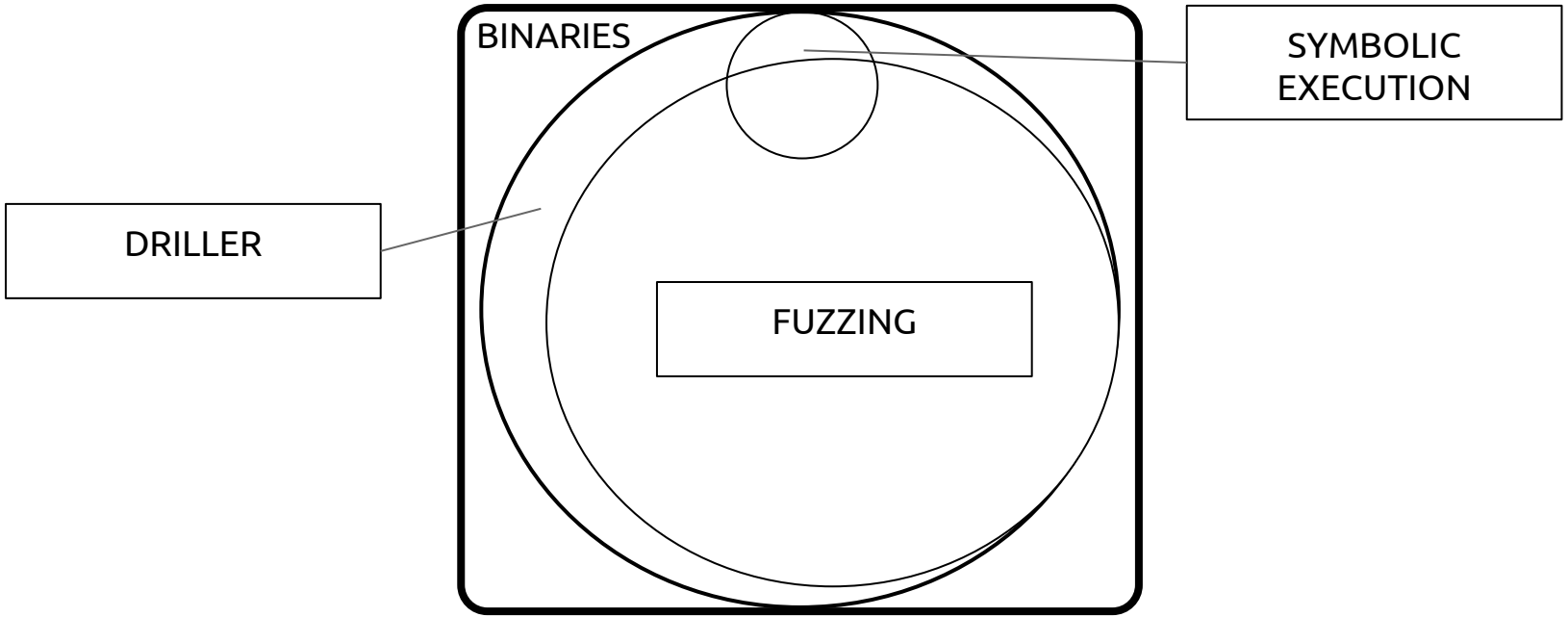
Future directions

- Combining the two approaches

- *“Driller: Augmenting Fuzzing Through Selective Symbolic Execution”*
  - Network and Distributed System Security Symposium (NDSS), February 2016, San Diego

Future directions

- *"Driller: Augmenting Fuzzing Through Selective Symbolic Execution"*



Coming up:

- DARPA Cyber Grand Challenge (CGC)
  The (almost-)Million Dollar Baby
- Our Cyber Reasoning System (CRS)
  Fancy term for auto-playing a CTF
- Automated Vulnerability Discovery
- **Live example using angr**
  Open-source binary analysis framework
- Towards the Cyber Grand Challenge Finals (CFE)

angr

- Binary analysis platform, developed at UCSB
- Open source: github.com/angr (star it!)
- Architecture independent
  - x86 (ELF, **CGC**, PE), amd64, mips, mips64, arm, aarch64, ppc, ppc64

angr

- Written in Python!
  - installable with one (two?) command!
    - `mkvirtualenv angr`
    - `pip install angr`

(optional, but don't complain if it's broken)

  - interactive shell (using IPython)
  - it has an interactive GUI

angr – Demonstration

Grub: “Back to 28” vulnerability

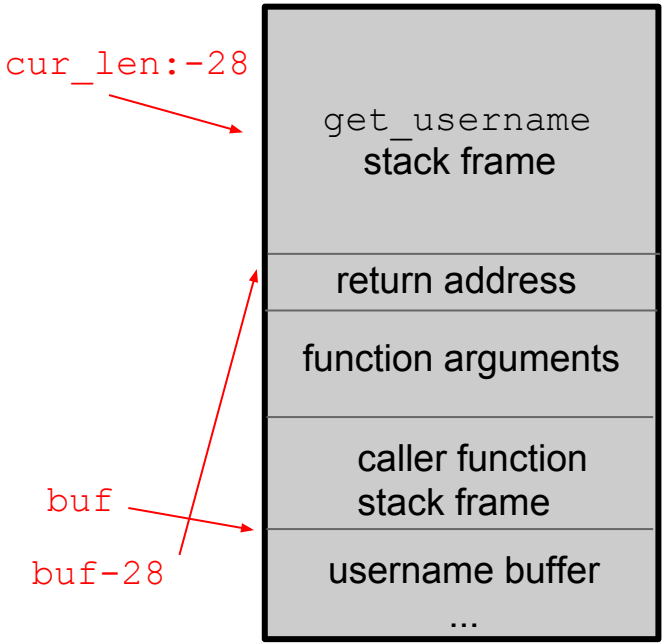
- Pressing backspace 28 times on the grub username prompt can get you a rescue shell
- http://hmarco.org/bugs/CVE-2015-8370-Grub2-authentication-bypass.html

```
if (key == '\b')  // Does not checks underflows !!
  {
    cur_len--; // Integer underflow !!
    grub_printf ("\b");
    continue;
  }
```

angr – Demonstration

Grub: “Back to 28” vulnerability

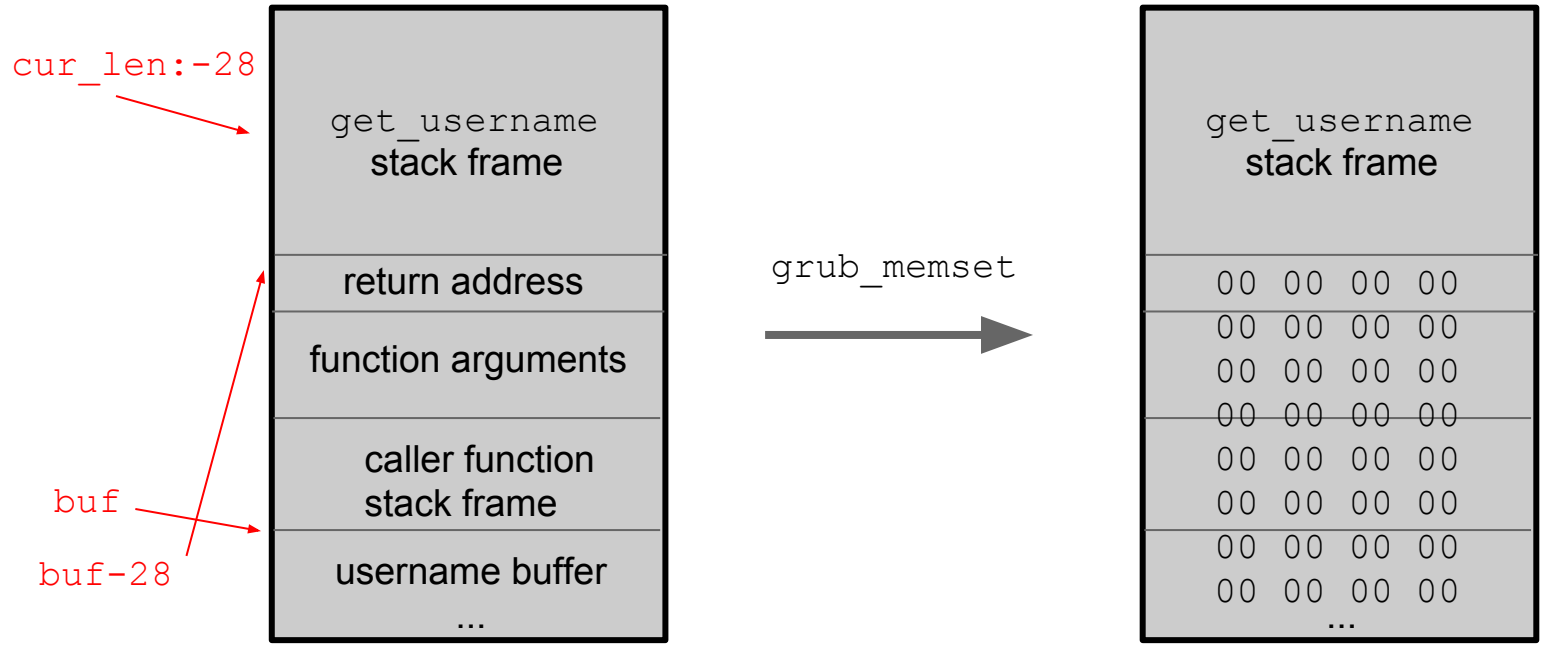
```
grub_memset( buf + cur_len, 0, buf_size - cur_len); // Out of bounds overwrite
```



angr – Demonstration

Grub: “Back to 28” vulnerability

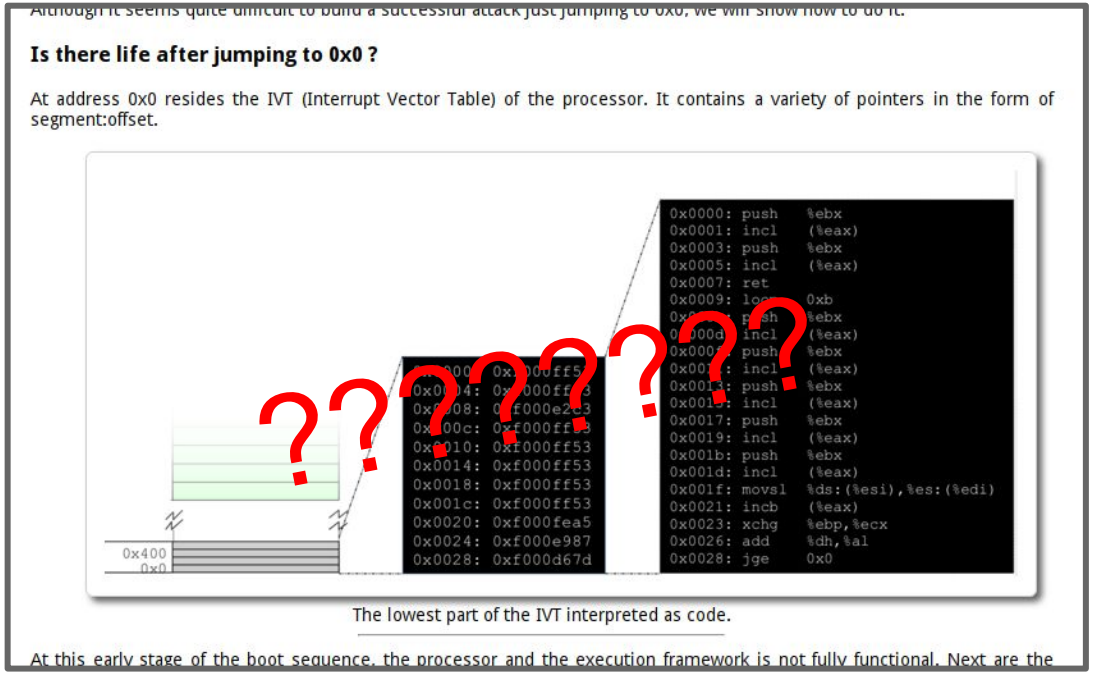
```
grub_memset( buf + cur_len, 0, buf_size - cur_len); // Out of bounds overwrite
```



angr – Demonstration

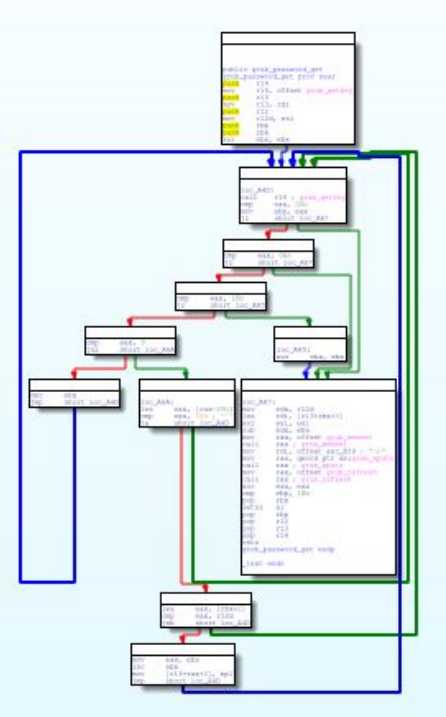
Grub: “Back to 28” vulnerability

- Somehow, jumping to `00:0000` is completely exploitable
- *Way* beyond the scope of this demo



angr – Demonstration

Grub: “Back to 28” vulnerability



- The correct path to the exploit goes around this loop 28 times, each of which has to follow a specific path
- The universe will grow old and die before naive symbolic execution finds this bug
- **Demonstration: this doesn’t work, really!**
- A technique (implemented by angr) called veritesting[1] solves this problem in *some cases* by merging states when their instruction pointers converge, but in *this case* the complexity generated is too much for the constraint solver

[1] http://users.ece.cmu.edu/~aavgerin/papers/veritesting-icse-2014.pdf

angr – Demonstration

- Symbolic execution is powerful
- Symbolic execution is stupid

- You are incredibly weak
- You are very clever

Use angr to unlock your true potential

angr – Demonstration

Grub: “Back to 28” vulnerability

Manual examination of the state explosion tells you:

- Where the wasted computational power is going
- How to be more efficient

The naive approach is doing lots of weird things like entering letters and then deleting them again and again, or pressing the “home” key several times in a row, which don’t produce any interesting new states to analyze.

You can fix this!

Grub: “Back to 28” vulnerability

Final demonstration

Finding the bug

Coming up:

- DARPA Cyber Grand Challenge (CGC)
  The (almost-)Million Dollar Baby
- Our Cyber Reasoning System (CRS)
  Fancy term for auto-playing a CTF
- Automated Vulnerability Discovery
- Live example using **angr**
  Open-source binary analysis framework
- **Towards the Cyber Grand Challenge Finals (CFE)**

CGC Quals – Results

- 7 teams passed the qualification phase
- Shellphish is one of them! :-)
- We exploited 44 binaries out of 131
- Every qualified team received $750,000 !

CGC Finals

- Different setup
  - Round-based attack-defense CTF
  - Probably, zero human intervention allowed
    - Not even bug fixing?
  - Data about previous rounds is available:
    - submitted exploits/patched binaries performance
    - (anonymized) network traffic
    - patches from other teams
  - Stealing patched binaries/exploits?

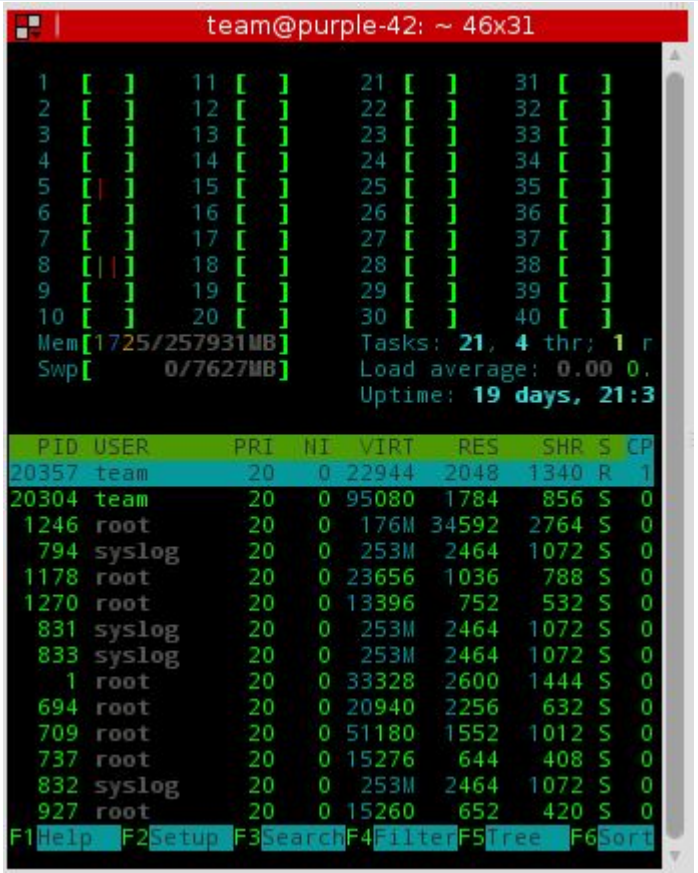
CGC Finals

- Exploits are more realistic:
  - Two types:
    - Crash at a specific location and set a specific register to a specific value
    - Leak data from a specific memory page
  - We'll need a more realistic exploit generator:
    - **angr** automatic ROP-chain builder!
- Every team can also deploy network-level filtering rules

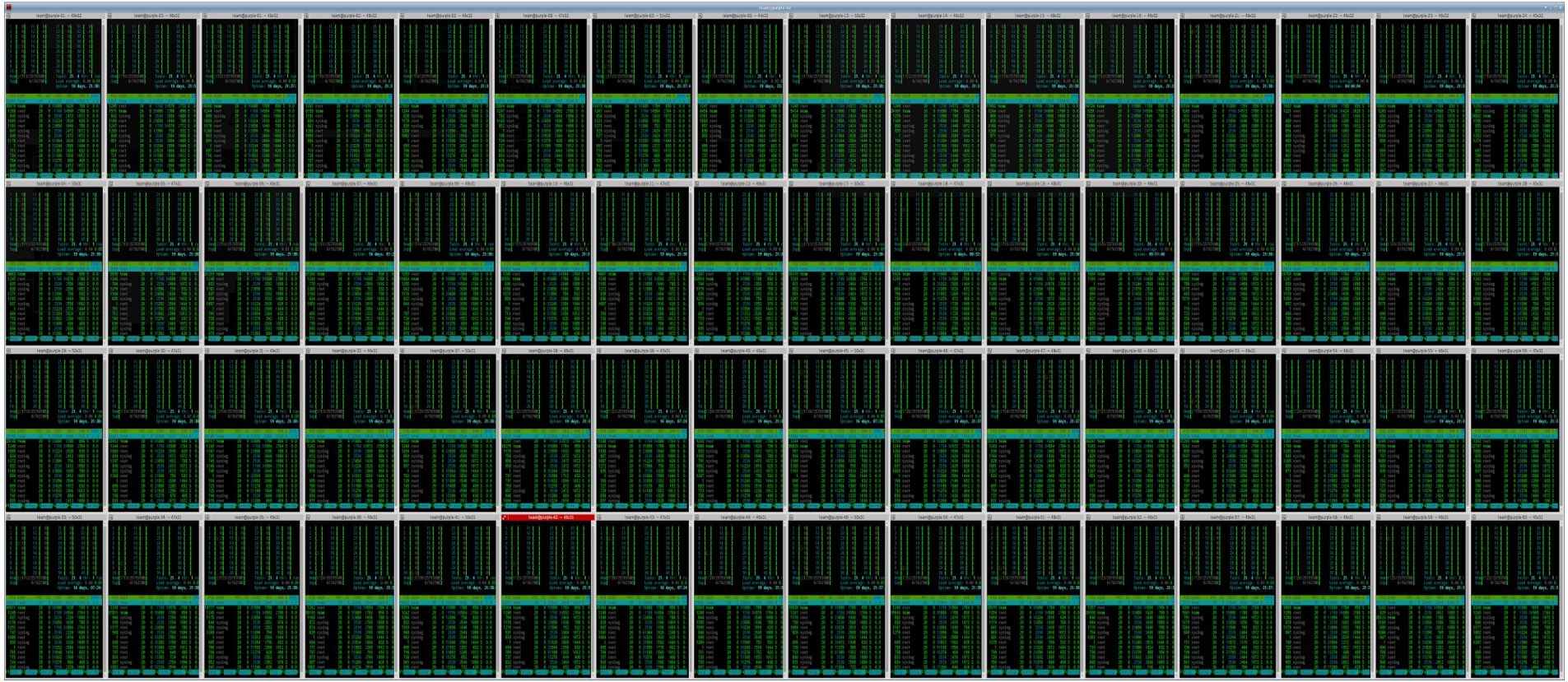
CGC Finals

- Every team has access to a cluster of:
  - 1280 cores
  - 16 TB of RAM
  - 128 TB of storage

CGC Finals


team@purple-42: ~ 46x31
1 [ ] 11 [ ] 21 [ ] 31 [ ]
2 [ ] 12 [ ] 22 [ ] 32 [ ]
3 [ ] 13 [ ] 23 [ ] 33 [ ]
4 [ ] 14 [ ] 24 [ ] 34 [ ]
5 [|] 15 [ ] 25 [ ] 35 [ ]
6 [ ] 16 [ ] 26 [ ] 36 [ ]
7 [ ] 17 [ ] 27 [ ] 37 [ ]
8 [||] 18 [ ] 28 [ ] 38 [ ]
9 [ ] 19 [ ] 29 [ ] 39 [ ]
10 [ ] 20 [ ] 30 [ ] 40 [ ]
Mem[1725/257931MB]
Swp[0/7627MB]
Tasks: 21, 4 thr; 1 r
Load average: 0.00 0.
Uptime: 19 days, 21:3
PID USER PRI NI VIRT RES SHR S CP
20357 team 20 0 22944 2048 1340 R 1
20304 team 20 0 95080 1784 856 S 0
1246 root 20 0 176M 34592 2764 S 0
794 syslog 20 0 253M 2464 1072 S 0
1178 root 20 0 23656 1036 788 S 0
1270 root 20 0 13396 752 532 S 0
831 syslog 20 0 253M 2464 1072 S 0
833 syslog 20 0 253M 2464 1072 S 0
1 root 20 0 33328 2600 1444 S 0
694 root 20 0 20940 2256 632 S 0
709 root 20 0 51180 1552 1012 S 0
737 root 20 0 15276 644 408 S 0
832 syslog 20 0 253M 2464 1072 S 0
927 root 20 0 15260 652 420 S 0
F1Help F2Setup F3Search F4Filter F5Tree F6Sort

CGC Finals



CGC Finals

- Finals will take place on August 2016
  - DEFCON, Las Vegas

- Money prices!
  - First place: $2’000’000
  - Second place: $1’000’000
  - Third place: $750’000

- The winning team will compete against human teams at DEFCON CTF Finals :-)

Shellphish CGC Team

I want more...

- **angr** hands-on workshop
  - Just after this talk
  - Hall 13 (first floor)
  - Bring your laptop!

“That’s all folks!”

Questions?

References:
CGC: cybergrandchallenge.org – cgc.darpa.mil
DARPA CGC presentation (DEFCON 2015): youtu.be/gnyCbU7jGYA
angr: angr.io – github.com/angr
emails: antoniob@cs.ucsb.edu – jacopo@cs.ucsb.edu – dutcher@cs.ucsb.edu