CGC Monitor

A vetting system for the DARPA Cyber Grand Challenge

Michael F. Thompson & Timothy Vidas (@tvidas)

DFRWS 2018

CGC Monitor: Presentation outline

DARPA Cyber Grand Challenge overview

Motivation for infrastructure integrity assurances (proactive forensics)

Software vetting on a full system emulator

Running a computer backwards to analyze vulnerabilities

CTF?

What is CTF in this context?

- A cyber security based Capture-the-Flag contest (aka exercise, event, game)
- Typically these contests involve demonstrating proficiency or excellence in one or more areas of computer and network security
- There are different models for architecting these contests, which can stress different skills, lend to particular objectives
- Increasingly popular, common

It is not:

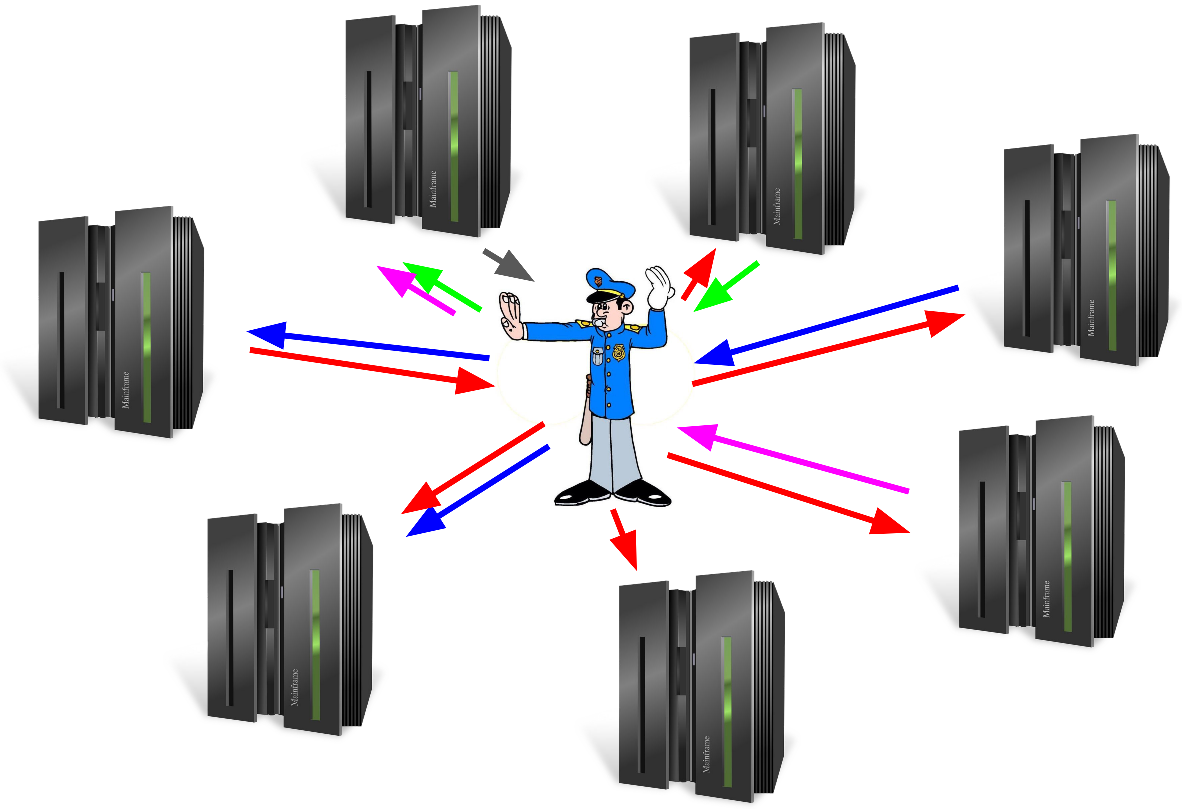
- A game kids play with physical flags on hills
- A first-person shooter video game CTF (usually)
- Focused in the field of Social Engineering
- A hackathon

Though there are certainly similarities to these other games.

Today, the characters “CTF” are appended to many contests, in most cases this simply means “contest,” sometimes there are flags involved

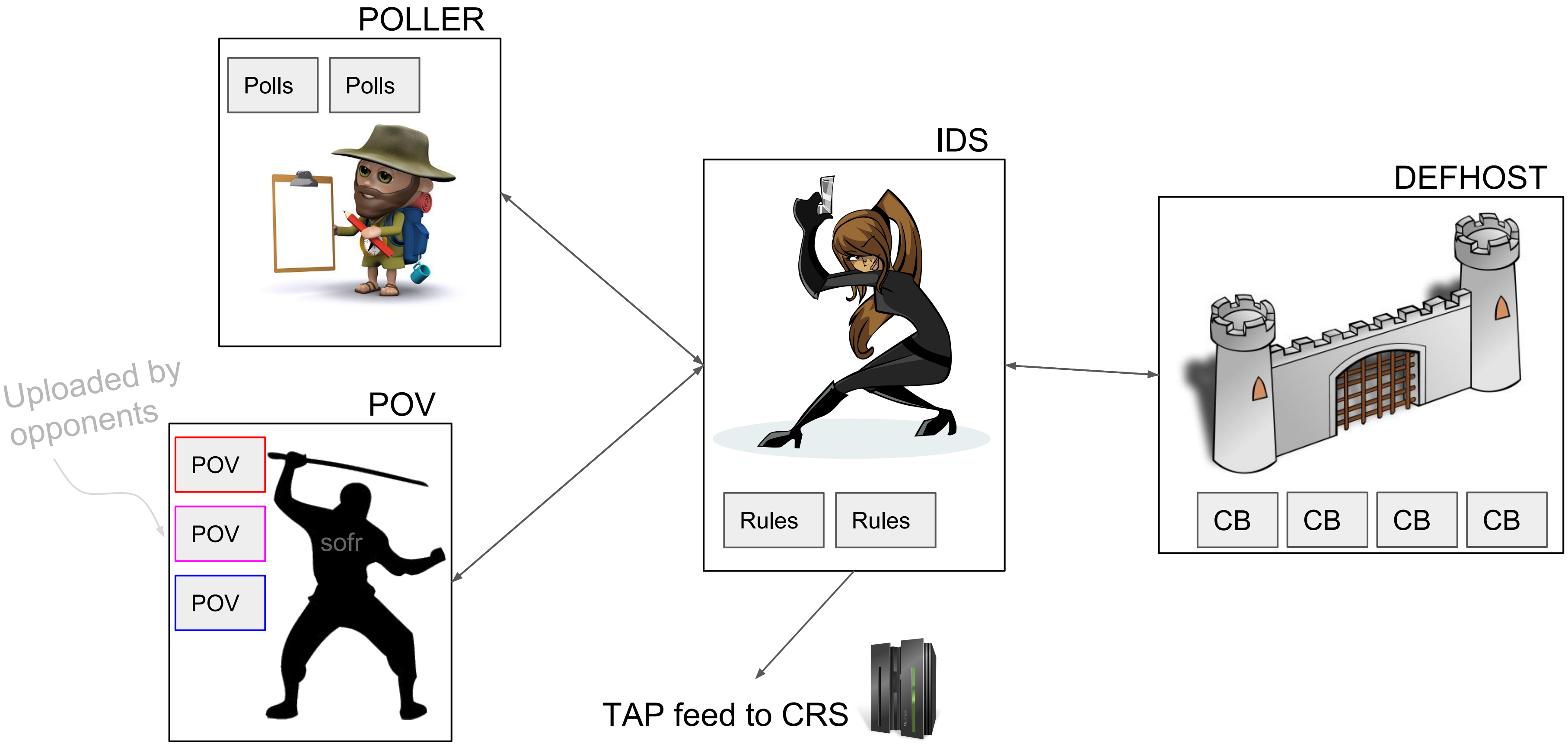
Game Flow

- **Brokered game**
  - Infrastructure mediates everything
  - API designed for autonomous systems
- **Download** binary software
- **Upload** binary software (replacements)
- Register “moves” against targets



https://github.com/CyberGrandChallenge/cgc-release-documentation/blob/master/ti-api-spec.txt

Game Flow



POLLER

Polls

Polls

IDS

Rules

Rules

DEFHOST

CB

CB

CB

CB

Uploaded by opponents

POV

POV

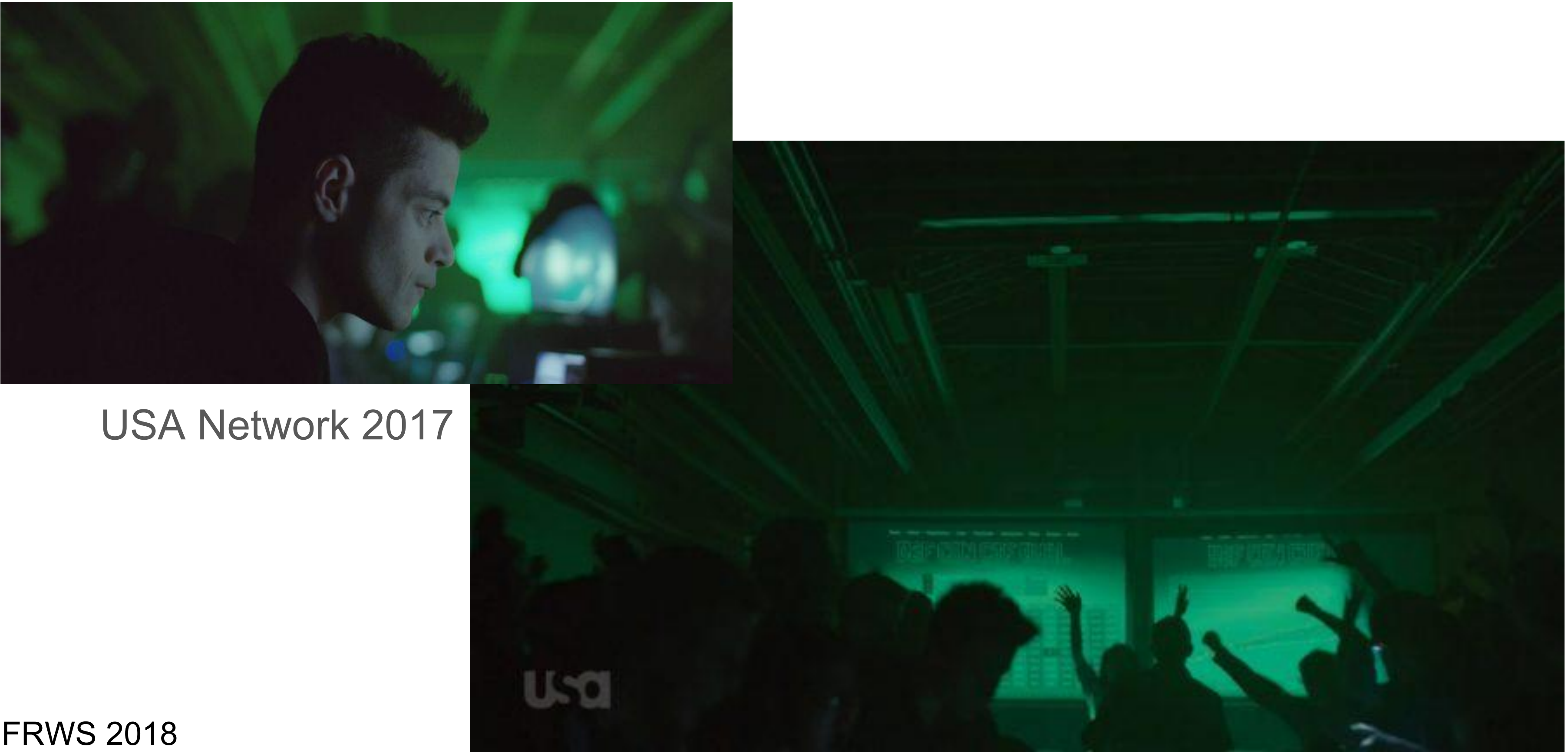
POV

POV

sofr

TAP feed to CRS

CTF: Hollywood style (well, USA Network)



USA Network 2017

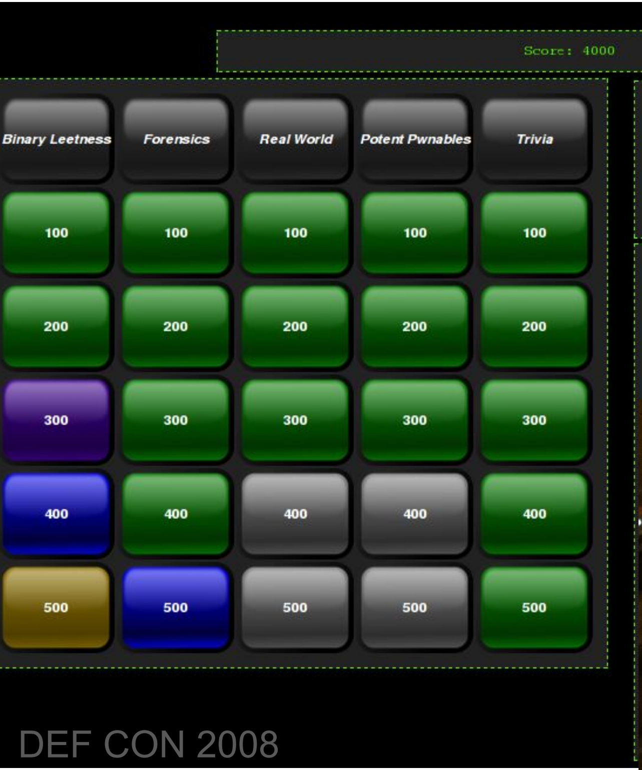
CTF: real life

DEF CON 2002

DEF CON 2016

DEF CON's CTF is often cited as the "world series" or "superbowl" of CTFs

CTF: real life



Logout TheUnd

Score: 4000

Binary Leetness	Forensics	Real World	Potent Pwnables	Trivia
100	100	100	100	100
200	200	200	200	200
300	300	300	300	300
400	400	400	400	400
500	500	500	500	500

What libc function is this?

- file

i hate you

I owned it

Leaders

1. Routards (5200)
2. Pandas with Gambas (5200)
3. Guard@MyLan0 (4800)

DEF CON 2008

DEF CON 2012

DEF CON 2011?

CGC?

Could a purpose-built super computer play in DEF CON’s Capture-the-flag (CTF)?

Autonomous…

- ○ Binary analysis
- ○ Binary patching
- ○ Vulnerability discovery
- ○ Service Resiliency (availability)
- ○ Network Defense (IDS)

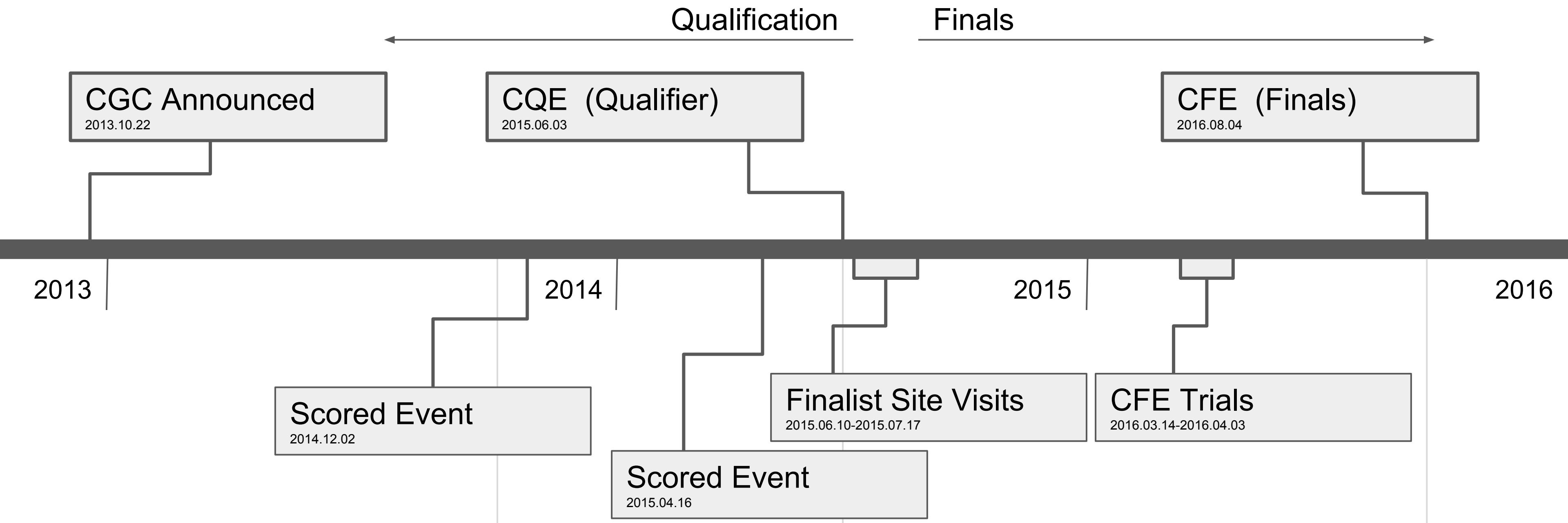
CGC: Real life

Image: DARPA

CGC: Real life

Image: DARPA

Competition Overview



Qualification

Finals

CGC Announced
2013.10.22

CQE (Qualifier)
2015.06.03

CFE (Finals)
2016.08.04

2013

2014

2015

2016

Scored Event
2014.12.02

Scored Event
2015.04.16

Finalist Site Visits
2015.06.10-2015.07.17

CFE Trials
2016.03.14-2016.04.03

104 applicants
2014.11..2

(7 Funded Track)

28 Scored Event participants

13 CQE participants

7 Finalists

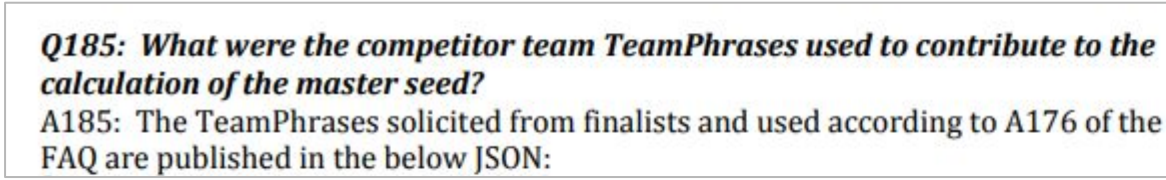
(3 Funded Track)

Building the Competition

- Design concerns from the outset
    - Repeatability
        - Anyone should be able to verify CFE results
    - Competition integrity
        - Concerns with running competitor-provided code (POV/RCB)
        - Concerns with parsing competitor-provided data (IDS filters)
    - Data collection
        - Desire to publish corpus to serve as a reference for program analysis going forward

Competition Integrity

- Given the amount of prize money at stake, integrity of the competition was a grave concern and drove many design decisions
- Randomness was limited and/or made to be deterministically pseudorandom
- However, **nobody** should be able to predict aspects of CFE
  - The entire event was seeded with input from DARPA and all competitors (XORed) (Collected between June 10-17, 2016)
  - To ensure that DARPA did not select a particular input after knowing all competitor inputs DARPAs input was cryptographically committed to early (June 10,2016)
- Similarly, the CFE event plan (including challenge set schedule was committed to on Aug 2, 2016)
  - Organizers could not change the schedule in order to influence the event outcome



Q185: What were the competitor team TeamPhrases used to contribute to the calculation of the master seed?
A185: The TeamPhrases solicited from finalists and used according to A176 of the FAQ are published in the below JSON:

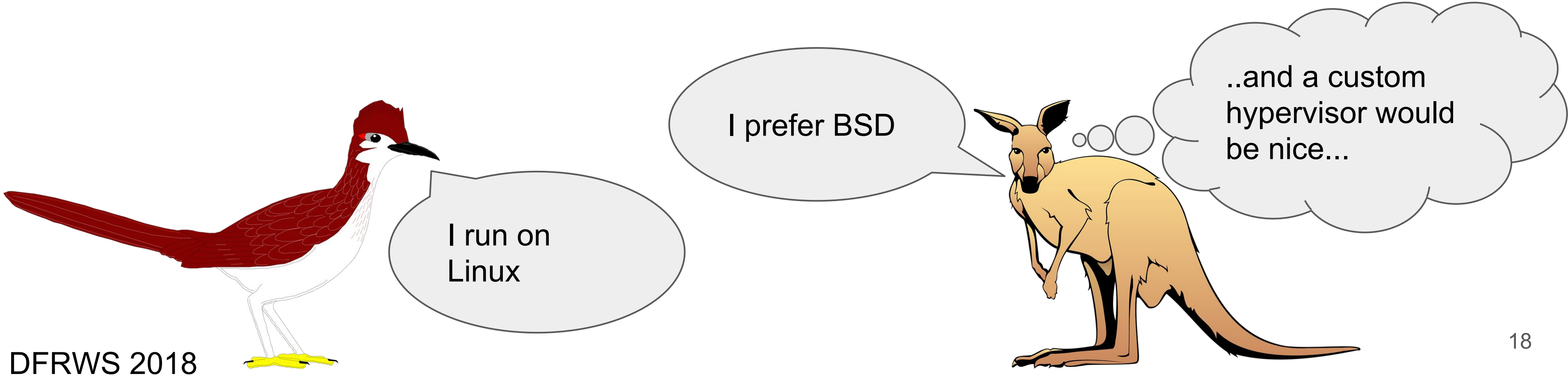
Weeks of my life were lost to this

https://github.com/CyberGrandChallenge/Event-FAQ/blob/master/event_faq.md
http://archive.darpa.mil/cybergrandchallenge_competitorsite/Files/CGC_FAQ.pdf

Competition Integrity

7 system calls
_terminate, transmit, receive, fdwait,
allocate, deallocate, random

- Committed to kernels versions released prior to announcement of CGC
- Designed DECREE syscall environment / file format to reduce attack surface
- All game infrastructure components released to the public had private internal implementations
  - Notably, CFE ran on 64-bit FreeBSD 10 with a custom hypervisor module



Competition Integrity

- Air Gap
  - ○

Image: Vidas

Competition Integrity

- Air Gap
  - ○ Power, cooling

Image: Vidas

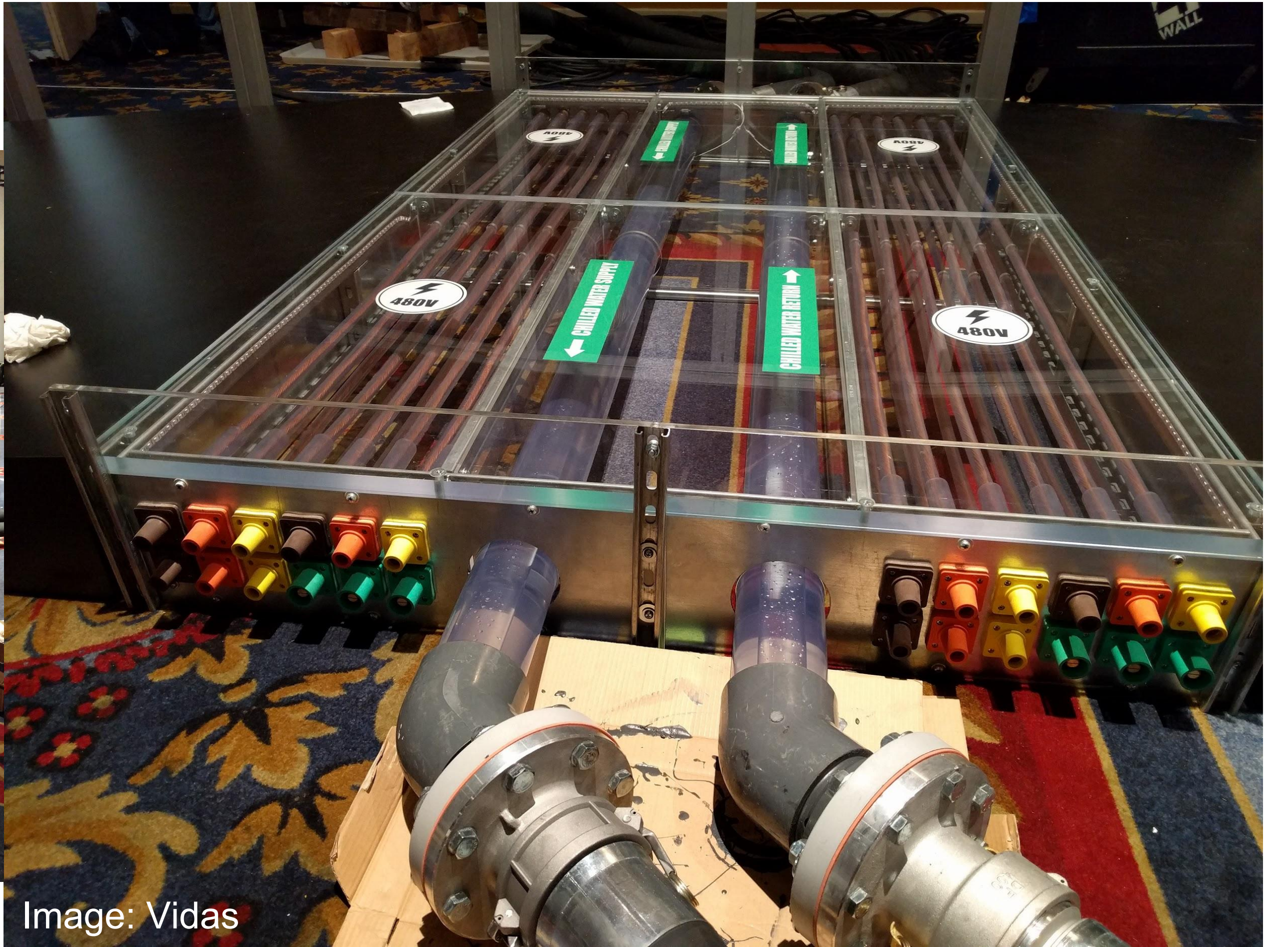


Image: Vidas

Competition Integrity

- Air Gap
  - One-way data

Image: Vidas

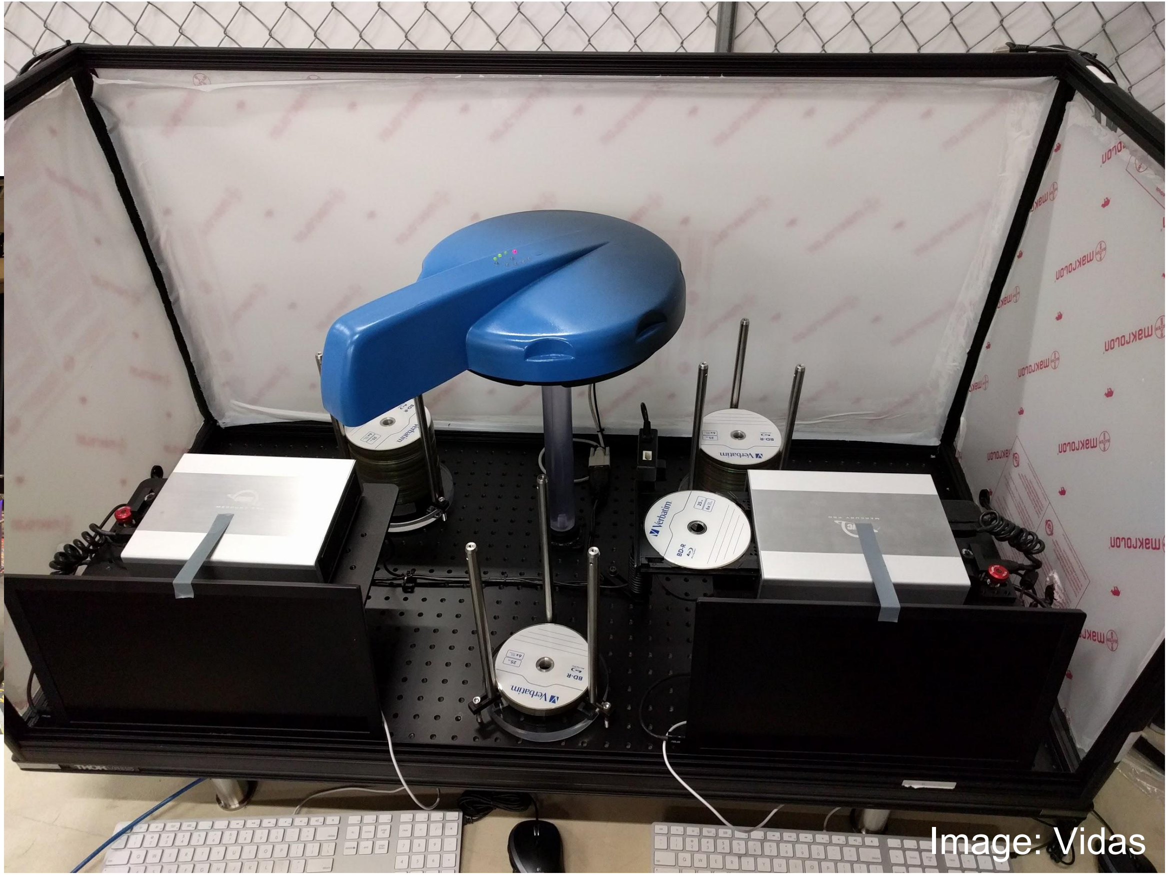


Image: Vidas

Competition Integrity

- Competitors were required to be autonomous, organizers weren’t
- Referees
- However, air gap

- Redundant HW
- Power/cooling
- Monitoring

Image: DARPA

Competition Integrity: Forensics

- Real-time forensics harness to vet software
  - Monitor OS for execution & data integrity
  - Built upon a full system emulator (Simics)
  - High fidelity x86 model from Intel
- Evaluated non-trusted code (POV/RCB) for attempts to breakout of DECREE environment
- Analyst replay tool
  - Replay any CFE session via IDA Pro gdb client
  - Reverse execution & scoring event detection

CGC Monitor vetted all competitor submissions

CGC infrastructure duplicated on the Simics full system simulator

- Multiple components; all game services
- Monitor OS for execution & data integrity
- High fidelity x86 model from Intel

GCG Monitor built upon Simics primarily from breakpoints and callbacks

- Implementation similar to dynamic VM introspection
- No monitoring functions execute on monitored systems
- Built custom “OS awareness” subsystem based on OS internals
- Variations for 32/64 bit Linux and FreeBSD (and combinations thereof)

Implemented on 32 blade servers with multiple instances of CGC systems

What was monitored?

Competitor-supplied software

- Proofs of vulnerabilities executing on PoV throwers
- Replacement challenge binaries on the defended host
- IDS subsystem while consuming competitor’s filters

While scheduled for execution:

- Kernel ROP -- execution of a “ret” not following a “call”
- Page tables allocated to the kernel
- Process credentials -- e.g., effective user ID
- Unexpected code sections -- e.g., process create while an RCB runs

Artifacts generated by monitoring

Anomalous events from kernel monitoring

Full execution traces, including data references

System call logs, including all parameters

Successful Proofs of Vulnerabilities (PoVs) against services

ROP or stack area execution in services

Faults in services, e.g., segmentation violations leading to crashes

CGC Analysis Tool: Running a computer backwards

Real world analogy: Your hybrid fuzzer found a vulnerability: But what is the bug?

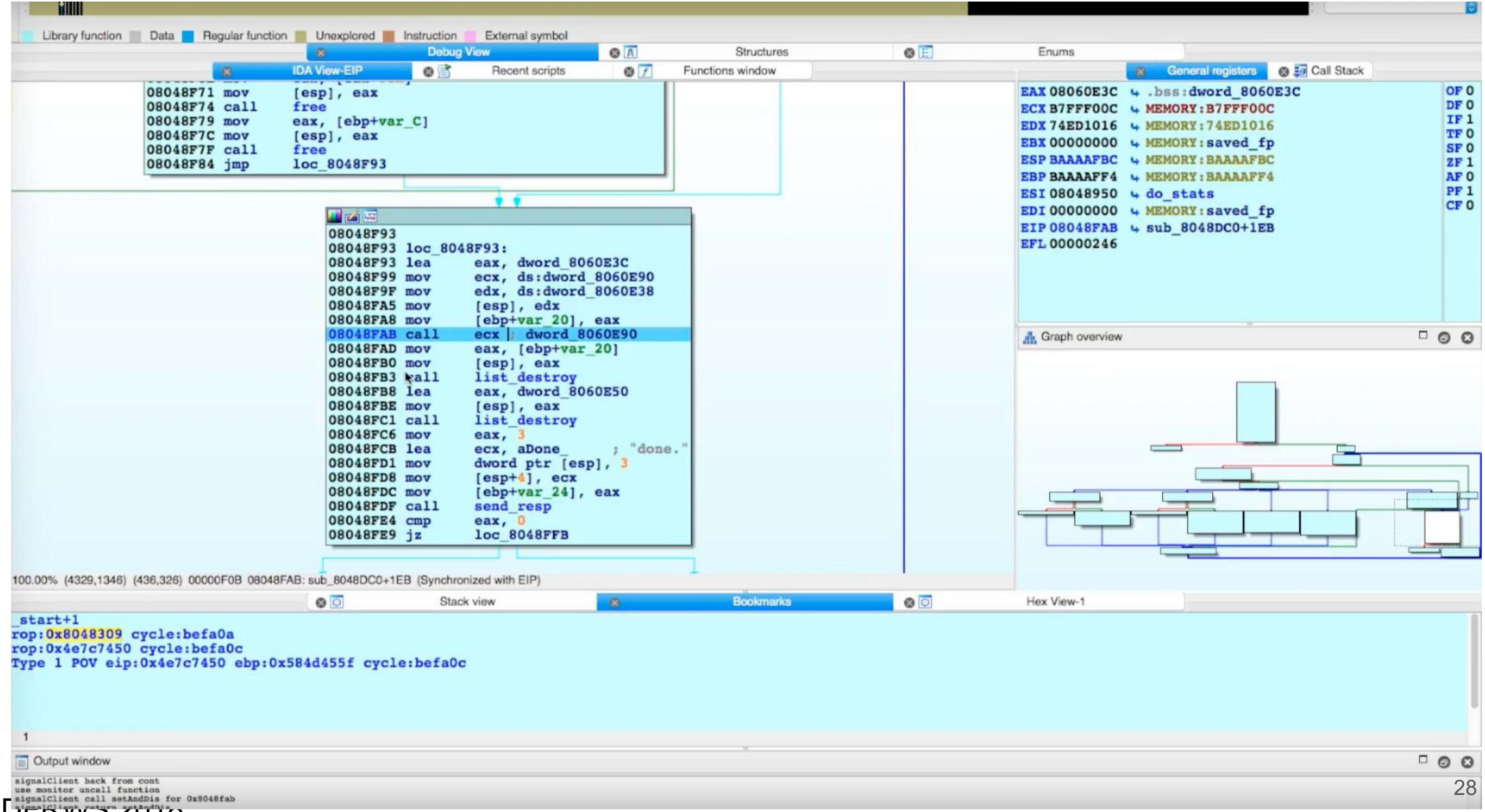
Competitors found 20 vulnerabilities in 82 challenge sets. But what flaws?

Analysis of effective patches would not help: they were all generic

Instrumented the full system simulator for analysis of application exploitation

Automatically detect a successful exploit and pause the session

- Analyst can then use reverse execution to track the bug
- IDA Pro debugger client as a front end to the CGC Monitor



```
08048F71 mov     [esp], eax
08048F74 call    free
08048F79 mov     eax, [ebp+var_C]
08048F7C mov     [esp], eax
08048F7F call    free
08048F84 jmp     loc_8048F93
```

```
08048F93
08048F93 loc_8048F93:
08048F93 lea     eax, dword_8060E3C
08048F99 mov     ecx, ds:dword_8060E90
08048F9F mov     edx, ds:dword_8060E38
08048FA5 mov     [esp], edx
08048FA8 mov     [ebp+var_20], eax
08048FAB call    ecx ; dword_8060E90
08048FAD mov     eax, [ebp+var_20]
08048FB0 mov     [esp], eax
08048FB3 call    list_destroy
08048FB8 lea     eax, dword_8060E50
08048FBE mov     [esp], eax
08048FC1 call    list_destroy
08048FC6 mov     eax, 3
08048FCB lea     ecx, aDone_      ; "done."
08048FD1 mov     dword ptr [esp], 3
08048FD8 mov     [esp+4], ecx
08048FDC mov     [ebp+var_24], eax
08048FDF call    send_resp
08048FE4 cmp     eax, 0
08048FE9 jz      loc_8048FFB
```

```
EAX 08060E3C ↳ .bss:dword_8060E3C     OF 0
ECX B7FFF00C ↳ MEMORY:B7FFF00C        DF 0
EDX 74ED1016 ↳ MEMORY:74ED1016        IF 1
EBX 00000000 ↳ MEMORY:saved_fp        TF 0
ESP BAAAAFBC ↳ MEMORY:BAAAAFBC        SF 0
EBP BAAAAFF4 ↳ MEMORY:BAAAAFF4        ZF 1
ESI 08048950 ↳ do_stats               AF 0
EDI 00000000 ↳ MEMORY:saved_fp        PF 1
EIP 08048FAB ↳ sub_8048DC0+1EB        CF 0
EFL 00000246
```

100.00% (4329,1346) (436,326) 00000F0B 08048FAB: sub_8048DC0+1EB (Synchronized with EIP)

```
_start+1
rop:0x8048309 cycle:befa0a
rop:0x4e7c7450 cycle:befa0c
Type 1 POV eip:0x4e7c7450 ebp:0x584d455f cycle:befa0c
```

1

```
signalClient back from cont
use monitor uncall function
signalClient call setAndDis for 0x8048fab
signalClient return setAndDis
```

IDA Pro Extensions for Reverse Execution

Reverse (e.g., until a breakpoint is encountered)

Step backwards over or into a function

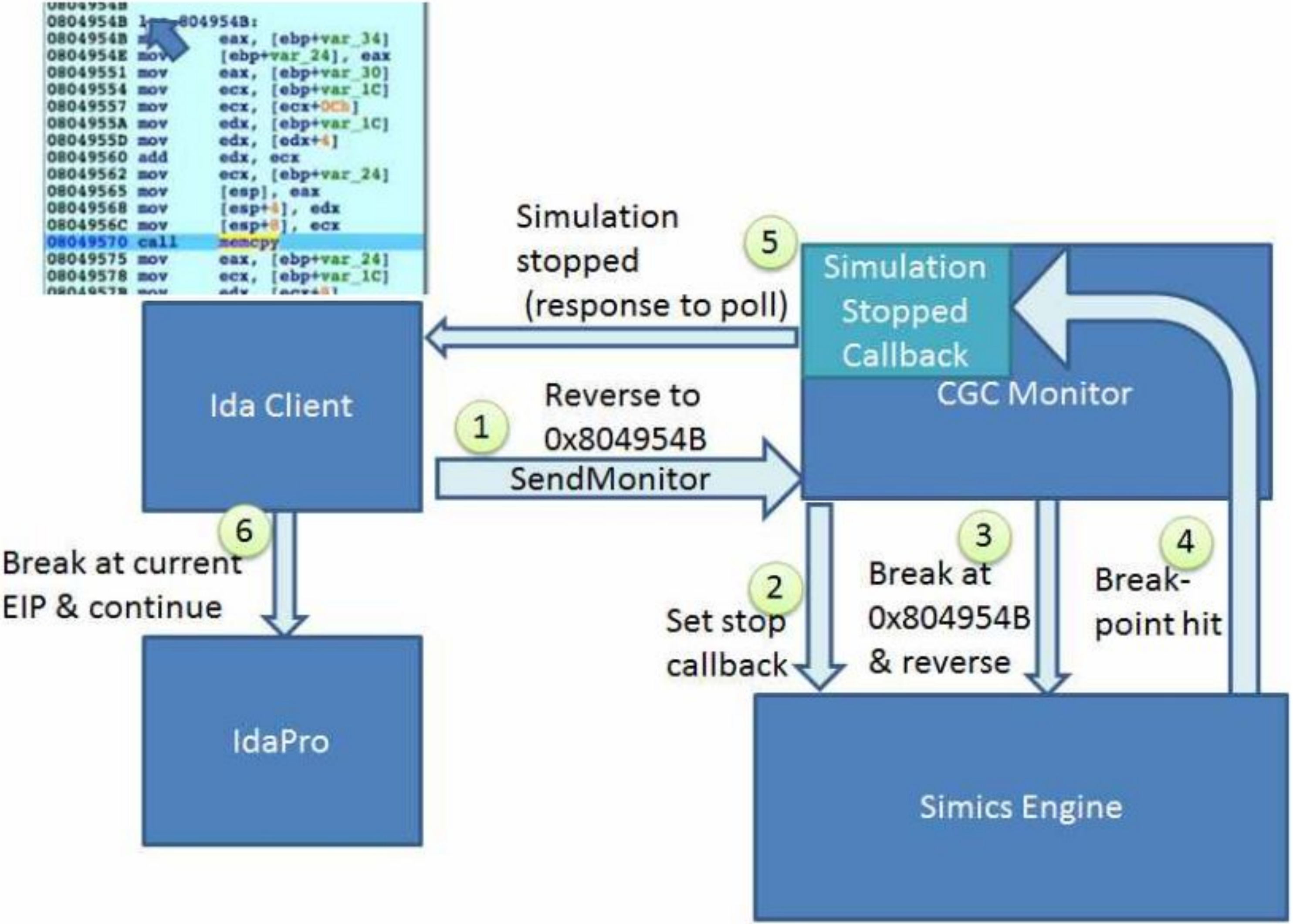
Reverse to cursor

Reverse to previous write of selected register or address

Set or jump to an execution bookmark managed by the user

Back trace the source of data in address or register

- Often leads all the way back to the syscall that received data
- Halts on computed assignments (e.g., addition -- but not increment)



```
0804954B
0804954B loc_804954B:
0804954B mov     eax, [ebp+var_34]
0804954E mov     [ebp+var_24], eax
08049551 mov     eax, [ebp+var_30]
08049554 mov     ecx, [ebp+var_1C]
08049557 mov     ecx, [ecx+0Ch]
0804955A mov     edx, [ebp+var_1C]
0804955D mov     edx, [edx+4]
08049560 add     edx, ecx
08049562 mov     ecx, [ebp+var_24]
08049565 mov     [esp], eax
08049568 mov     [esp+4], edx
0804956C mov     [esp+8], ecx
08049570 call    memcpy
08049575 mov     eax, [ebp+var_24]
08049578 mov     ecx, [ebp+var_1C]
0804957B mov     edx, [ecx+8]
```

Ida Client

IdaPro

CGC Monitor

Simulation Stopped Callback

Simics Engine

1 Reverse to 0x804954B SendMonitor

2 Set stop callback

3 Break at 0x804954B & reverse

4 Break-point hit

5 Simulation stopped (response to poll)

6 Break at current EIP & continue

Simics illusion of reverse execution

Resource intensive, enable only for analyst sessions

Records “micro-checkpoints” referenced during reverse execution

Iterates from checkpoints, running forward until “most recent” breakpoint

Warning: backwards progression is not serial

> Callback for one breakpoint may be invoked many times
> Breakpoint callbacks are therefore not useful when reversing
> Associate callback with simulation “stop” event
> Then figure out where you are and why you stopped

Analysis of CGC Final Event

82 Challenge sets, having 109 intended vulnerabilities

20 challenge sets had working POVs in CFE

Half of these working POVs were not what the author intended

Six were different vulnerabilities (2 services exploited via same bug)

Four were the intended bug, exploited via an simpler alternate path

All exploits of each challenge set used the same vulnerability and path

Fully automated back trace of data

Back trace sources of data, e.g., to a receive syscall (like backwards taint analysis)

Corrupted return addresses

Corrupt values of call registers

Executable payloads

General register values negotiated in Type 1 PoVs

The source of protected memory addresses

Traces available in the CGC Corpus at: http://www.lungetech.com/cgc-corpus/

Future Work & Availability

Extend for general application environments (currently DECREE)

Package Analysis Tool as a remotely accessible service

CGC Monitor at https://github.com/mfthomps/cgc-monitor

BYOS (bring your own Simics)

Analysis results at https://github.com/mfthomps/CGC-Analysis