Angry Hacking

How angr pwned CTFs and the CGC

Zardus

rhelmot

salls

Fish

nezorg

kereoz

Motivation **6 mins**

Fundamentals of angr **3 mins**

- Pure awesomeness

Live demos **20 mins**

- Symbolic execution
- Static analysis
- Emulation

angr applications **10 mins**

- Rop gadget finder
- Binary diffing
- Cyber Grand Challenge

Open source! **3 minutes**

- http://angr.io
- Credits

Why angr?

BAP

CodeReason

radare2

rdis

amoco

SemTrax

BitBlaze

BARF

insight

Triton

Bindead

PySysEmu

miasm

paimei

2005 Hex-Rays was founded

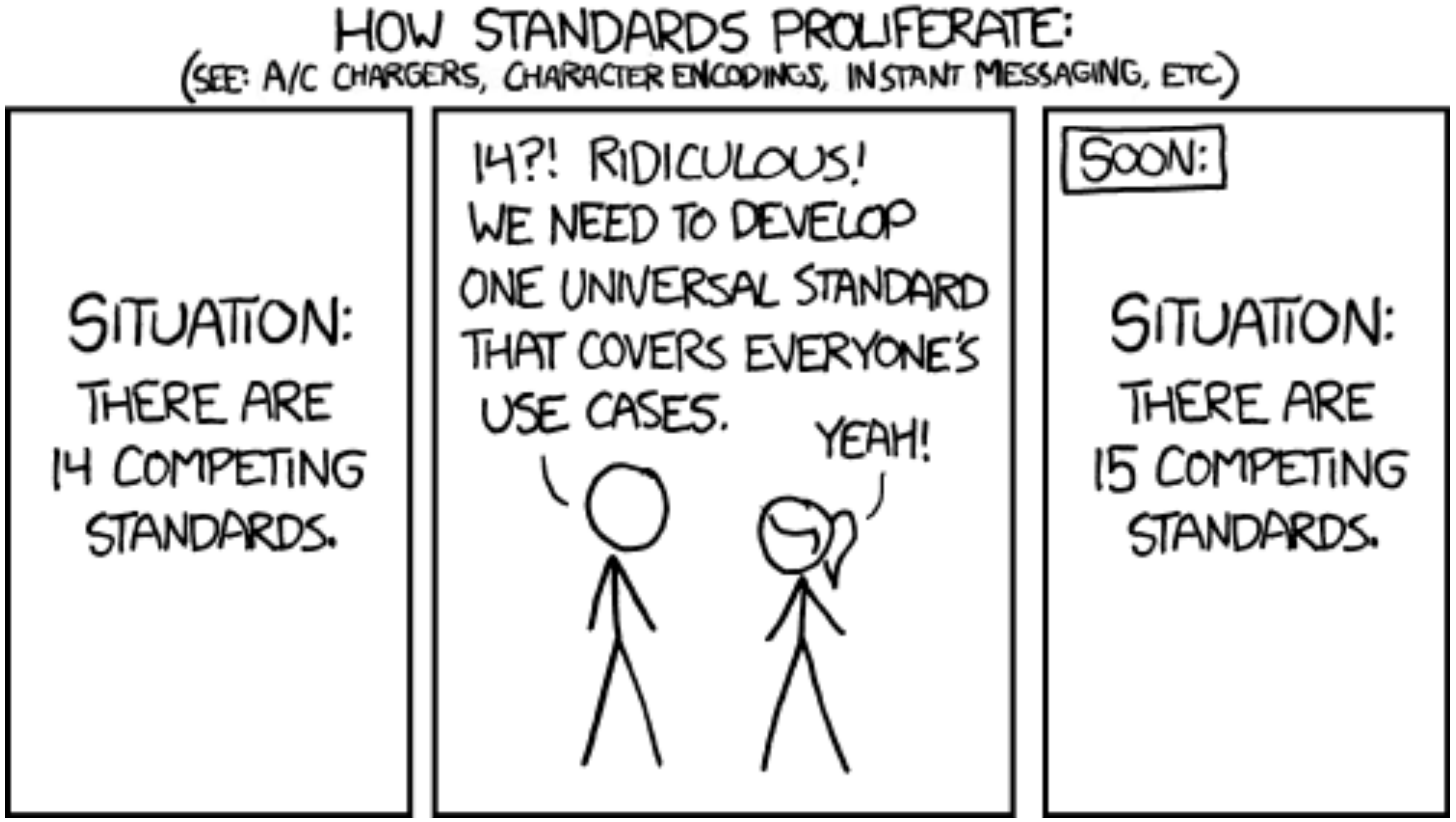
2007 Hex-Rays Decompiler 1.0

2009 Hex-Rays IDA 5.5

2011 Hex-Rays IDA 6.1

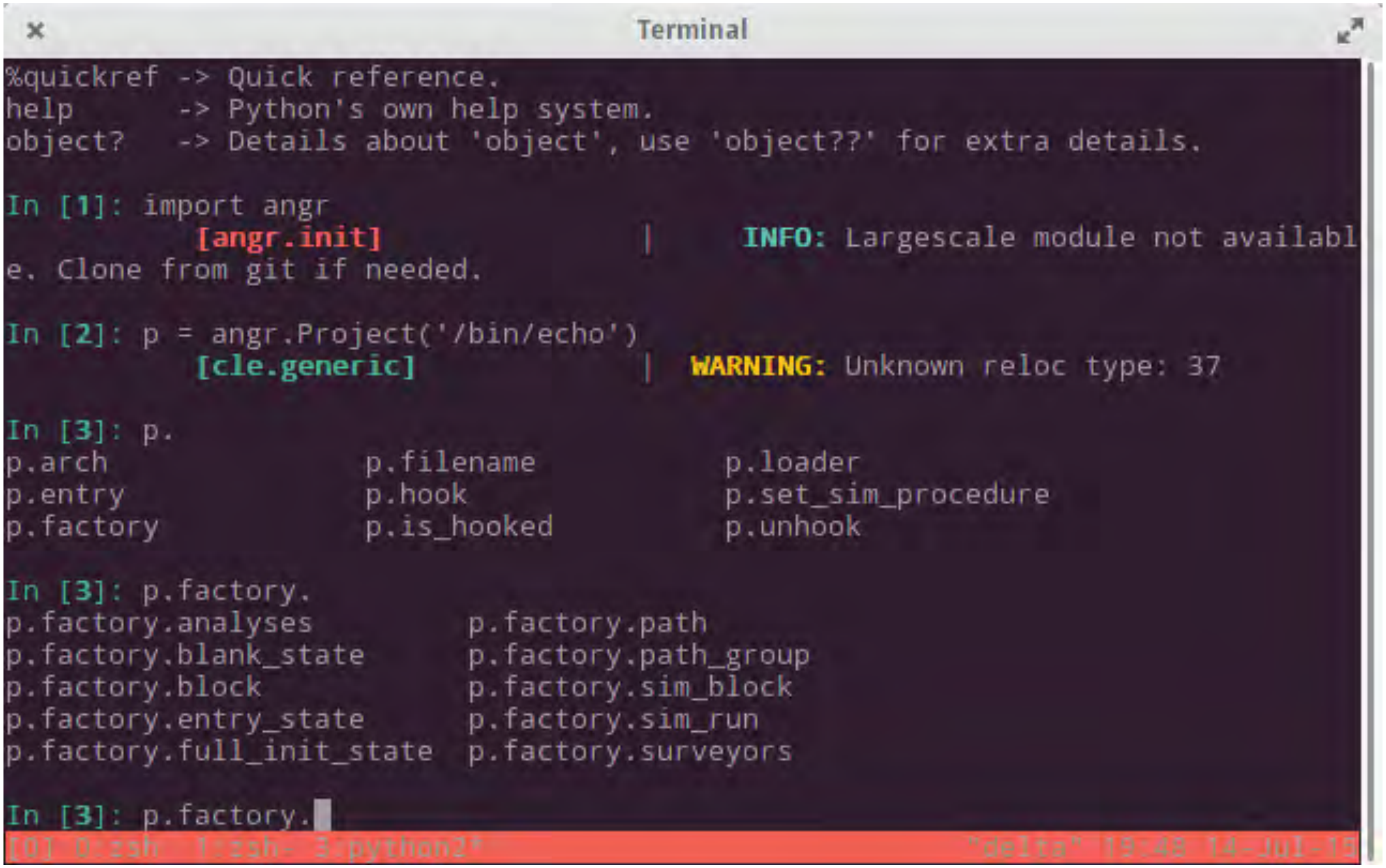
2013 Hex-Rays IDA 6.4

2015 ???


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Fundamentals of angr

- iPython-accessible
- powerful analyses
- versatile
- well-encapsulated
- open and expandable
- architecture "independent"



```
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import angr
           [angr.init]                  |      INFO: Largescale module not availabl
e. Clone from git if needed.

In [2]: p = angr.Project('/bin/echo')
           [cle.generic]                |   WARNING: Unknown reloc type: 37

In [3]: p.
p.arch                p.filename            p.loader
p.entry               p.hook                p.set_sim_procedure
p.factory             p.is_hooked           p.unhook

In [3]: p.factory.
p.factory.analyses          p.factory.path
p.factory.blank_state       p.factory.path_group
p.factory.block             p.factory.sim_block
p.factory.entry_state       p.factory.sim_run
p.factory.full_init_state   p.factory.surveyors

In [3]: p.factory.
[0] 0:zsh  1:zsh- 3:python2*                                   "delta" 19:48 14-Jul-15
```

- angr
  - Binary Loader
  - Static Analysis Routines
    - Control-Flow Graph
    - Data-Flow Analysis
    - Value-Set Analysis
  - Symbolic Execution Engine

ARE YOU READY FOR THE ANGRY POWER?

Victim binary

Demo

Symbolic execution

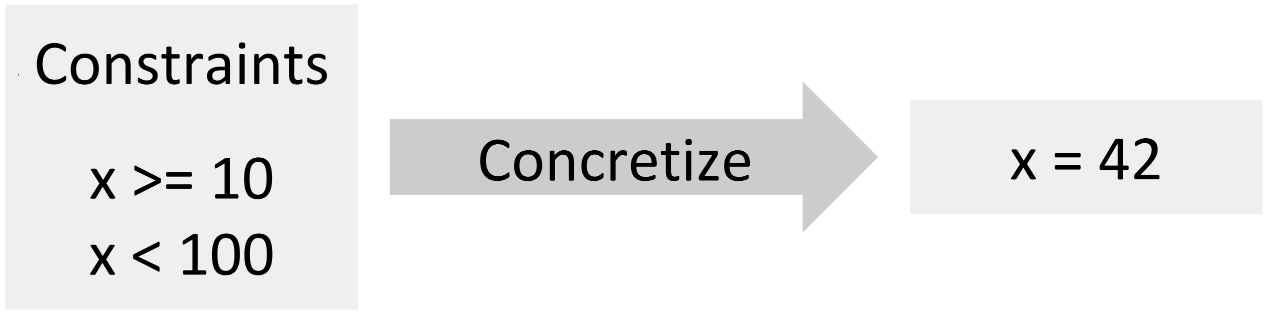
"How do I trigger path X or condition Y?"

- ❑ Dynamic analysis
  - ❑ Input A? No. Input B? No. Input C? …
  - ❑ Based on concrete inputs to application.
- ❑ (Concrete) static analysis
  - ❑ "You can't"/"You might be able to"
  - ❑ Based on various static techniques.

We need something slightly different.

"How do I trigger path X or condition Y?"

1. Interpret the application.
2. Track "constraints" on variables.
3. When the required condition is triggered, "concretize" to obtain a possible input.



Constraint solving:

- Conversion from set of constraints to set of concrete values that satisfy them.
- NP-complete, in general.

```
x = int(input())
if x >= 10:
  if x < 100:
    print "Two!"
  else:
    print "Lots!"
else:
  print "One!"
```

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```

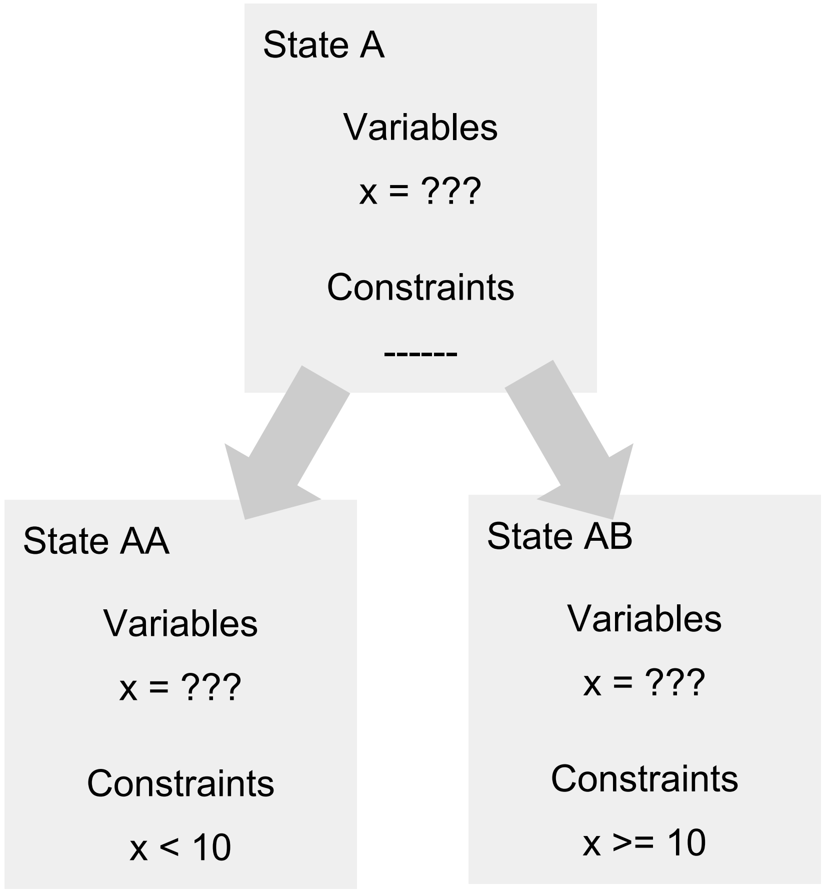
State A

Variables

x = ???

Constraints

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```

State AA

Variables

x = ???

Constraints

x < 10

State AB

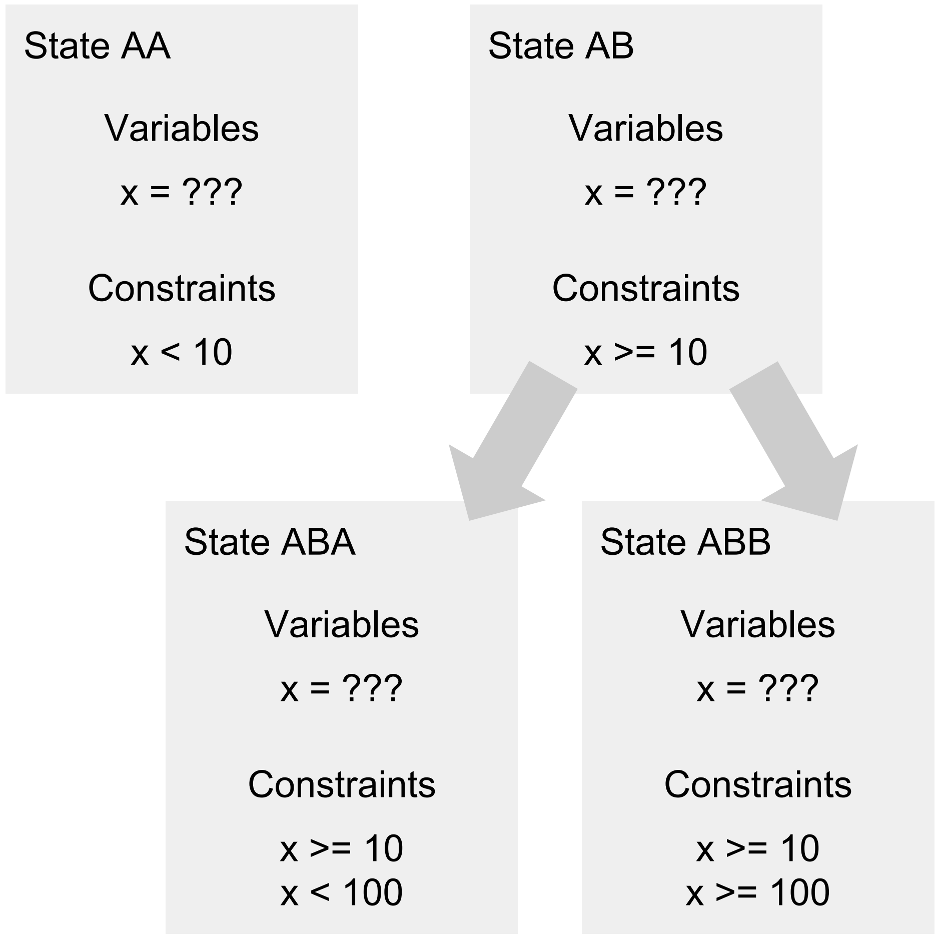
Variables

x = ???

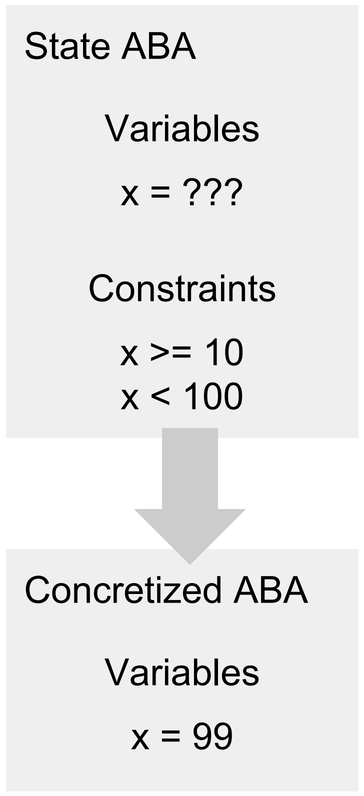
Constraints

x >= 10

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



Demo

Static analysis

Memory access checks *Type inference*

Variable recovery **Range recovery**

Wrapped-interval analysis

Value-set analysis

Abstract interpretation

What a value-set looks like

```
{
    ( global,              (4[0x601000, 0x602000],  32) ),
    ( stack_0x400957, (8[-0xc, -0x4],               32) )
}
```

global	stack_0x400957
...	...
0x601000, 0x601004	- 0xc
0x601008, 0x60100c	- 0x4
...	...

DEMO
TIME

angr applications

ROP gadget finder

Binary diffing

Cyber Grand Challenge

POV

exploit

CB

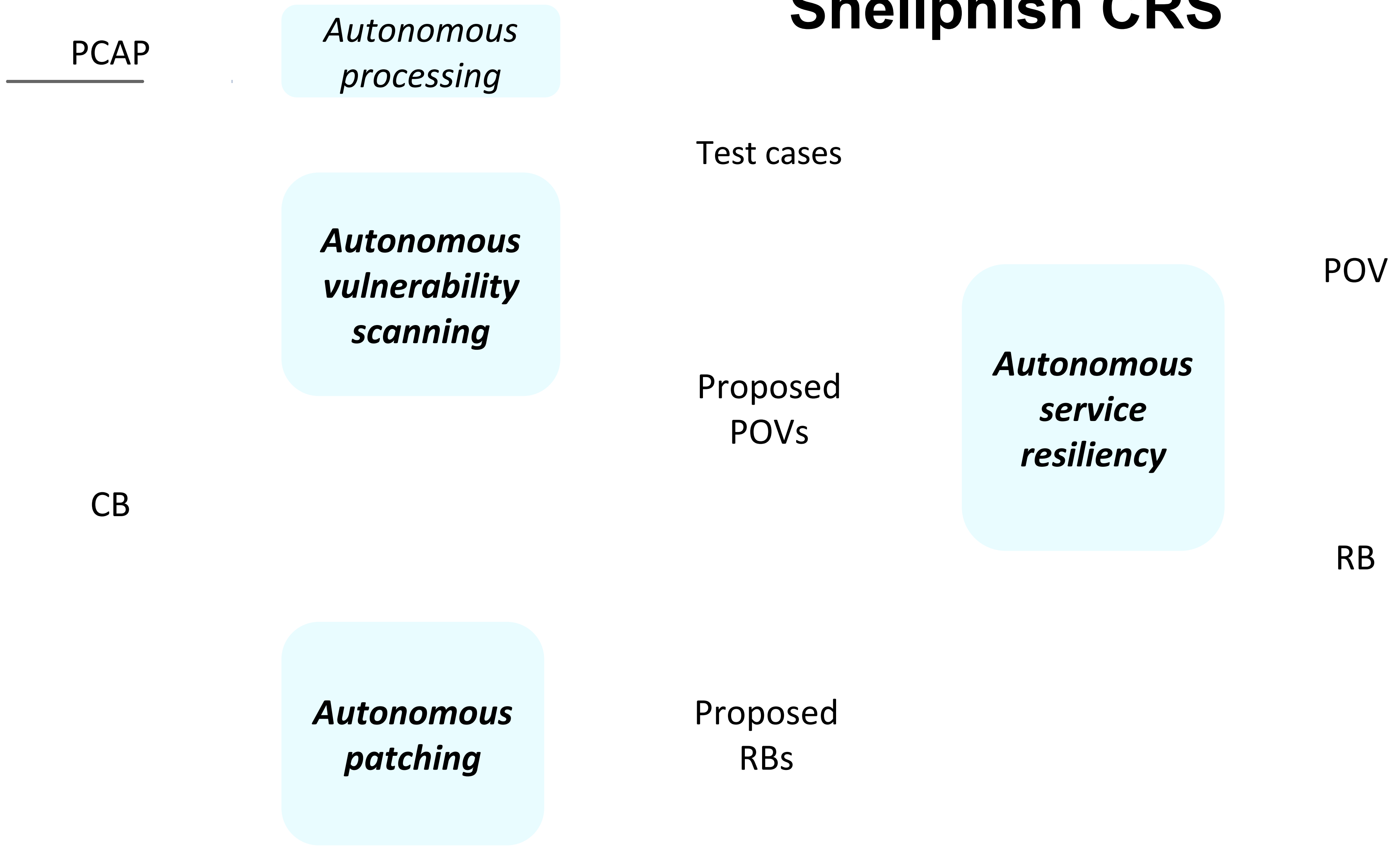
vulnerable program

Cyber Reasoning System

RB

patched program

Shellphish CRS


PCAP
Autonomous processing
Test cases
Autonomous vulnerability scanning
POV
Proposed POVs
Autonomous service resiliency
CB
RB
Autonomous patching
Proposed RBs

Vulnerability Discovery via SymExec

Program

Symbolic inputs

Security policies

Symbolic execution engine

Security policy checker

POVs

KEEP
CALM
ITS TIME FOR
A
DEMO!

Open Source

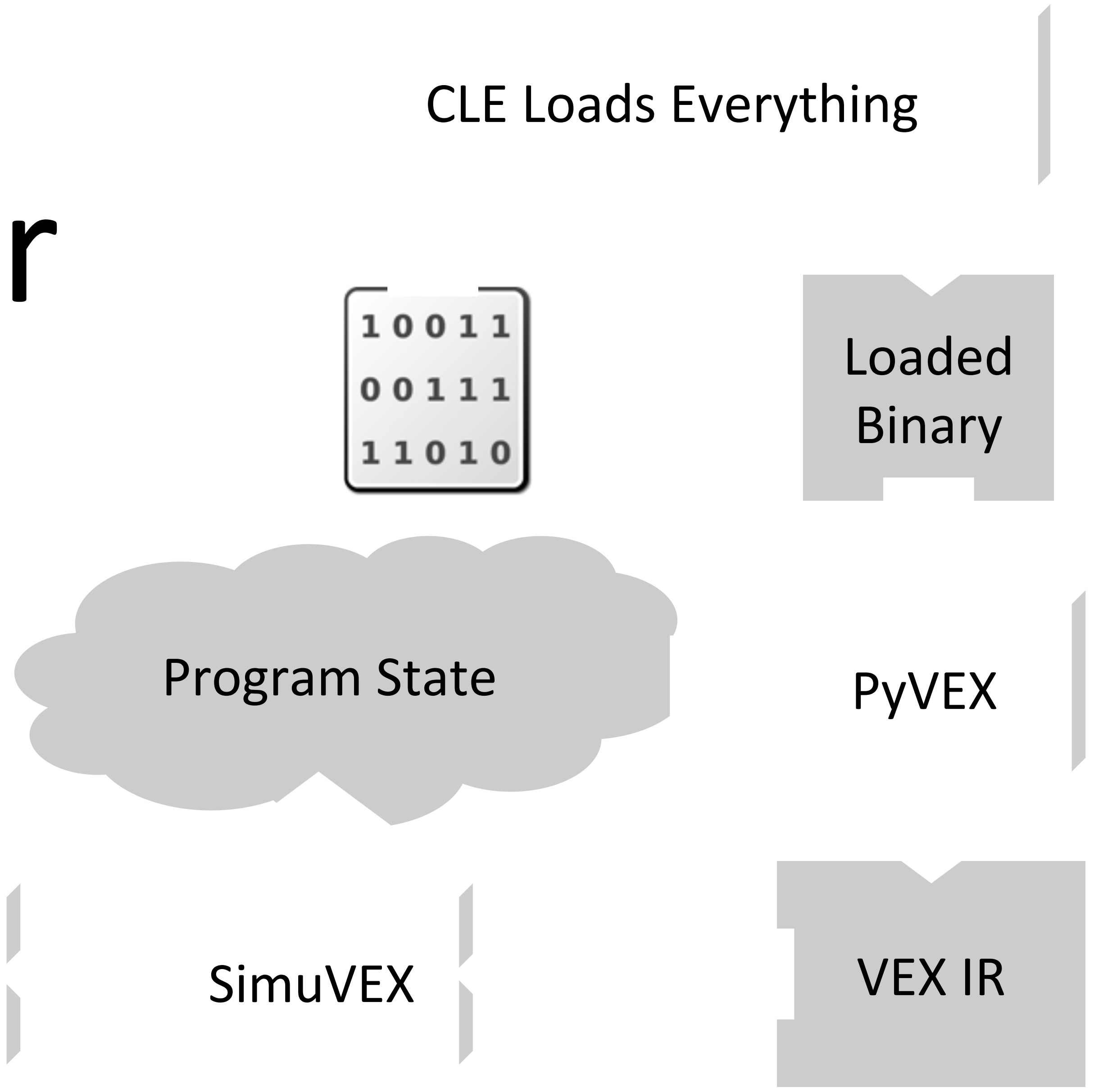
Major contributors:

- Zardus - *Yan Shoshitaishvili*
- Fish - *Ruoyu Wang*
- kereoz - *Christophe Hauser*
- rhelmot - *Andrew Dutcher*
- nezorg - *John Grosen*
- salls - *Chris Salls*

Special thanks to:

- our professors
- DARPA VET Project
- DARPA Cyber Grand Challenge

angr


CLE Loads Everything
Loaded Binary
10011
00111
11010
Program State
PyVEX
VEX IR
Claripy
SimuVEX

Open angr!

- ➔ http://angr.io
- ➔ https://github.com/angr
- ➔ angr@lists.cs.ucsb.edu

Pull requests, issues, questions, etc super-welcome! Let's bring on the next generation of binary analysis!

Draft and backups

- motivation (keep it quick) - 6 mins
    - "In the beginning, there was IDA. However, as the field of binary security advanced, there is now … still IDA?"
    - We need something more!
    - There are a few solutions, but they all suffer from lacking one of: cross-platform, open, active, usable.
- angr fundamentals - 3 mins
    - power (state-of-the-art)
    - ease of use (abstraction)
    - expandable, cross-platform, blah blah
- main components - 20 minutes
    - introduce a demo: some combination of a crackme and a pwnable
    - symbolic execution (slides + demo)
        - the demo should get us past the crackme portion using angr's symbolic execution
    - VSA (slides + demo)
        - the demo should allow us to identify an overflow to pwn
    - dynamic execution (slides + demo)
        - we'll demo a shellcode that's used to exploit the overflow
- angr applications - 10 minutes
    - rop gadget finder (demo)
    - binary diffing
    - Cyber Grand Challenge
- open source! - 3 minutes
    - http://angr.io