DARPA's Cyber Grand Challenge: Creating a League of Extra-Ordinary Machines*

Ben Price and Michael Zhivich

ACSAC
December 10, 2015

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

* This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under US Air Force Contract FA8721-05-C-0002.
Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Could a Purpose-Built Supercomputer Play DEF CON Capture-the-Flag (CTF)?

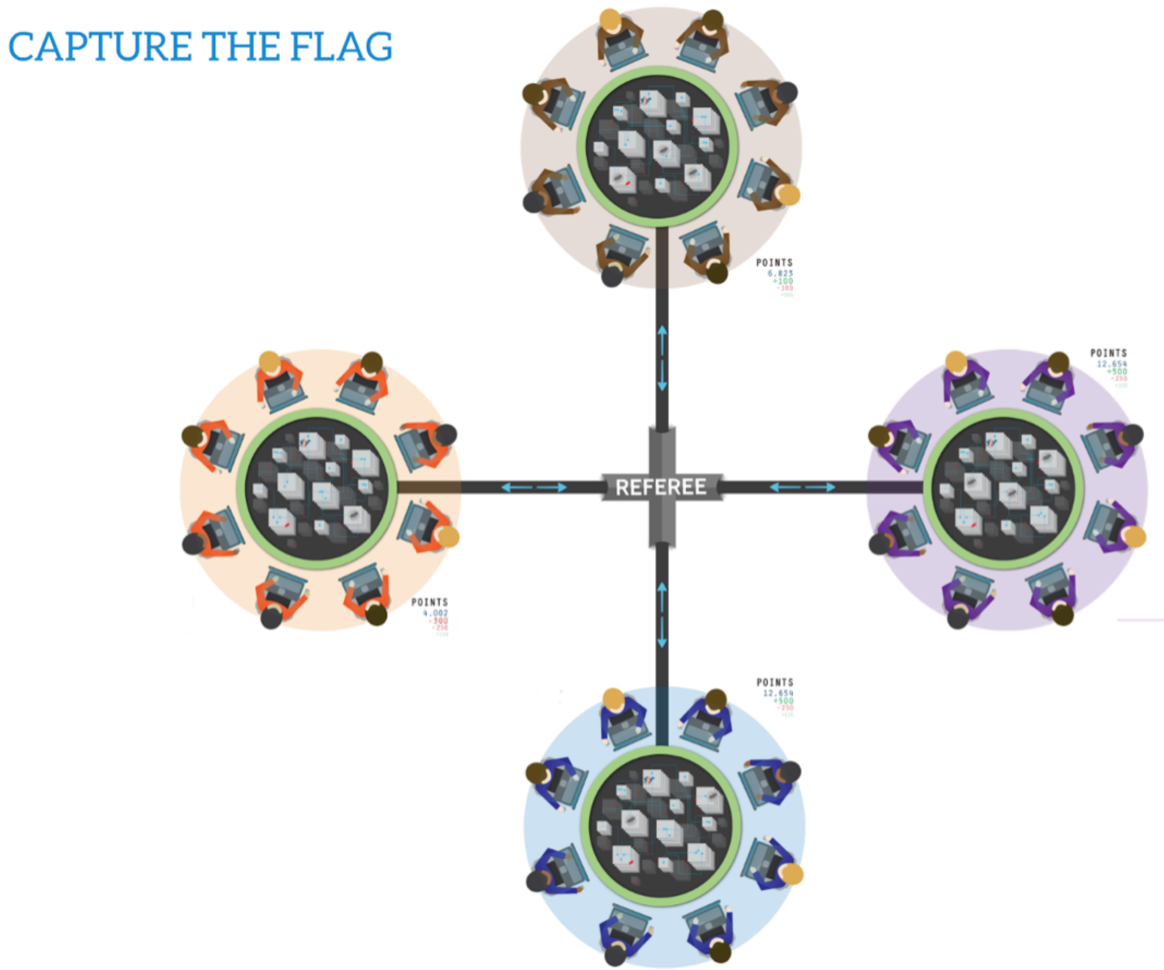
Cyber Grandmasters

Program Analysis Experts

Dedicated Systems

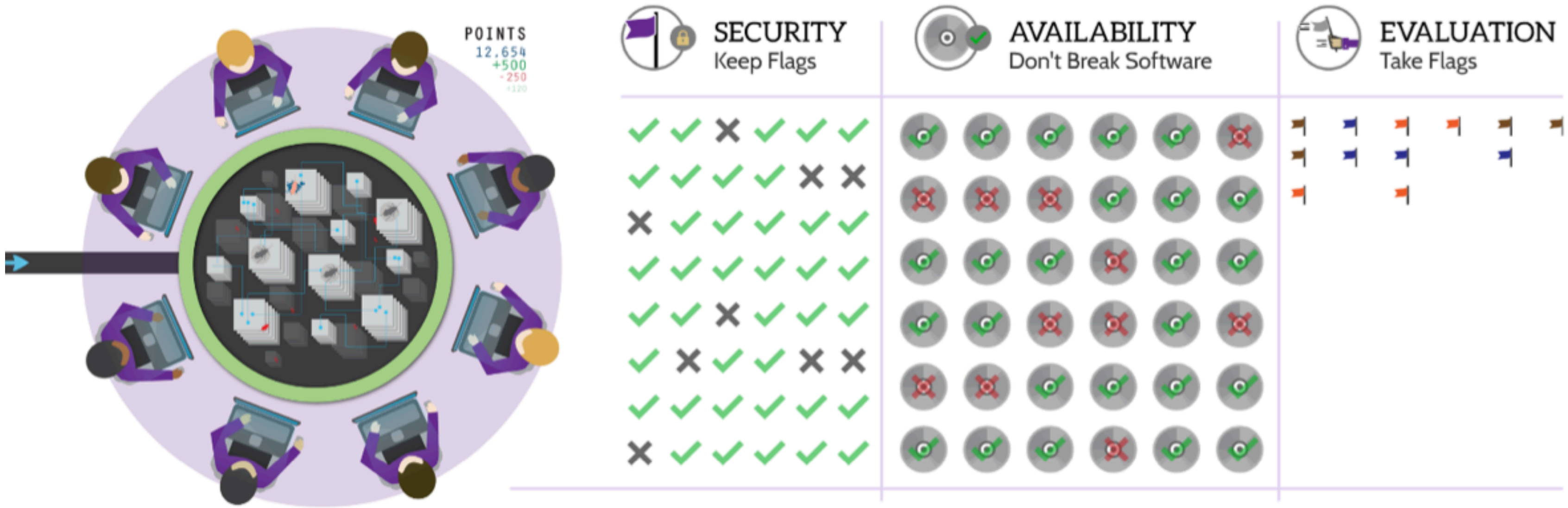
Capture-the-Flag (CTF)

What is Capture-the-Flag?



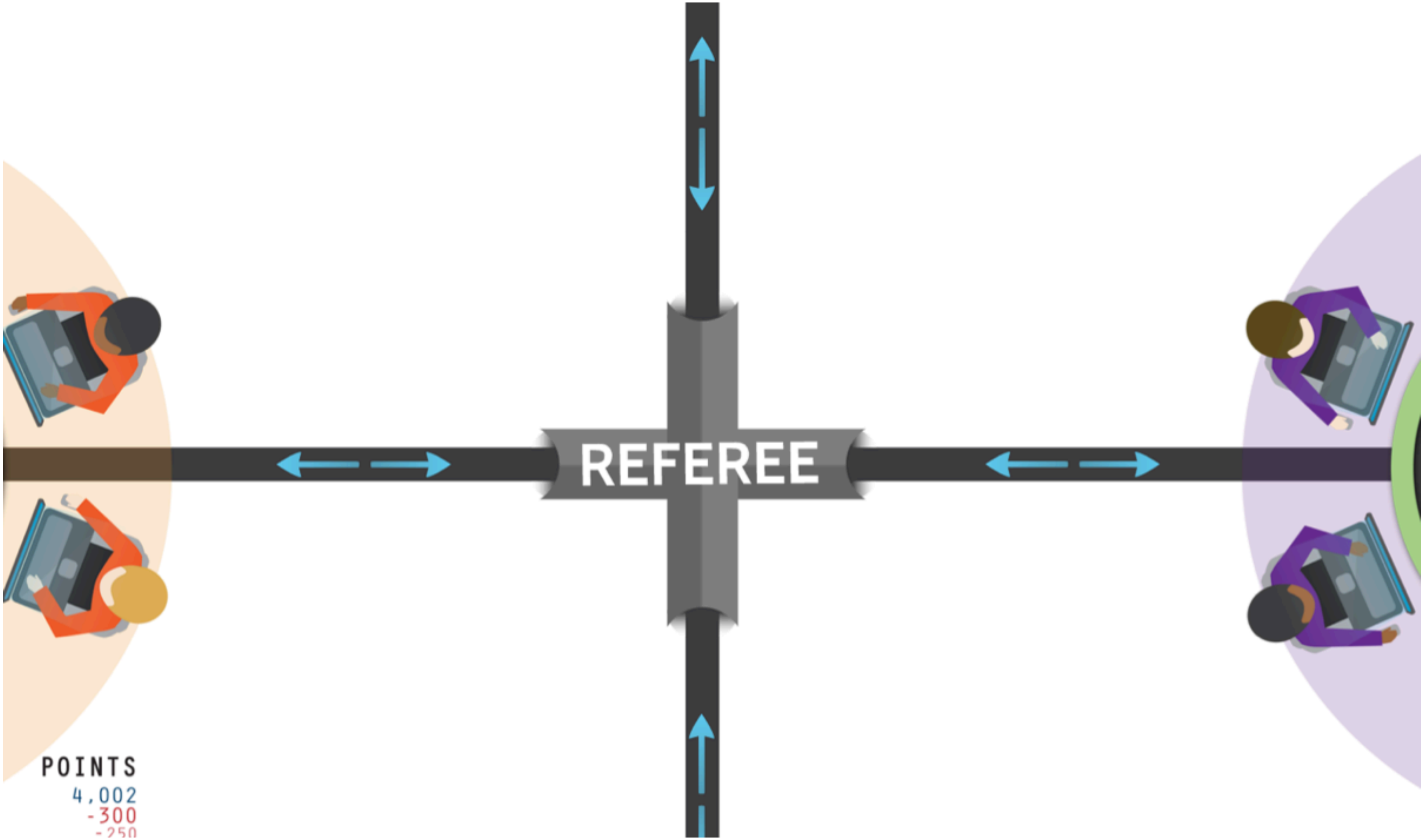
Source: DARPA

What is Capture-the-Flag?



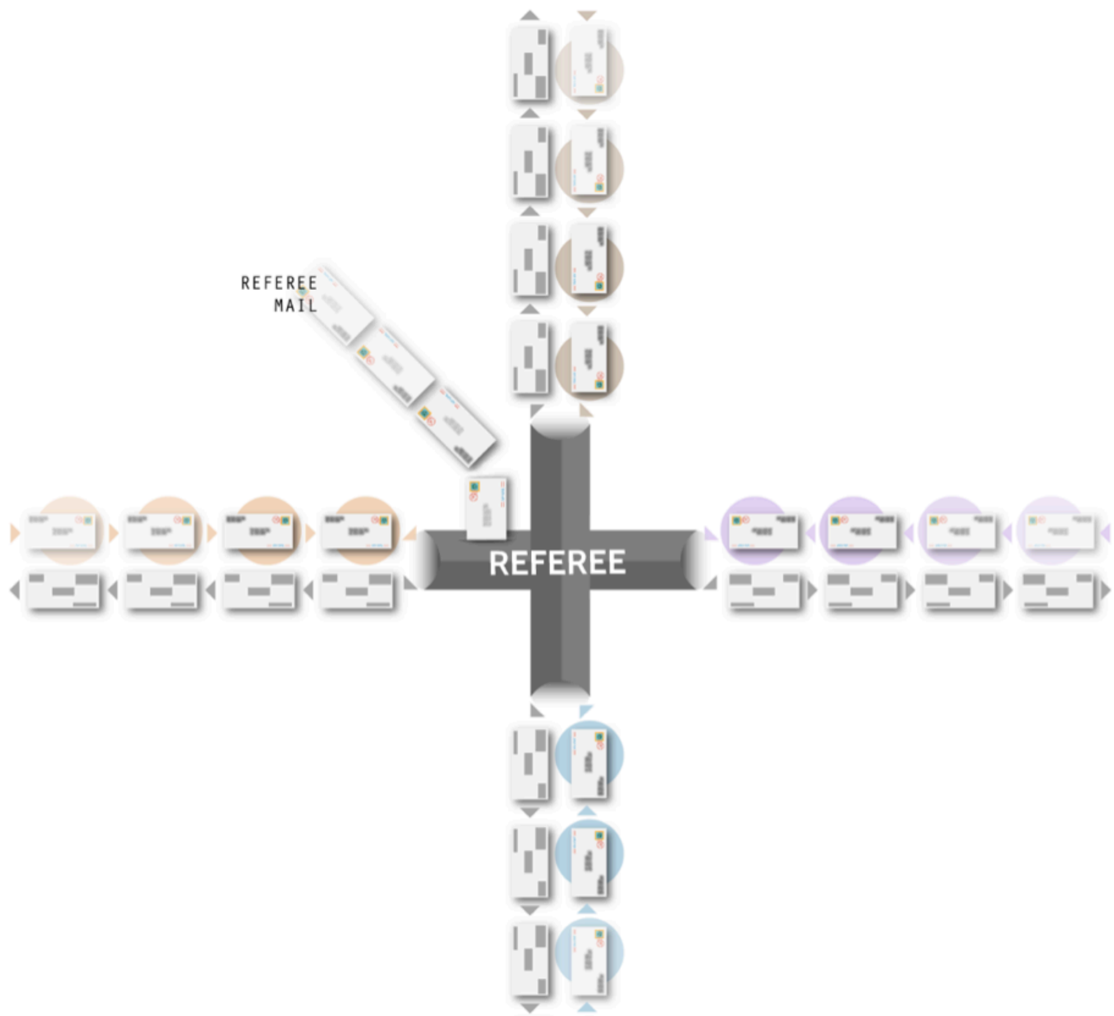
Source: DARPA

What is Capture-the-Flag?



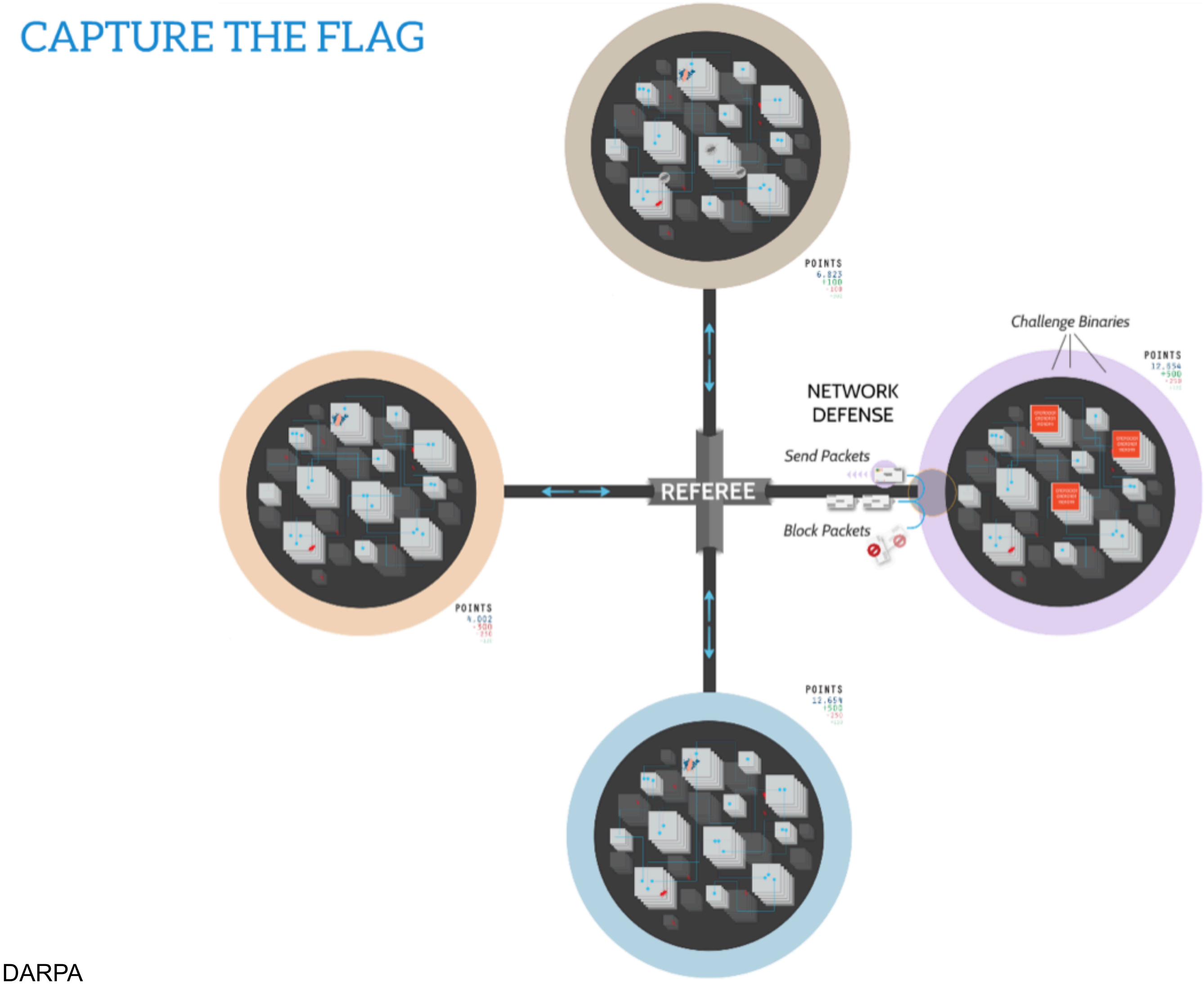
Source: DARPA

What is Capture-the-Flag?



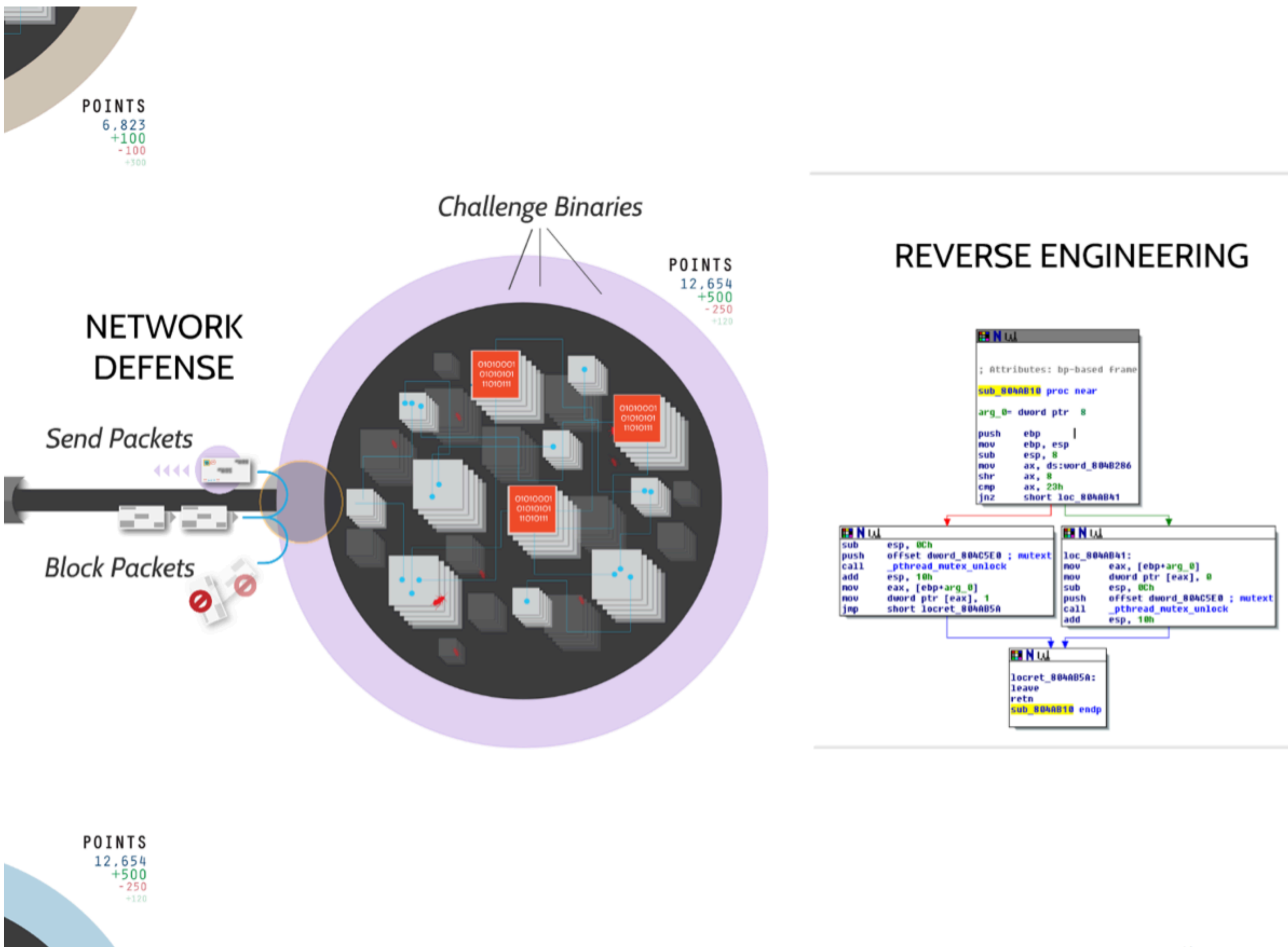
Source: DARPA

What is Capture-the-Flag?



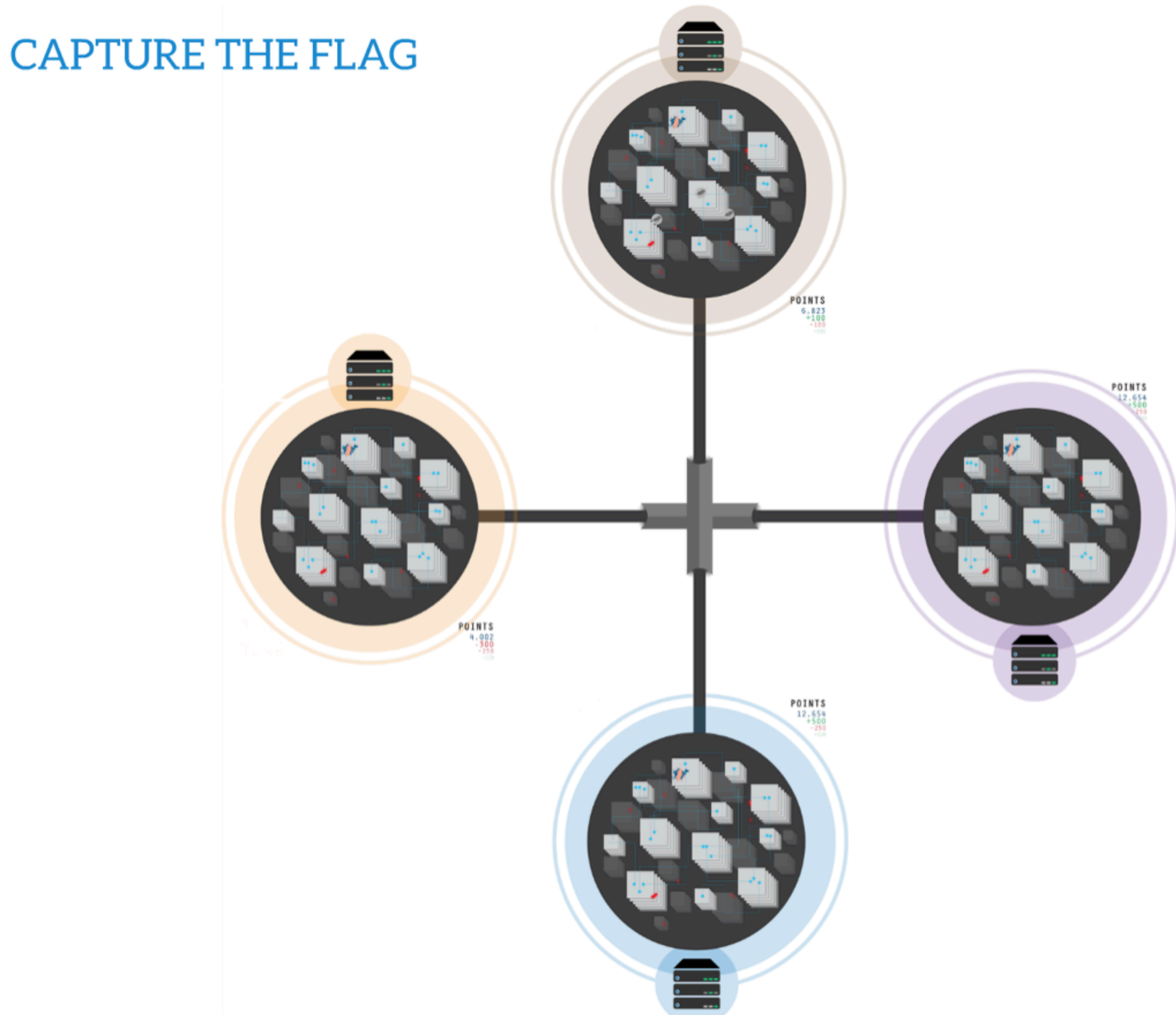
Source: DARPA

What is Capture-the-Flag?



Source: DARPA

Cyber Grand Challenge: Create Cyber Reasoning Systems (CRS)



Source: DARPA

A League of Extra-Ordinary Machines

Chess Grandmasters

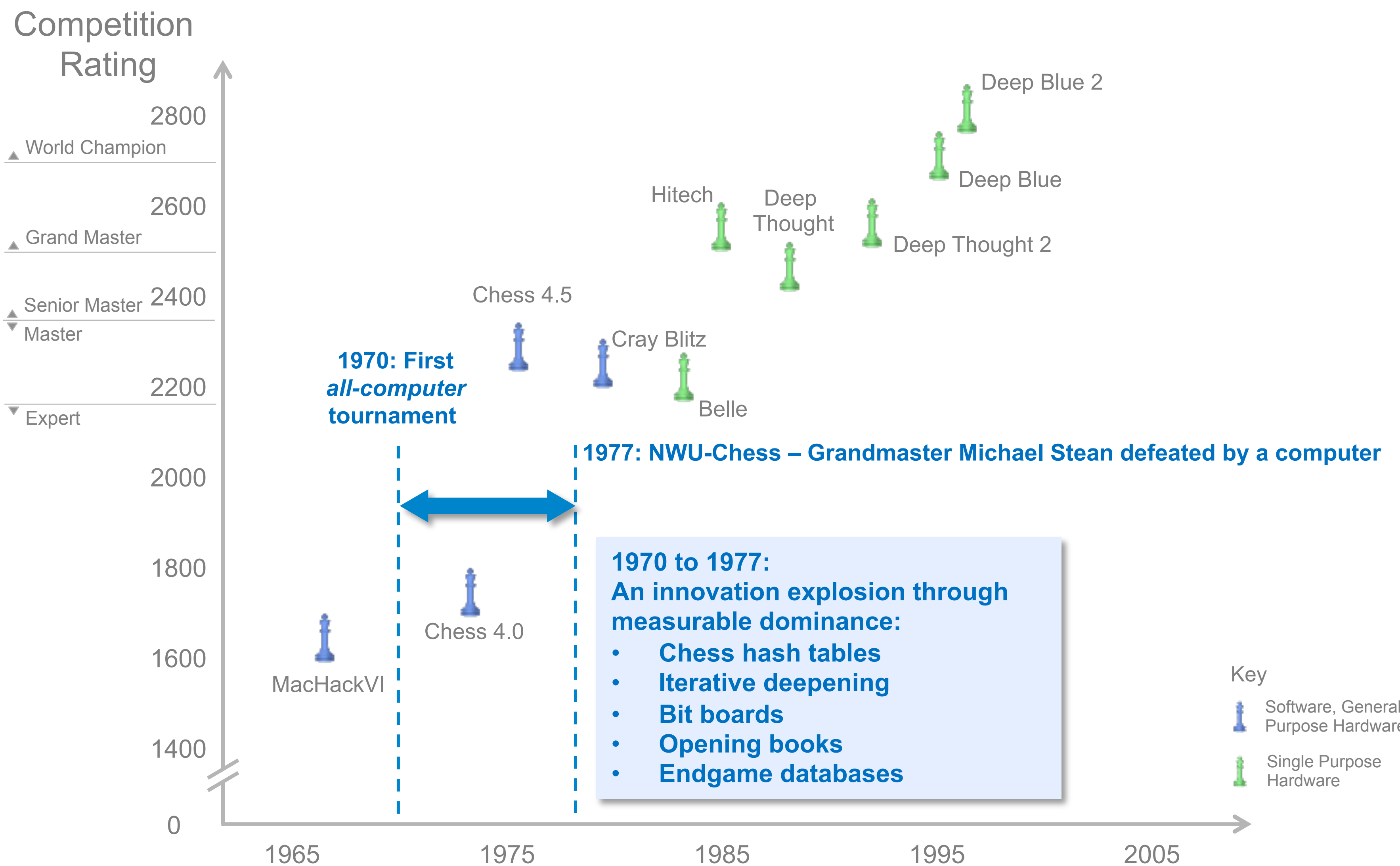
World Class Computer Science

Dedicated Systems

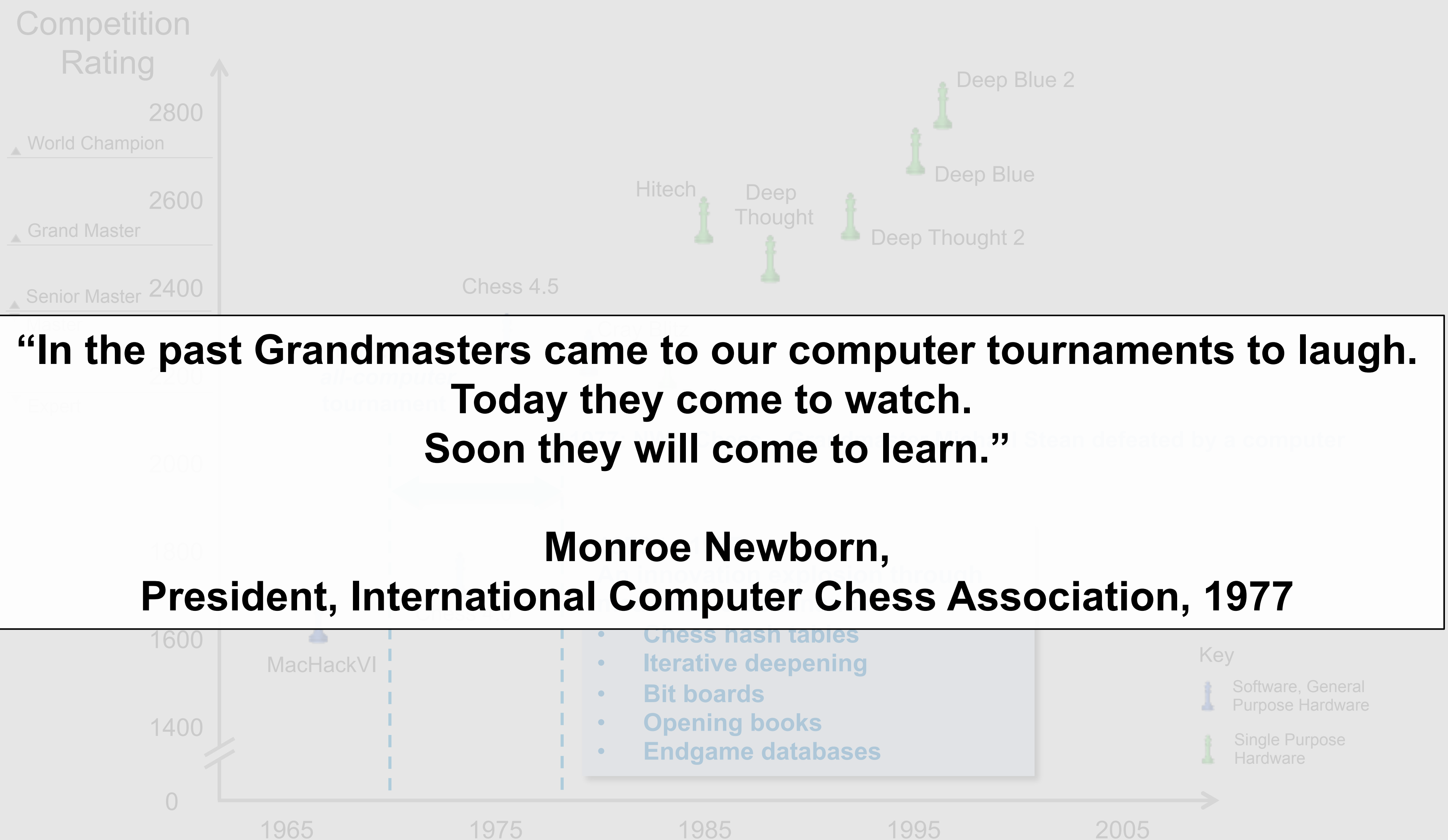
Chess

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A League of Their Own



A League of Their Own



"In the past Grandmasters came to our computer tournaments to laugh.
Today they come to watch.
Soon they will come to learn."

Monroe Newborn,
President, International Computer Chess Association, 1977

June 3, 2015: In the Beginning…

DARPA
@DARPA
Following

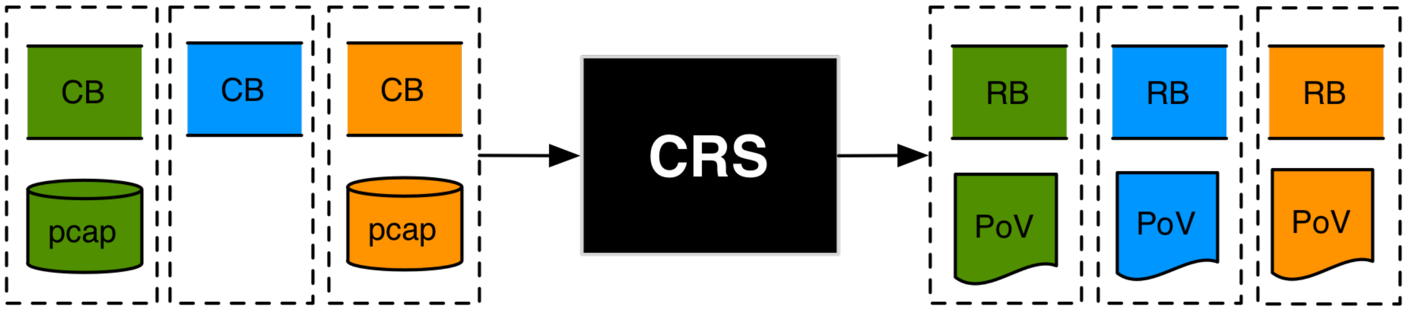
"We held the world's biggest [#capturetheflag] and all the contestants were robots." #cybersecurity #DARPACGC

Cyber Grand Challenge Timeline

Source: DARPA

Challenge for Infrastructure Team

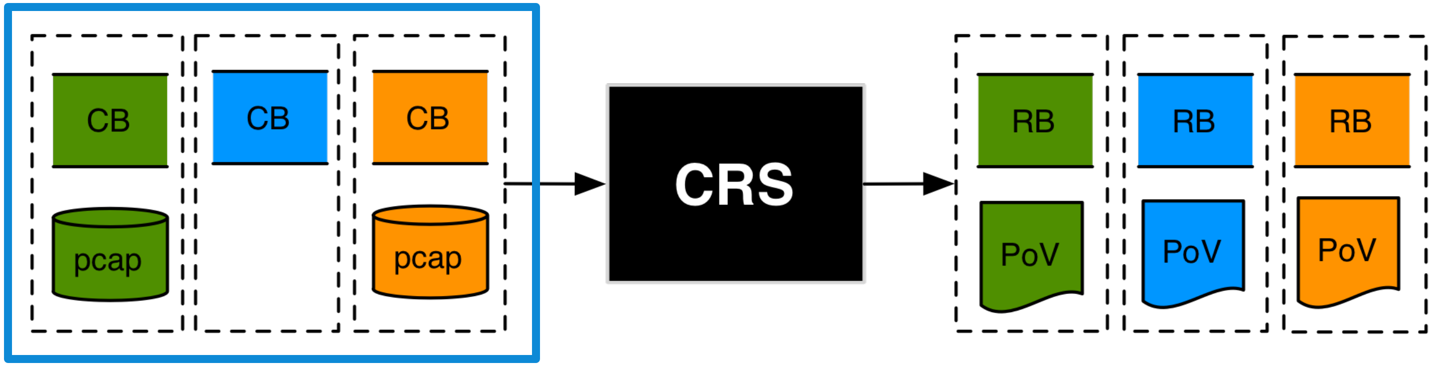
Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning



CRS: Cyber Reasoning System
CB: Challenge Binary
pcap: Packet capture
RB: Replacement Binary
PoV: Proof of Vulnerability

Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

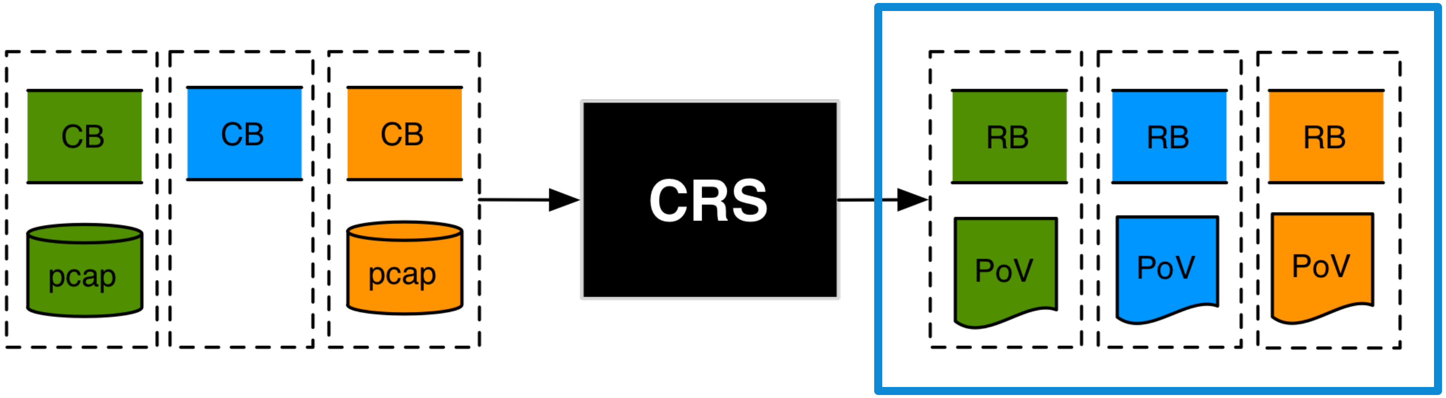


Alternative Ecosystem

How do we make the test corpus representative of real-world challenges but not tainted by prior knowledge?

Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning

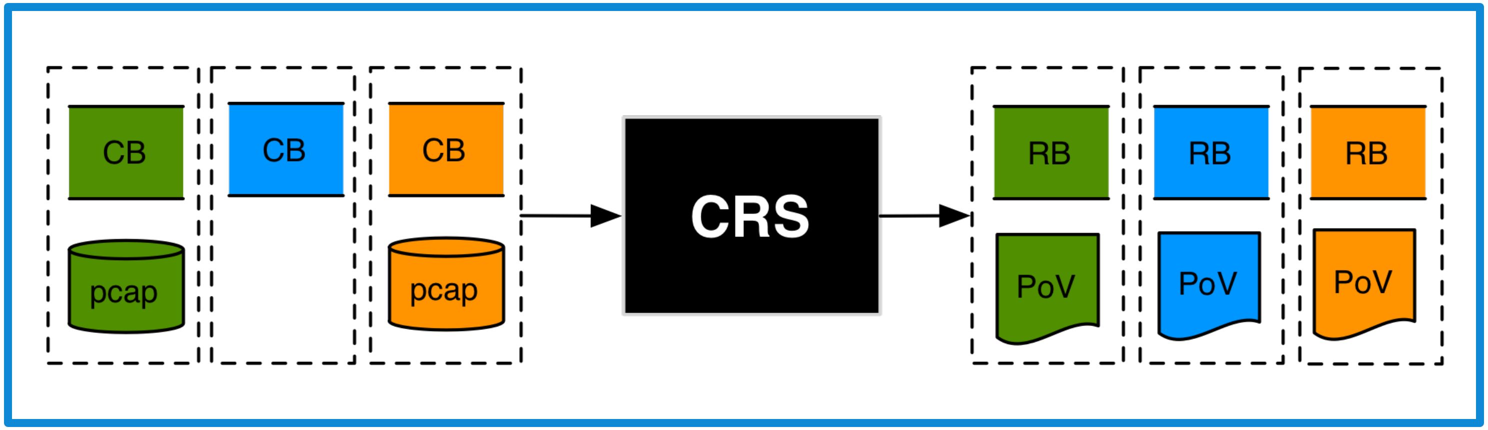


Real-World Incentives

How do we evaluate "patched" replacement binaries to encourage solutions that will stand up to real-world pressures?

Challenge for Infrastructure Team

Goal: Build a game that incentivizes improvements in automated program analysis and cyber reasoning



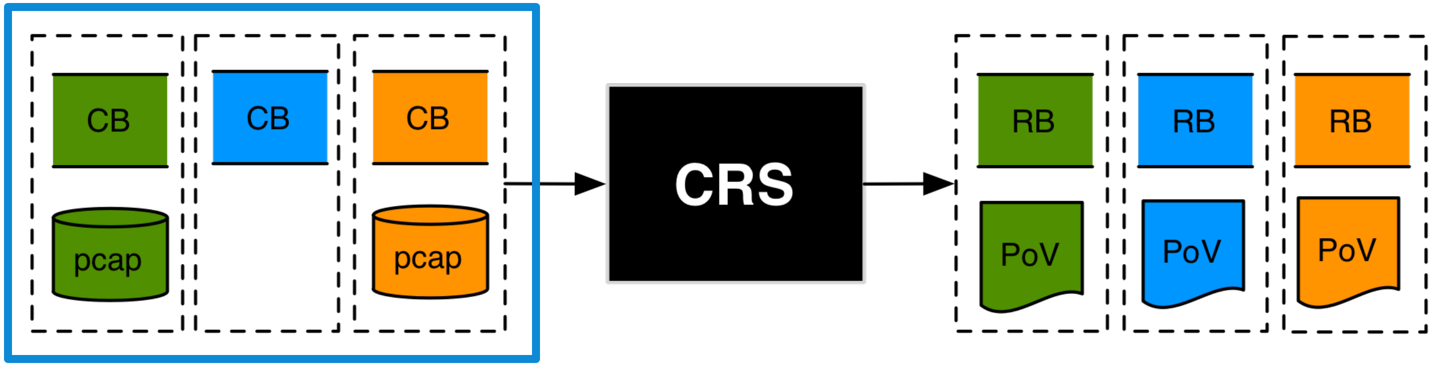
Repeatable, Scalable Experiments

How do we ensure measurement system is scalable, consistent, and robust?

Challenge for Infrastructure Team

Alternative Ecosystem

How do we make the test corpus representative of real-world challenges but not tainted by prior knowledge?



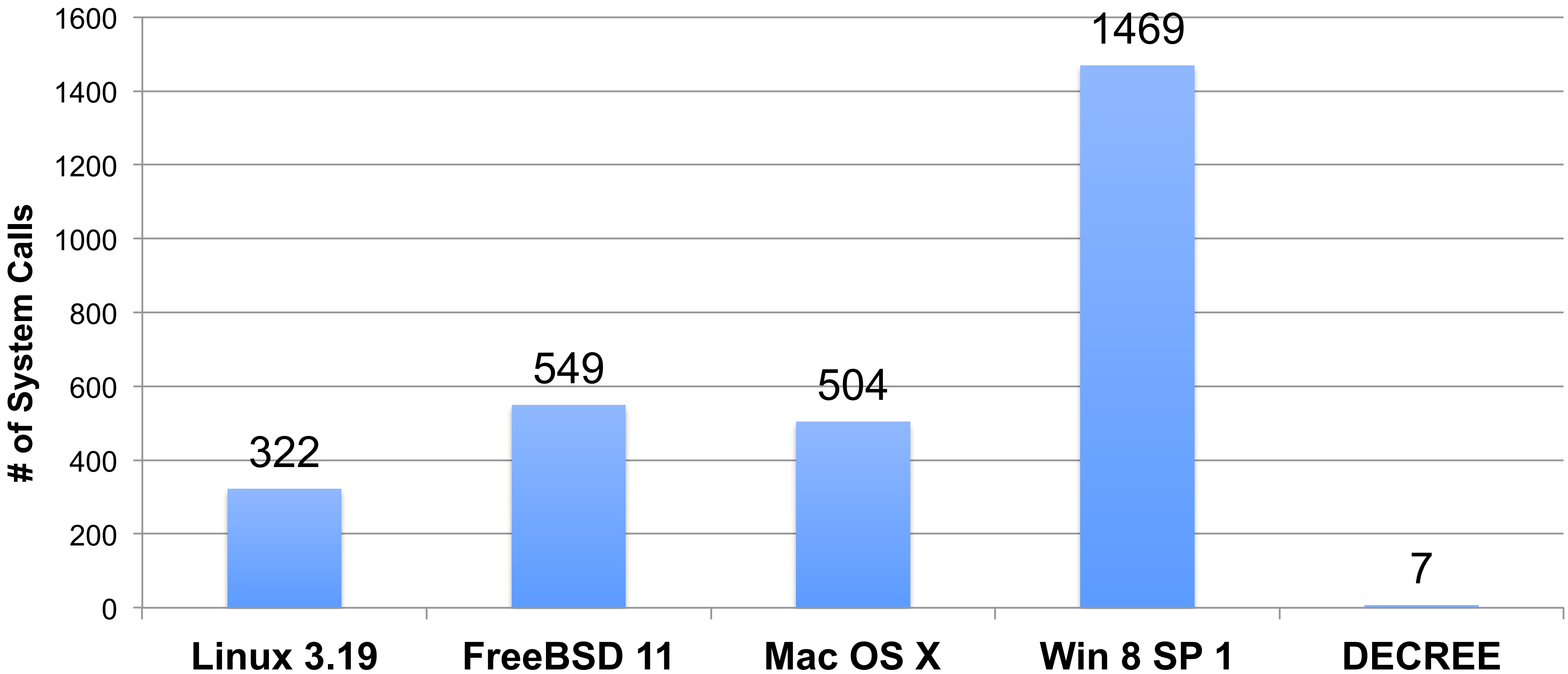
No known protocols
No code reuse

DECREE

DARPA Experimental Cyber Research Evaluation Environment

Syscall Name	Syscall #
`allocate`	5
`deallocate`	6
`transmit`	2
`receive`	3
`fdwait`	4
`random`	7
`_terminate`	1

Scoping the Problem: DECREE



Scoping the Problem: DECREE

Operating System	Process Creation	File System	Env Variables	Shared Libraries	Shared memory	Network sockets	IPC messaging	Sources of non-determinism
Linux	✔	✔	✔	✔	✔	✔	✔	✔
FreeBSD	✔	✔	✔	✔	✔	✔	✔	✔
Mac OS X	✔	✔	✔	✔	✔	✔	✔	✔
Win 8	✔	✔	✔	✔	✔	✔	✔	✔
DECREE	❖	✖	✖	✖	✖	❖	✔	❖

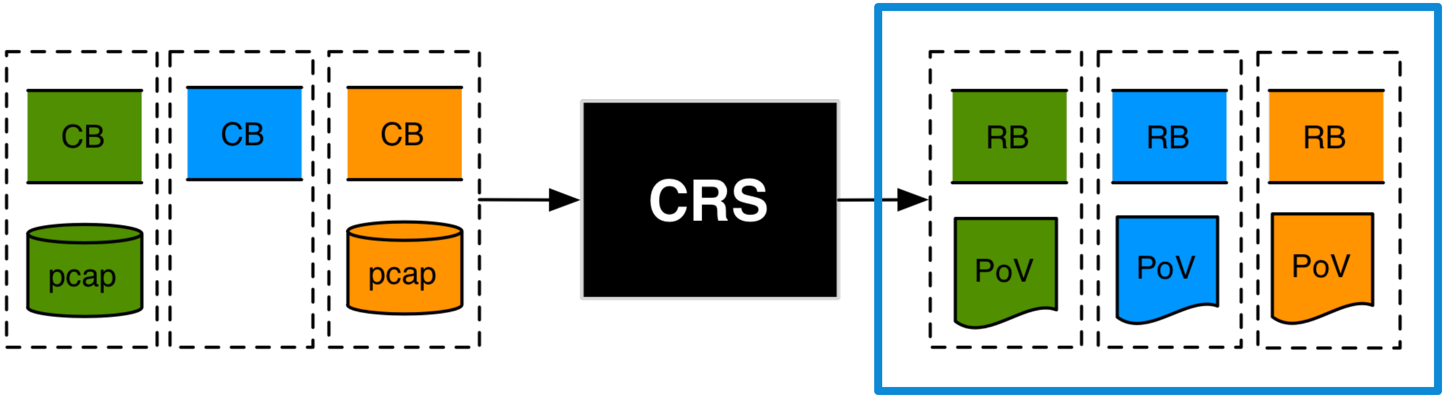
- ❖ Processes and network sockets are managed by CGC process launcher
- ❖ Only source of non-determinism is a pseudo-random number generator

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DECREE: Limited Scope, Limitless Possibilities

- Basic messaging application
- Simple particle physics simulator
- RAM-based filesystem
- Radio receiver after demodulation
- Spreadsheet program
- Basic virtual machine
- Industrial control system
- A new way to pay for everything
- …and 123 more…

Challenge for Infrastructure Team



Real-World Incentives

How do we evaluate "patched" replacement binaries to encourage solutions that will stand up to real-world pressures?

Real-World Constraints

- Security solutions should not break functionality
  - Antivirus delivers a false-positive detection…in SVCHOST.exe (https://support.microsoft.com/en-us/kb/2025695)

Real-World Constraints

- ❑ **Security solutions should not break functionality**

- ❑ **Significant performance degradation will not be tolerated**
  - **"CPU and memory cost below 5%" (<u>Microsoft BlueHat Contest</u>, "Practical and Functional" criterion)**

Real-World Constraints

- Security solutions should not break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
  - "Relying solely on perimeter defenses is now passé – and naïvely dangerous" (Kelly Jackson Higgins, "Damage Mitigation as the New Defense")

Real-World Constraints

- Security solutions should not break functionality
- Significant performance degradation will not be tolerated
- Security solutions must mitigate attacks
- Rewarding Proof of Vulnerability (PoV) discovery enables fixing bugs sooner
  - "SAGE found ~ 1/3 of all bugs found by file fuzzing in Windows 7" http://research.microsoft.com/en-us/um/people/pg/public_psfiles/cacm2012.pdf

Scoring Algorithm

SubScore(RB, PoV) = Availability(RB) × Security(RB, PoV) × Evaluation(PoV)

- ❑ Security solutions should not break functionality
- ❑ Significant performance degradation will not be tolerated
- ❑ Security solutions must mitigate attacks
- ❑ Rewarding PoV discovery enables fixing bugs sooner

Scoring Algorithm

$$\text{SubScore(RB, PoV)} = \underbrace{\boxed{\text{Availability(RB)}}}_{\boxed{\mathit{min}\text{(FuncFactor, PerfFactor)}}} \times \text{Security(RB, PoV)} \times \text{Evaluation(PoV)}$$

- ✓ Security solutions should not break functionality
- ✓ Significant performance degradation will not be tolerated
- ❑ Security solutions must mitigate attacks
- ❑ Rewarding PoV discovery enables fixing bugs sooner

Scoring Algorithm

$$\text{SubScore(RB, PoV)} = \text{Availability(RB)} \times \boxed{\text{Security(RB, PoV)}} \times \text{Evaluation(PoV)}$$

0	if no reference PoVs mitigated
1 + ½ × (Reference + Consensus)	otherwise

✓ Security solutions should not break functionality

✓ Significant performance degradation will not be tolerated

❑ Security solutions must mitigate attacks

❑ Rewarding PoV discovery enables fixing bugs sooner

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Scoring Algorithm

$$\text{SubScore(RB, PoV)} = \text{Availability(RB)} \times \boxed{\text{Security(RB, PoV)}} \times \text{Evaluation(PoV)}$$

$$\begin{cases} 0 & \text{if no reference PoVs mitigated} \\ 1 + \frac{1}{2} \times (\boxed{\text{Reference}} + \boxed{\text{Consensus}}) & \text{otherwise} \end{cases}$$

$$\frac{\text{\# reference PoVs mitigated}}{\text{\# reference PoVs}}$$

$$\begin{cases} 0 & \text{if } \textit{any} \text{ competitor PoV succeeded} \\ 1 & \text{otherwise} \end{cases}$$

- ✓ Security solutions should not break functionality
- ✓ Significant performance degradation will not be tolerated
- ✓ Security solutions must mitigate attacks
- ❑ Rewarding PoV discovery enables fixing bugs sooner

Scoring Algorithm

SubScore(RB, PoV) = Availability(RB) × Security(RB, PoV) × $\boxed{\text{Evaluation(PoV)}}$

1	if PoV was unsuccessful
2	if PoV was successful

- ✓ Security solutions should not break functionality
- ✓ Significant performance degradation will not be tolerated
- ✓ Security solutions must mitigate attacks
- ✓ Rewarding PoV discovery enables fixing bugs sooner

Measurement Framework

SubScore(RB, PoV) = Availability(RB) × Security(RB, PoV) × Evaluation(PoV)

Availability(RB)	Security(RB, PoV)	Evaluation(PoV)
min(FuncFactor, PerfFactor)	0 if no reference PoVs mitigated 1 + ½ × (Reference + Consensus)	1 if PoV unsuccessful 2 if PoV successful

- Availability (FuncFactor): How many tests does the service pass?
- Availability (PerfFactor): How much CPU and memory does the service use?
- Security (Reference): Do reference PoVs crash the replacement service?
- Security (Consensus): Do submitted PoVs crash the replacement service?
- Evaluation: Does submitted PoV crash the vulnerable service?

Measurement Framework

Beware the cracks in the abstraction layer

- **Availability (FuncFactor): How many tests does the service pass?**
- **Availability (PerfFactor): How much CPU and memory does the service use?**
- **Security (Reference): Do reference PoVs crash the replacement service?**
- **Security (Consensus): Do submitted PoVs crash the replacement service?**
- **Evaluation: Does submitted PoV crash the vulnerable service?**

Evaluating PoVs: Echo Service

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Evaluating PoVs: Proof of Vulnerability

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

- **Accessing unmapped memory (SIGSEGV)**
- **Executing illegal instruction (SIGILL)**

Evaluating PoVs: Audience Poll

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Send 128 bytes to this service – does it crash?

Evaluating PoVs: Audience Poll

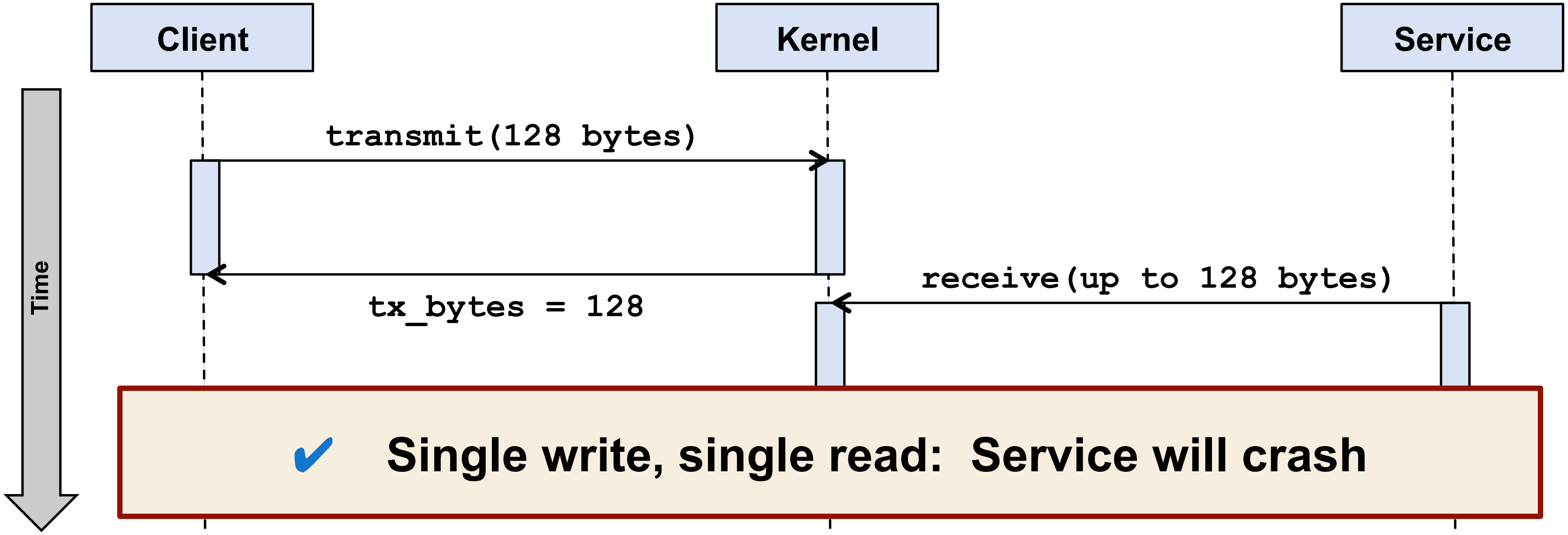
```c
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Send 128 bytes to this service – does it crash?

...

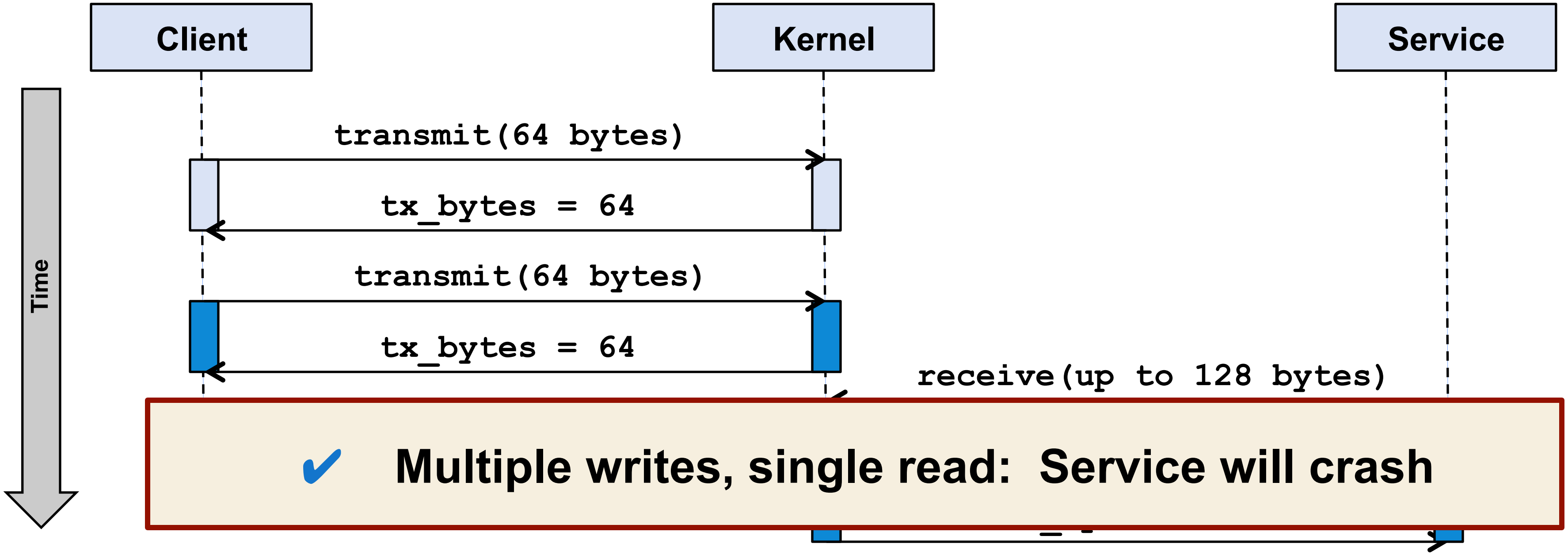
That depends on packet size and timing!

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```



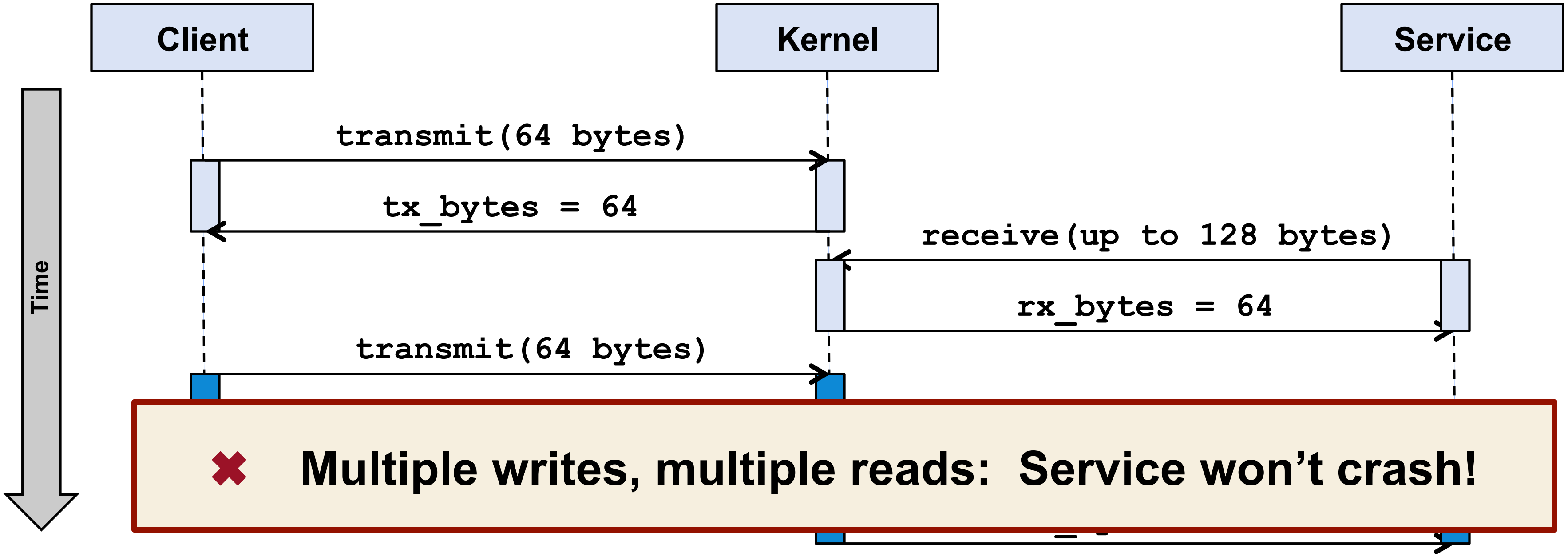
✔ Single write, single read: Service will crash

```c
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```



✔ **Multiple writes, single read: Service will crash**

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```



✖ **Multiple writes, multiple reads: Service won't crash!**

Evaluating PoVs

```
void echo() {
    char buf[64];
    while (receive(STDIN, &buf, 128, NULL) == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

Send 128 bytes to this service – does it crash?

…

That depends on packet size and timing!

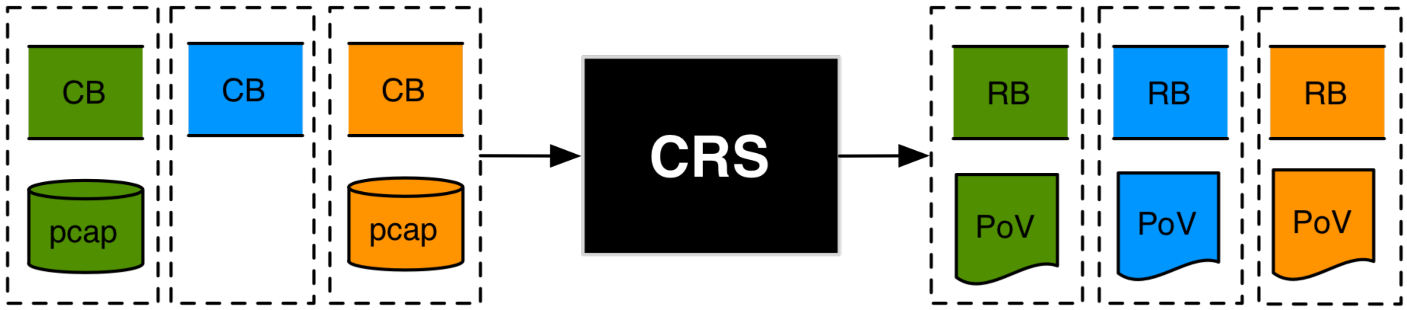
This behavior is non-deterministic!

Evaluating PoVs: High(er) Reliability

```
void echo() {
    char buf[64];
    while (receive_until(STDIN, &buf, '\n') == 0) {
        transmit(STDOUT, &buf, 64, NULL);
    }
}
```

- Change CB to avoid non-deterministic behavior
- Verify reference PoVs and polls work with different write chunk sizes and random seeds
- Re-run competitor PoVs several times with different random seeds; if it's ever successful, count as successful evaluation

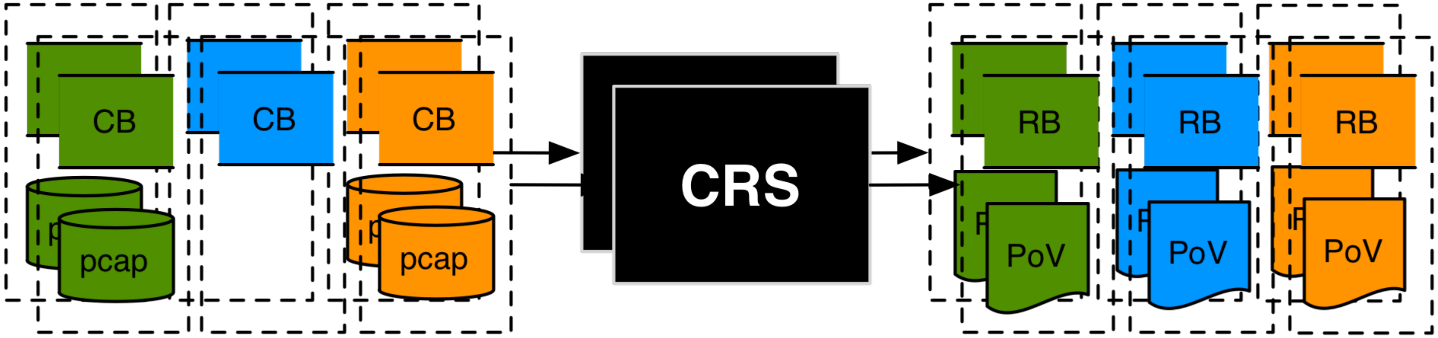
Challenge for Infrastructure Team



Repeatable, Scalable Experiments

How do we ensure the measurement system is scalable, consistent, and robust?

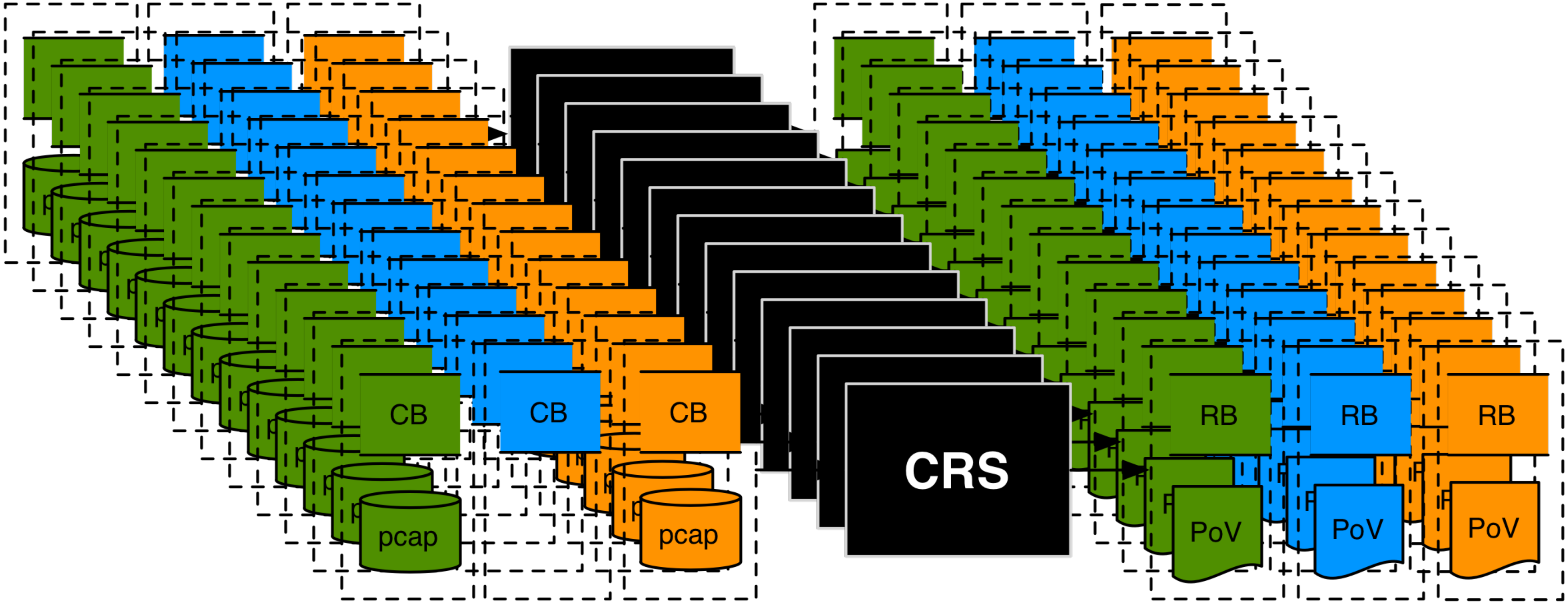
Challenge for Infrastructure Team



Repeatable, Scalable Experiments

How do we ensure the measurement system is scalable, consistent, and robust?

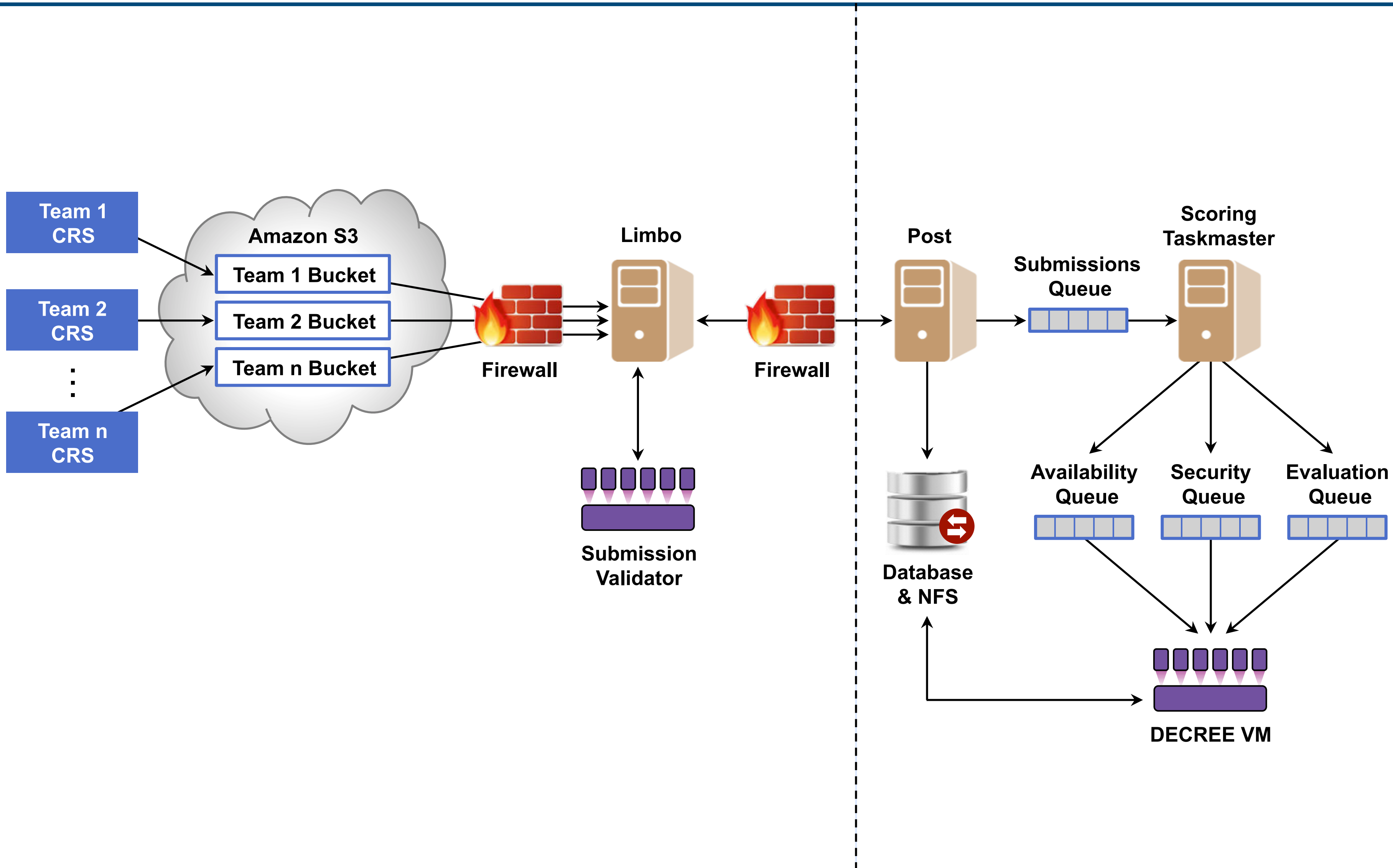
Challenge for Infrastructure Team



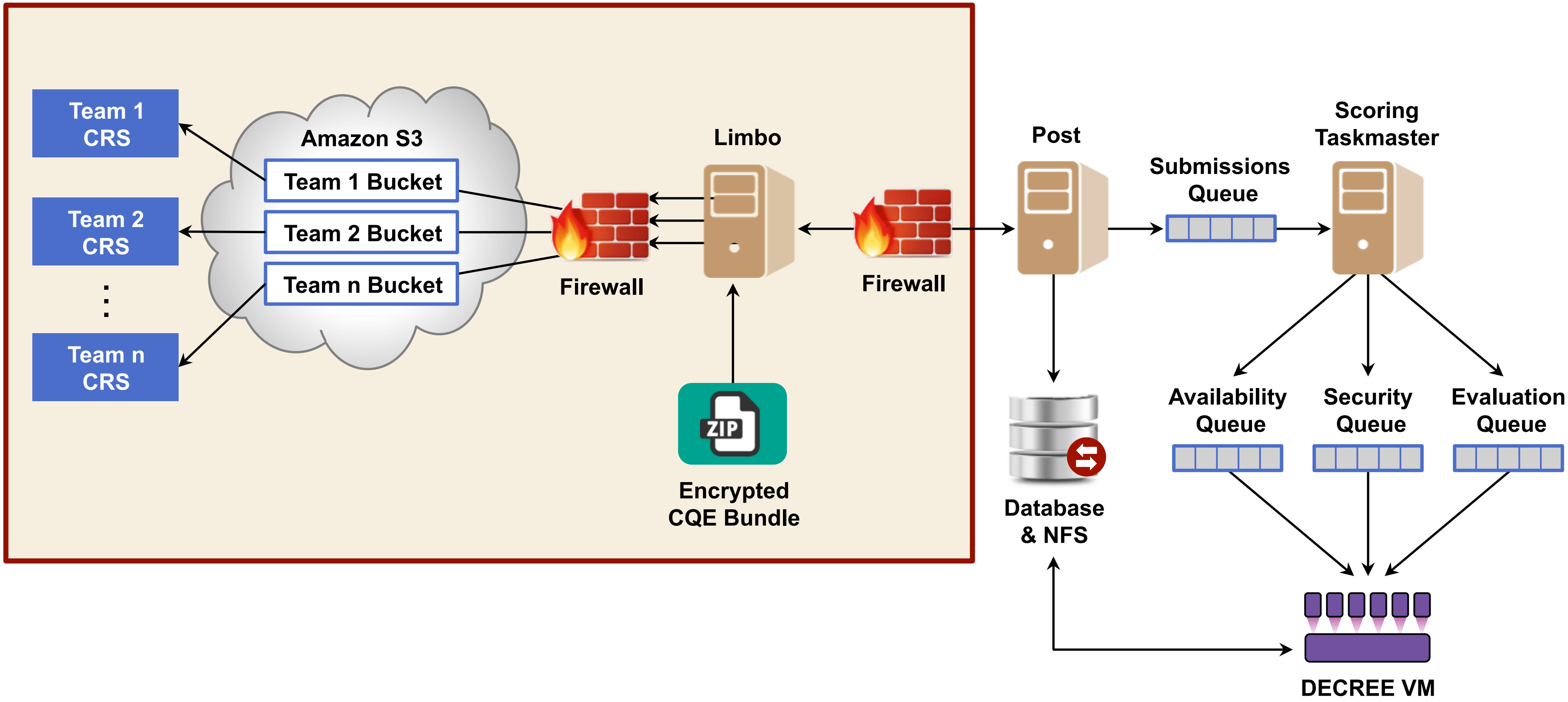
Repeatable, Scalable Experiments

How do we ensure the measurement system is scalable, consistent, and robust?

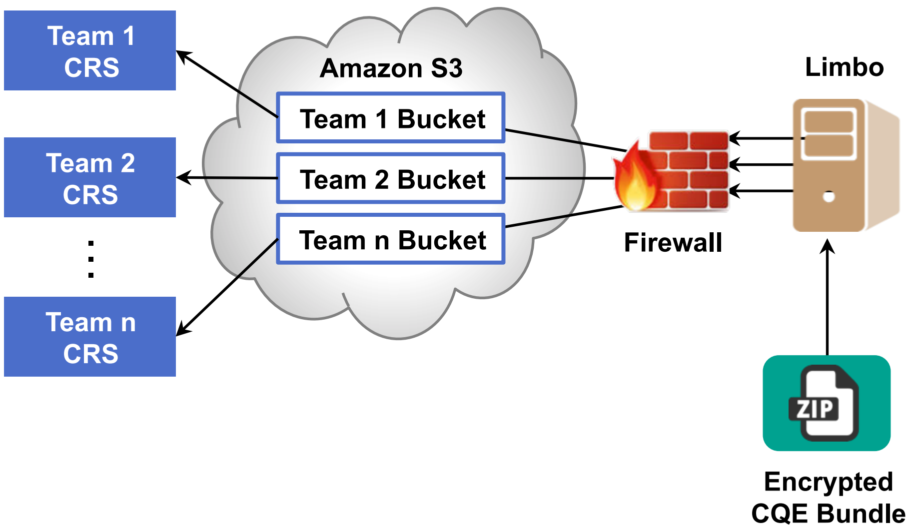
CQE Scoring System Architecture



CQE Distribution System

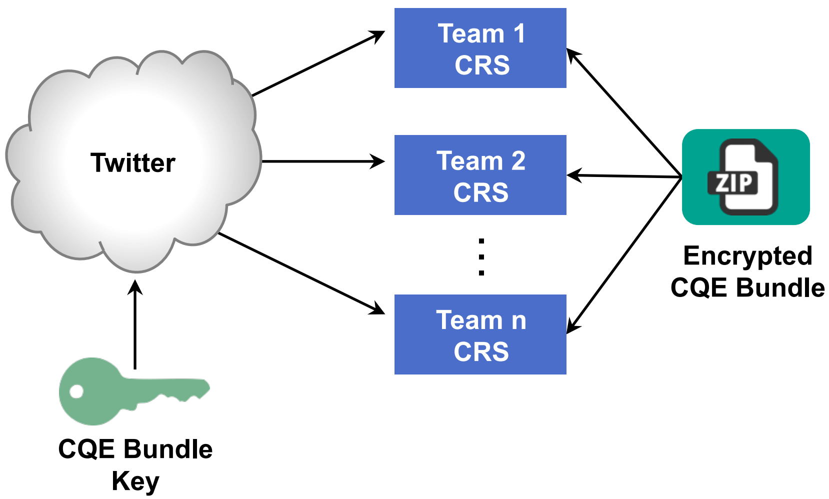


CQE Distribution System



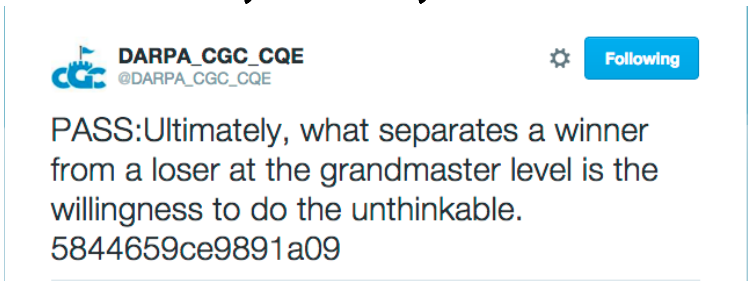
- ❑ Each team has equal access to the challenge bundle
- ❑ Contents of the challenge bundle remain secret until CQE begins

CQE Distribution System

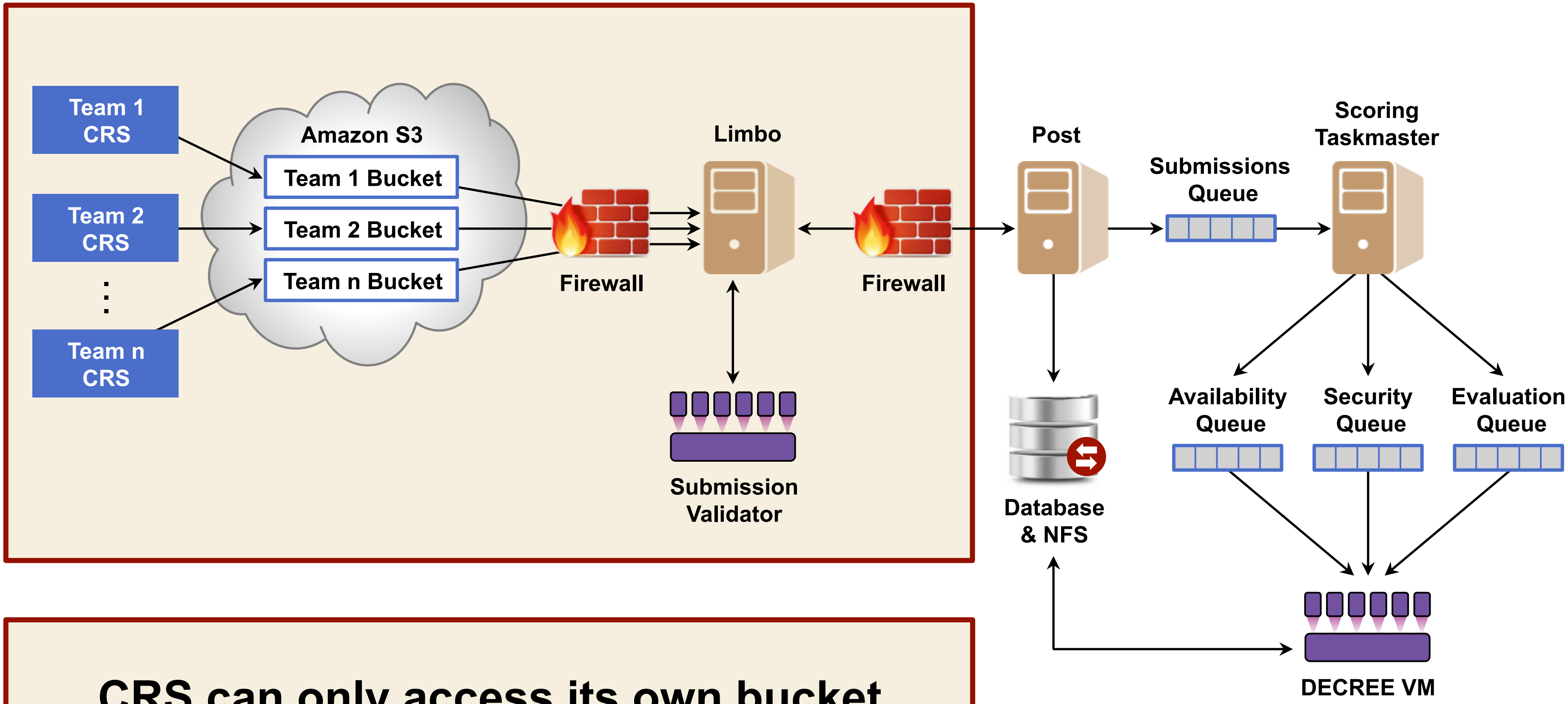


- ✓ Each team has equal access to the challenge bundle
- ✓ Contents of the challenge bundle remain secret until CQE begins

Key distribution: Twitter, SMS, Email

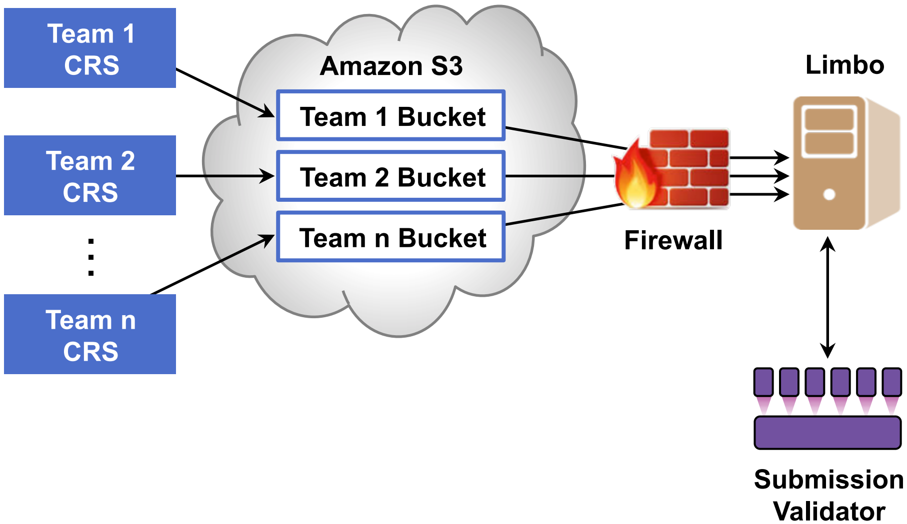


CQE Submission System



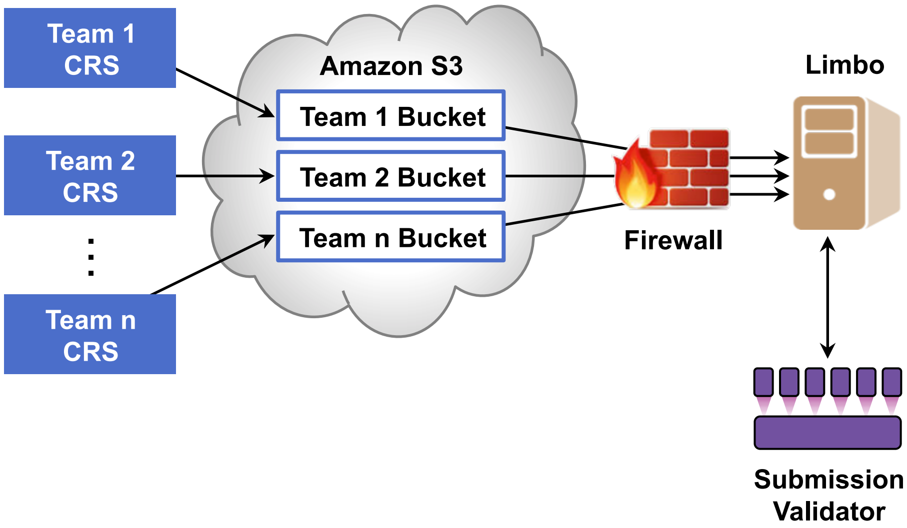
CRS can only access its own bucket

CQE Submission System



- ❑ Submission system must be "high-availability"
- ❑ Each submission must be time-stamped
- ❑ Only scoring system can read submissions
- ❑ Submission uploads must be atomic operations
- ❑ Submission uploads must be allowed only during CQE

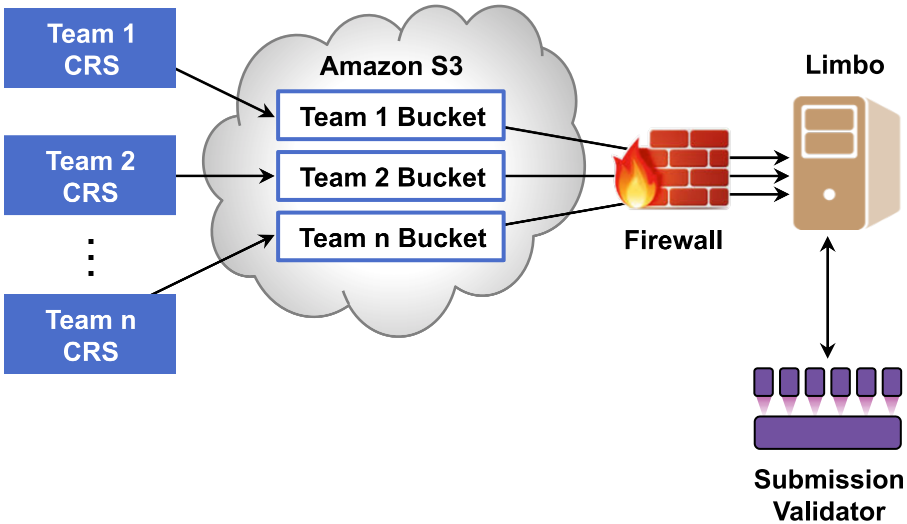
CQE Submission System



✓ Submission system must be "high-availability"

Submission system: Amazon S3

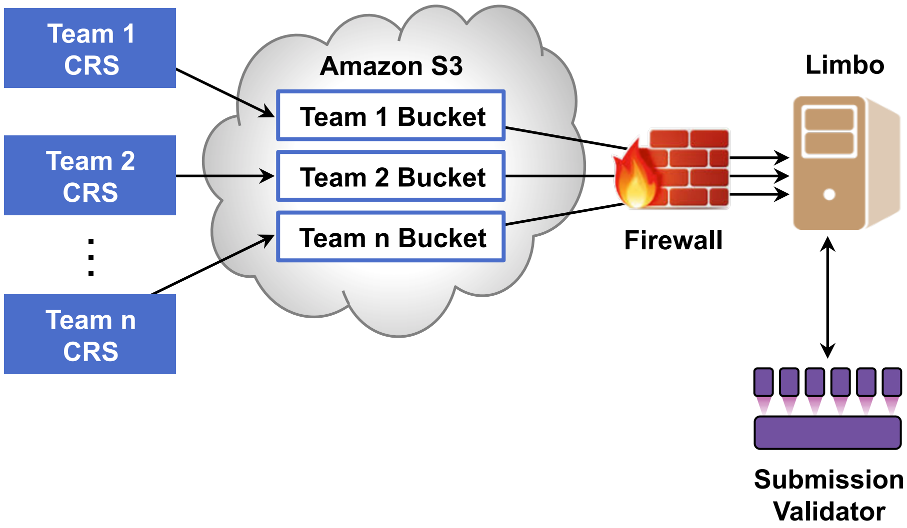
CQE Submission System



- ✓ Submission system must be "high-availability"
- ✓ Each submission must be time-stamped

Submission system: Amazon S3

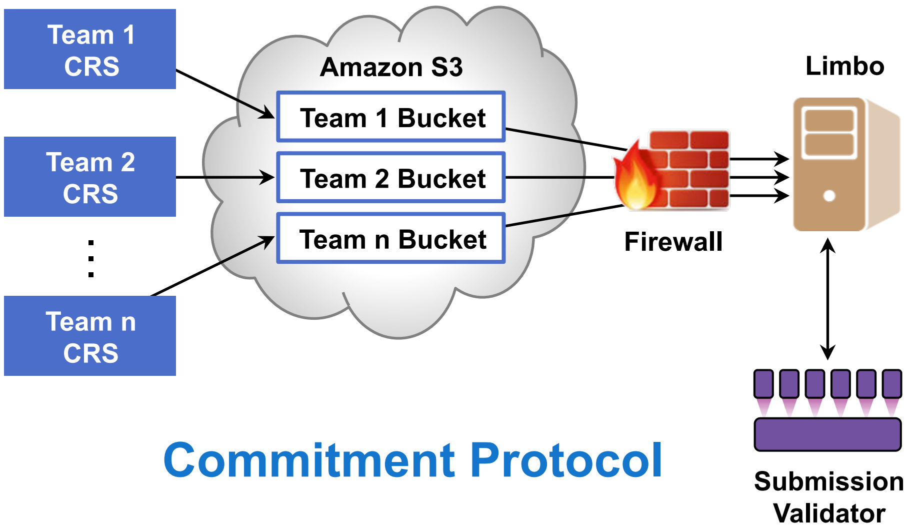
CQE Submission System



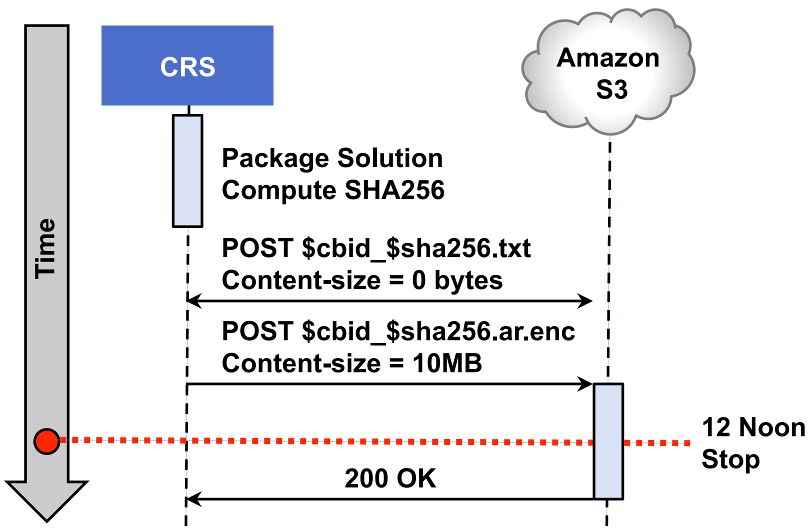
- ✓ Submission system must be “high-availability”
- ✓ Each submission must be time-stamped
- ✓ Only scoring system can read submissions

Encrypt submissions using pre-shared, per-team keys

CQE Submission System

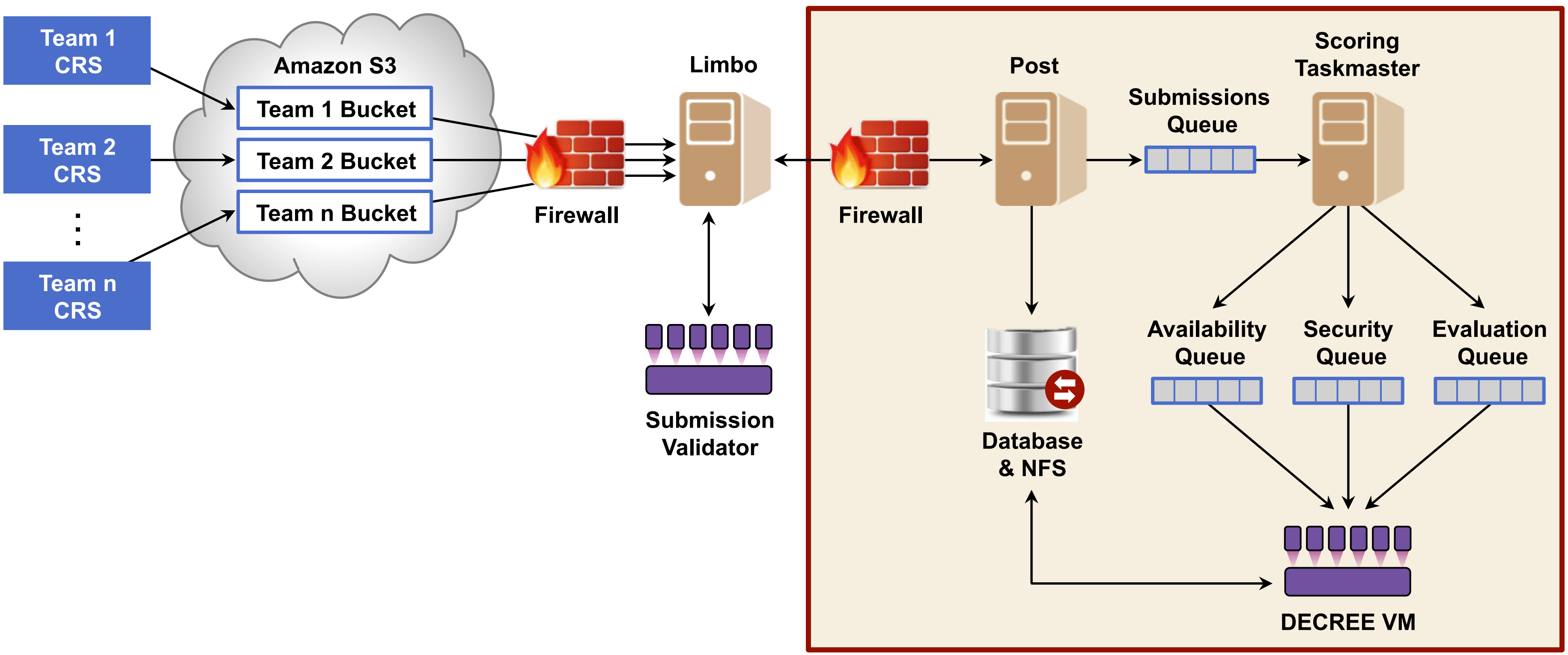


Commitment Protocol



- ✓ Submission system must be “high-availability”
- ✓ Each submission must be time-stamped
- ✓ Only scoring system can read submissions
- ✓ Submission uploads must be atomic operations
- ✓ Submission uploads must be allowed only during CQE

CQE Scoring System Architecture



LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

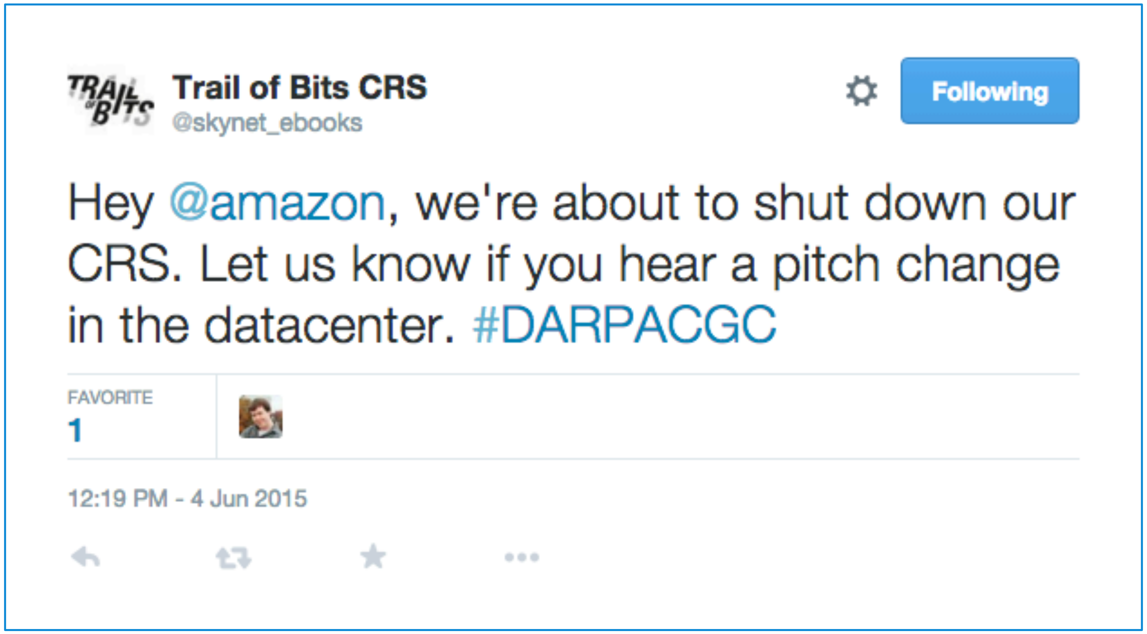
CQE Scoring System Statistics

Type of Test	Number of Tests
Functionality and Performance	16,167,316
Reference Security	186,720
Consensus Security	438,760
Proof of Vulnerability Evaluation	52,600
Total Tests	**16,845,396**

Hosts	Total RAM (GB)	Total CPU Cores	DECREE VMs	Time to Provision	Time to Score
12	3,072	240	192	40 min	**16 hours**

Results confirmed by independent (HW and SW) scoring system

CQE Results


Trail of Bits CRS
@skynet_ebooks
Following
Hey @amazon, we're about to shut down our CRS. Let us know if you hear a pitch change in the datacenter. #DARPACGC
FAVORITE
1
12:19 PM - 4 Jun 2015

CYBER GRAND_CHALLENGE

Qualifying Event

131 Challenge Sets

590 Embedded Vulnerabilities

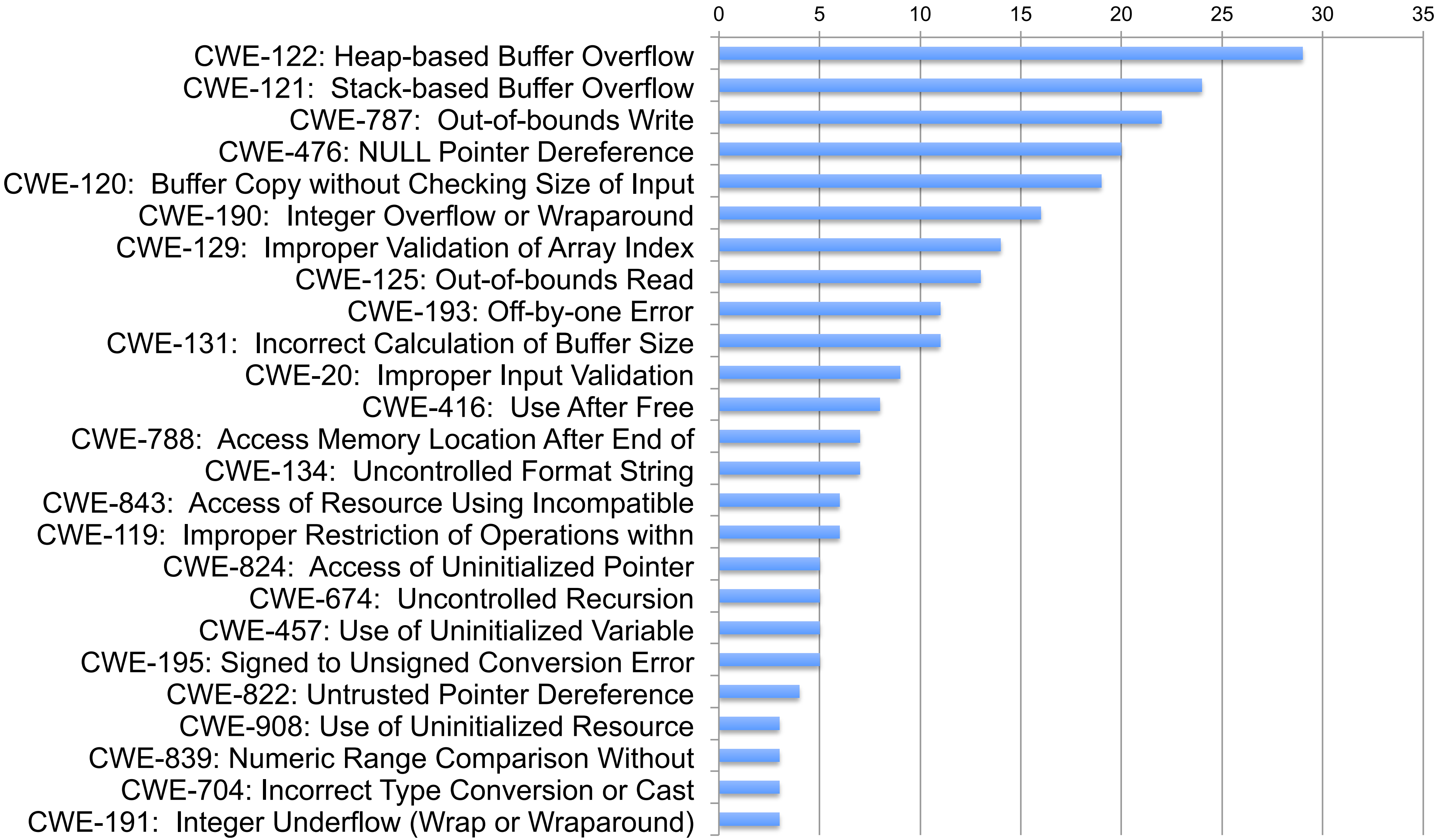
6,000+ Functions
190,000+ LOC .h
7,000+ LOC .cc
200,000+ LOC .c

5 IPC Challenge Sets

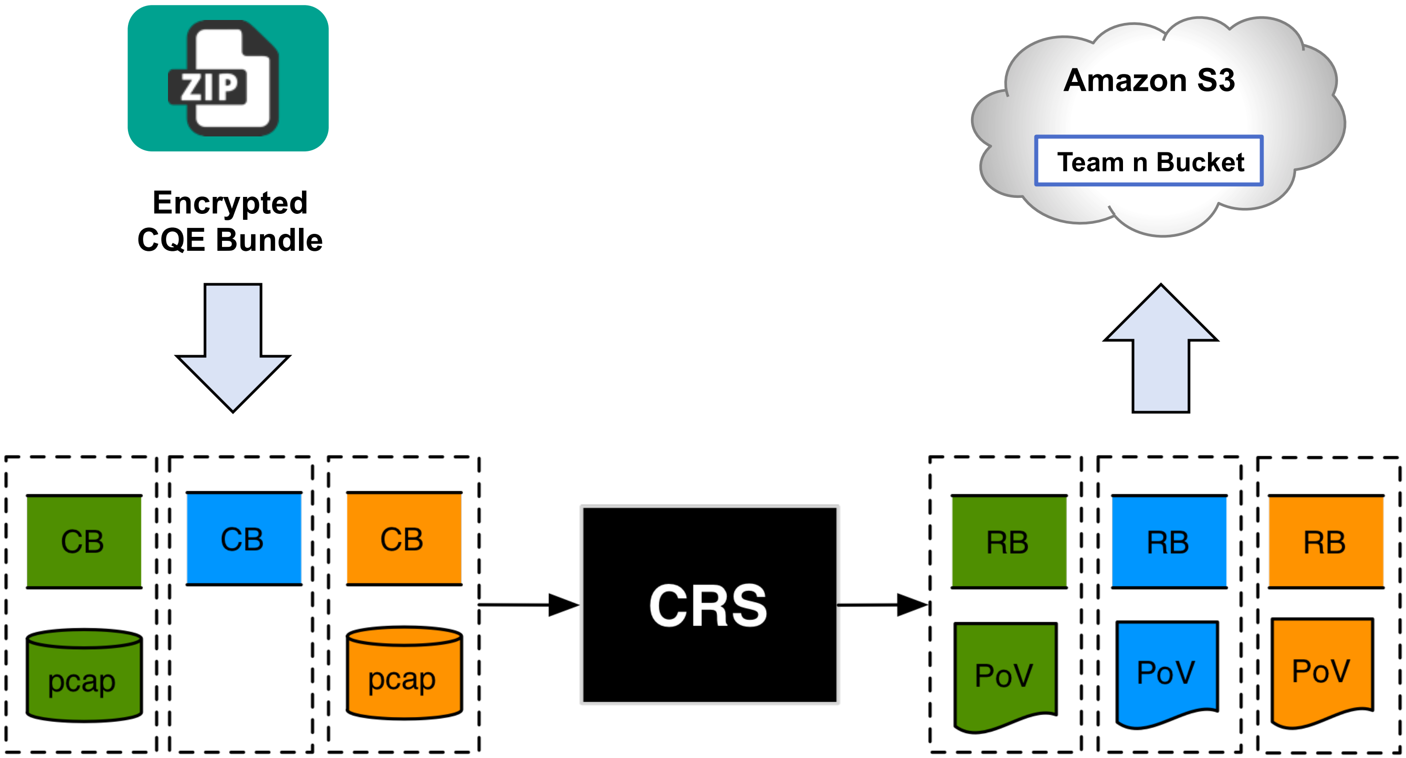
50+ Distinct CWEs

CWE = Common Weakness Enumeration IPC = Inter-Process Communication LOC = Lines of Code

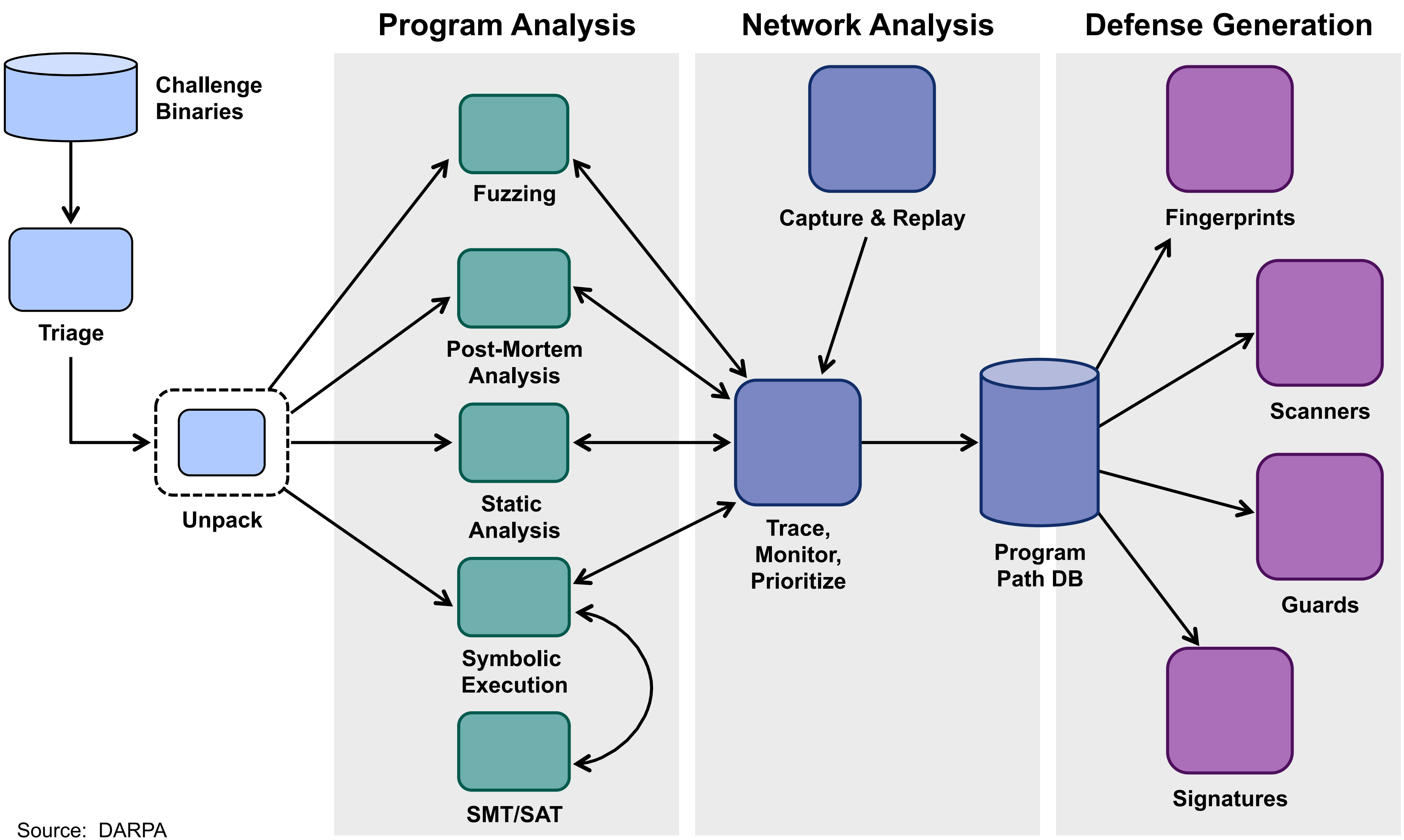
Common Weaknesses in CQE



CQE from Perspective of CRS



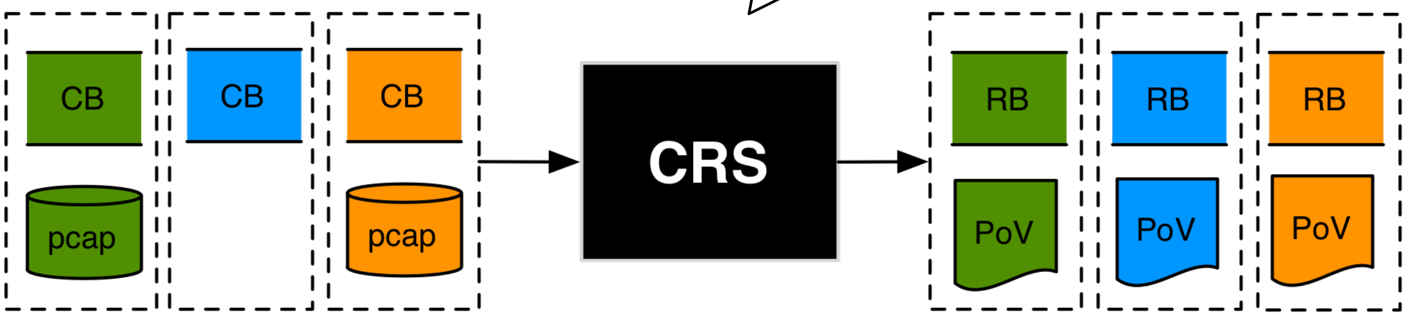
Grand Challenge for CRS Creators



Source: DARPA

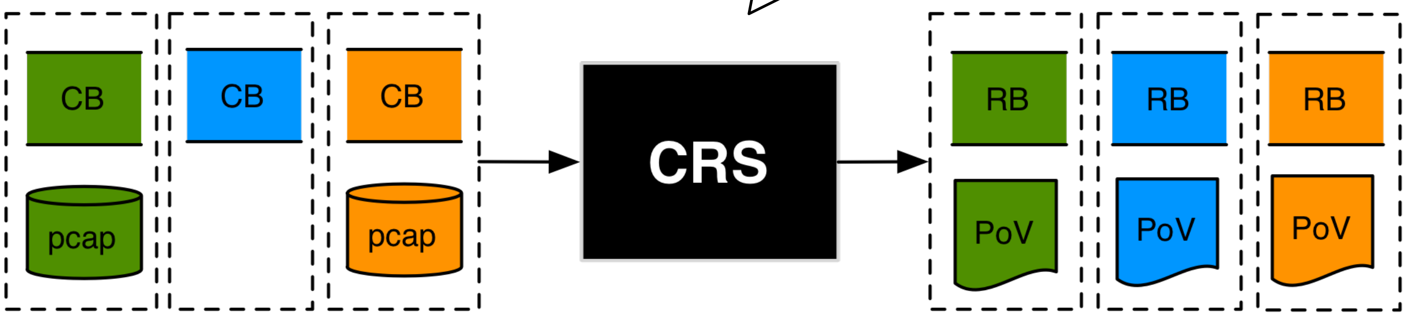
CRS Strategy 1: Fuzz and Fix

- Look at network traffic capture
- Fuzz to find a PoV
- Patch the observed crash



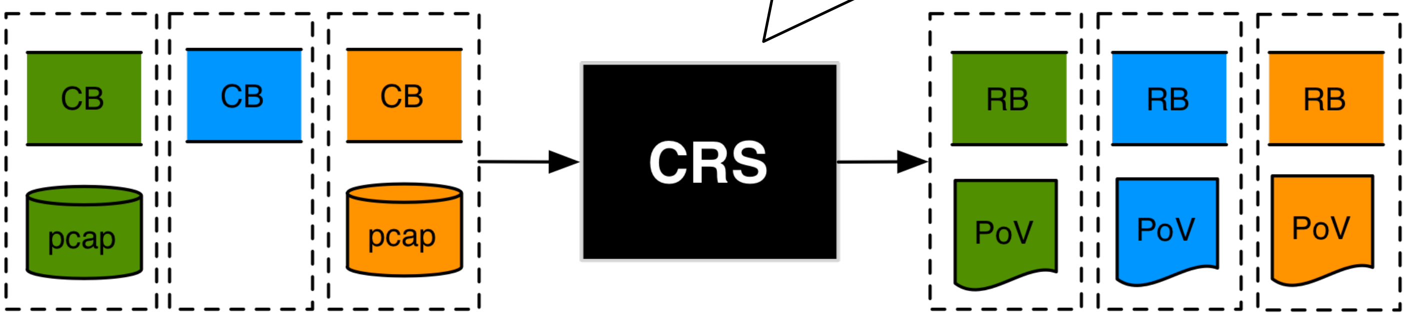
CRS Strategy 2: Generic Hardening

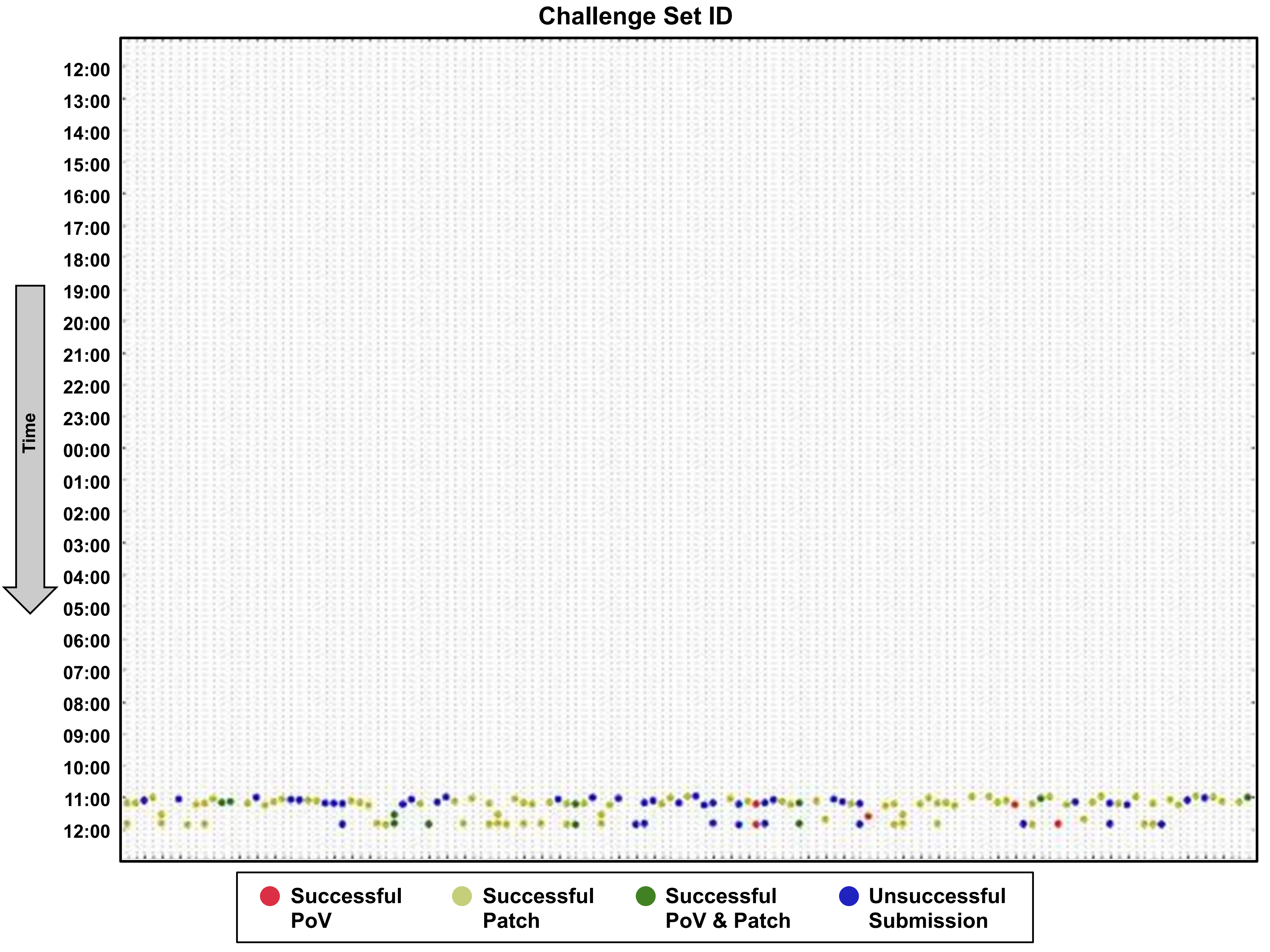
- **Analyze CB for possible memory corruption**
- **Patch to validate pointers before memory access**

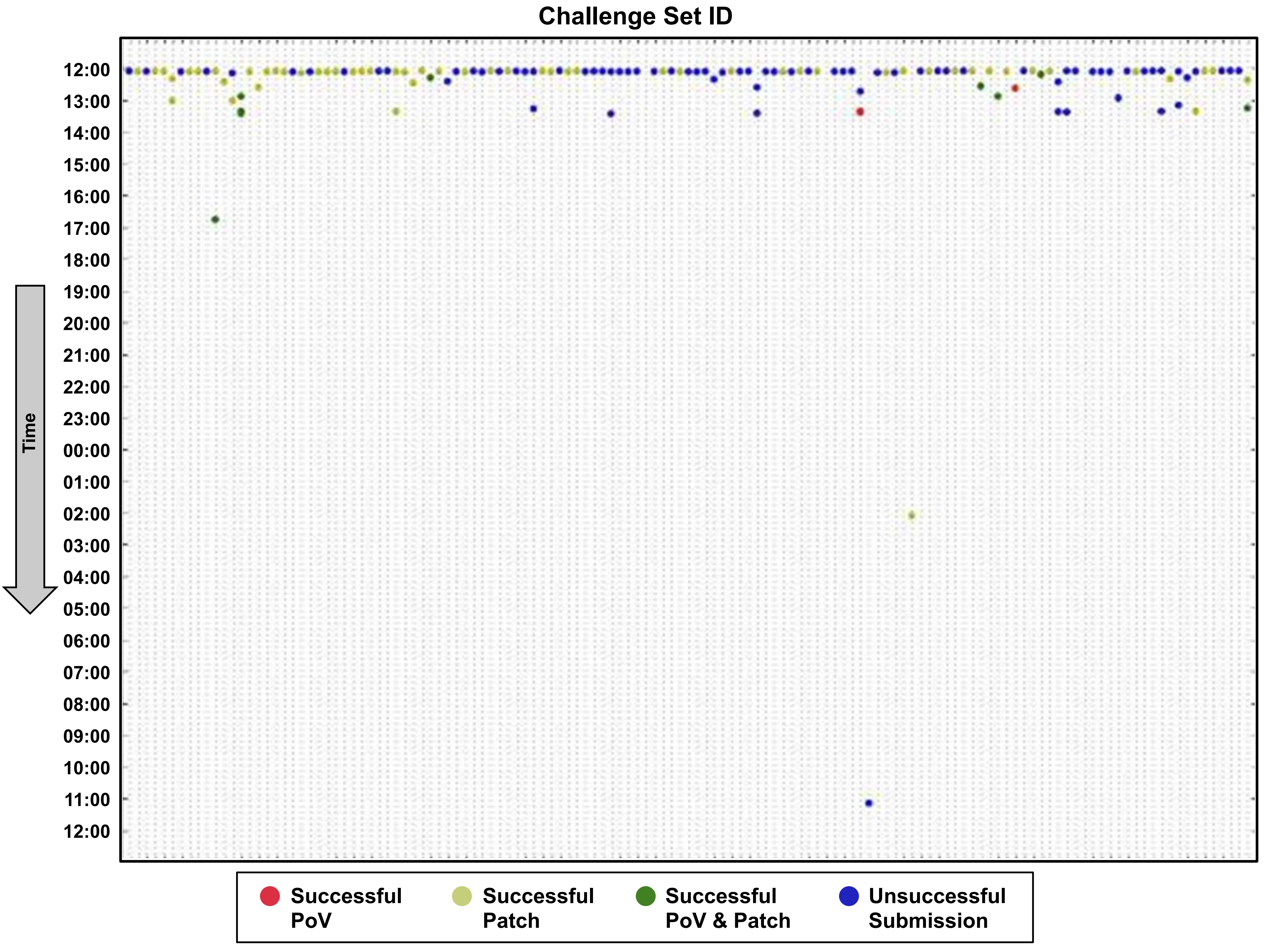


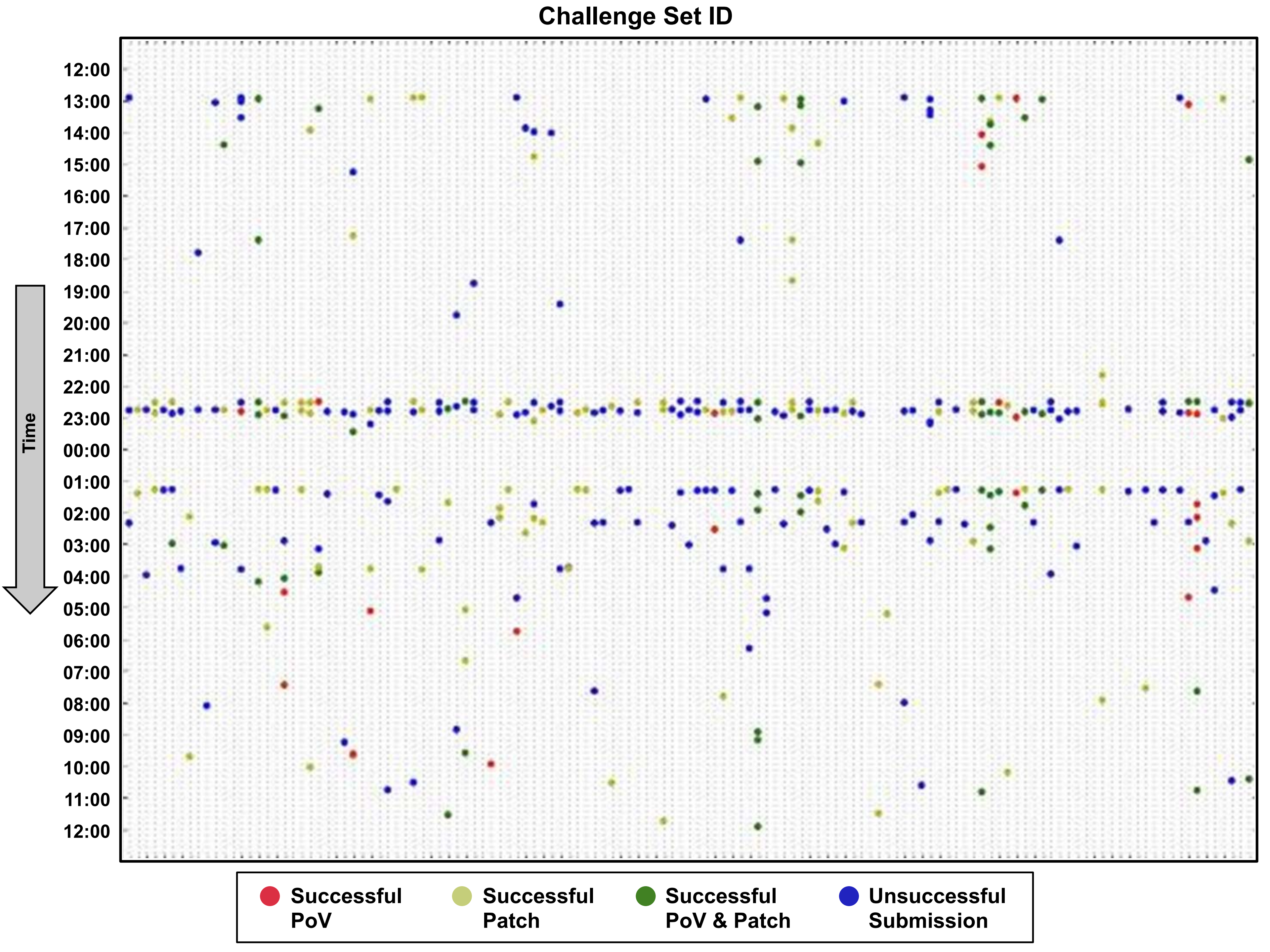
CRS Strategy 3: Symbolic Execution

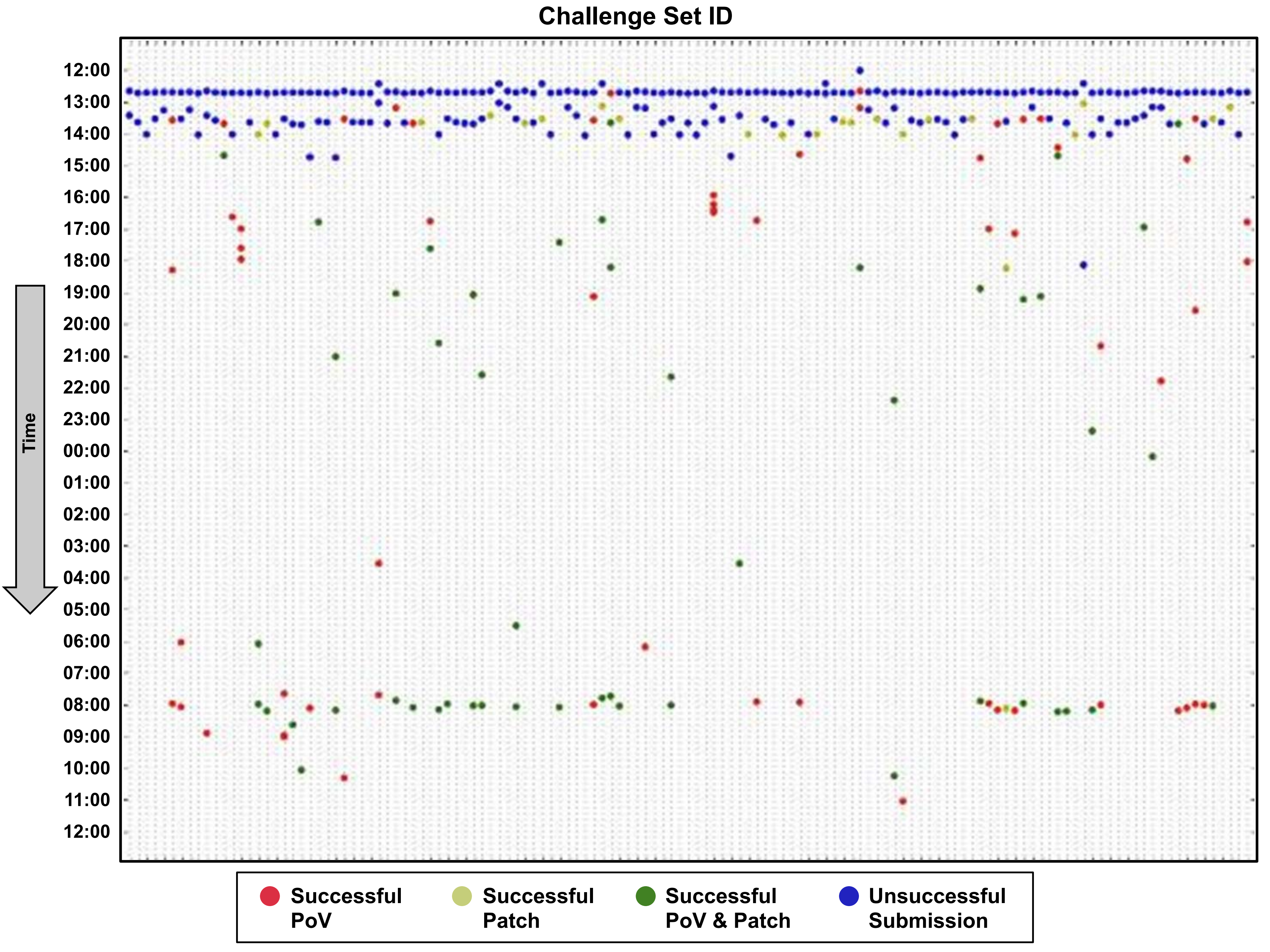
- Symbolically execute CB to collect path constraints
- Solve for possible memory corruption
- Verify via concrete execution
- Patch confirmed crash sites



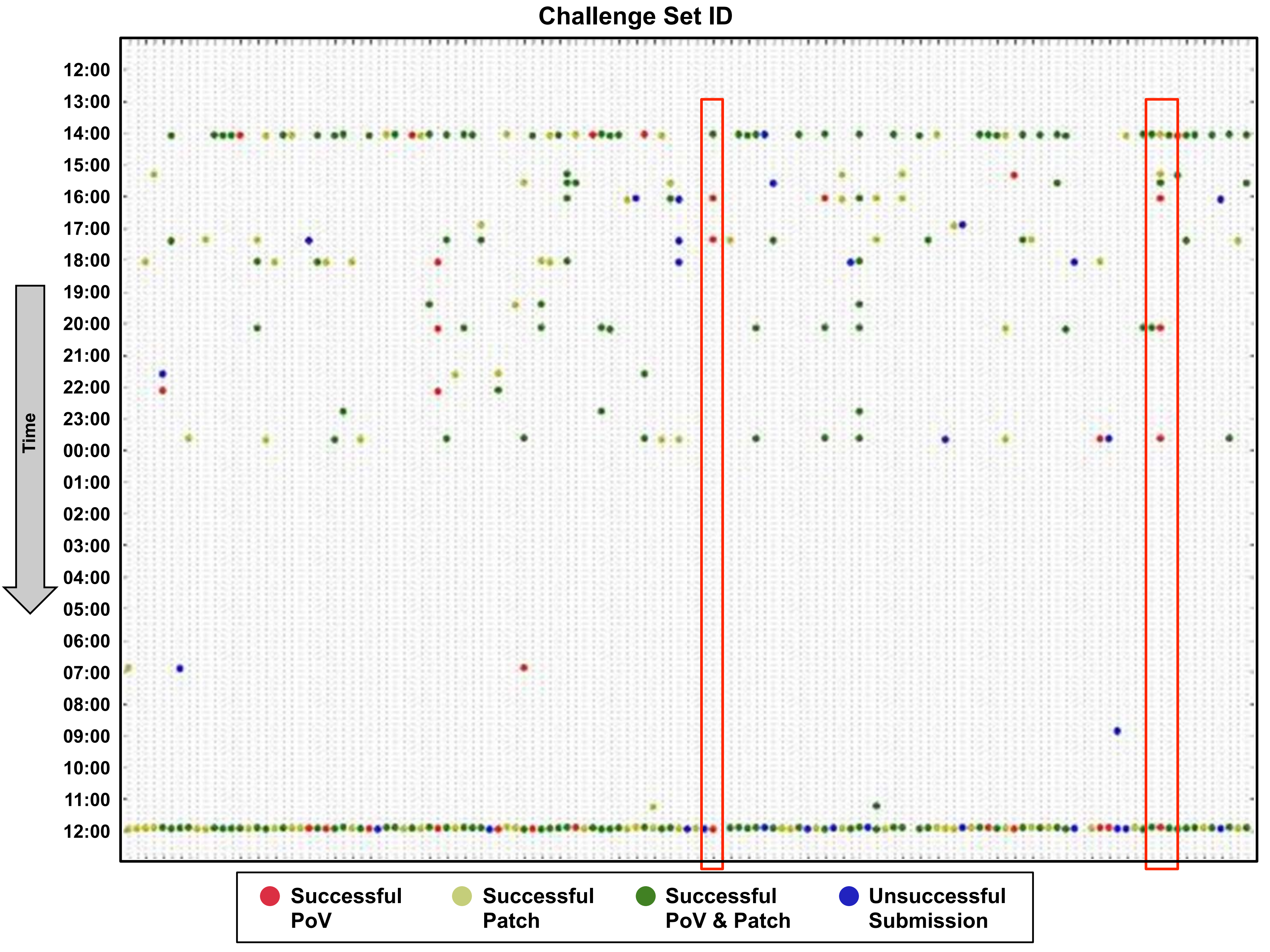

Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission


Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission

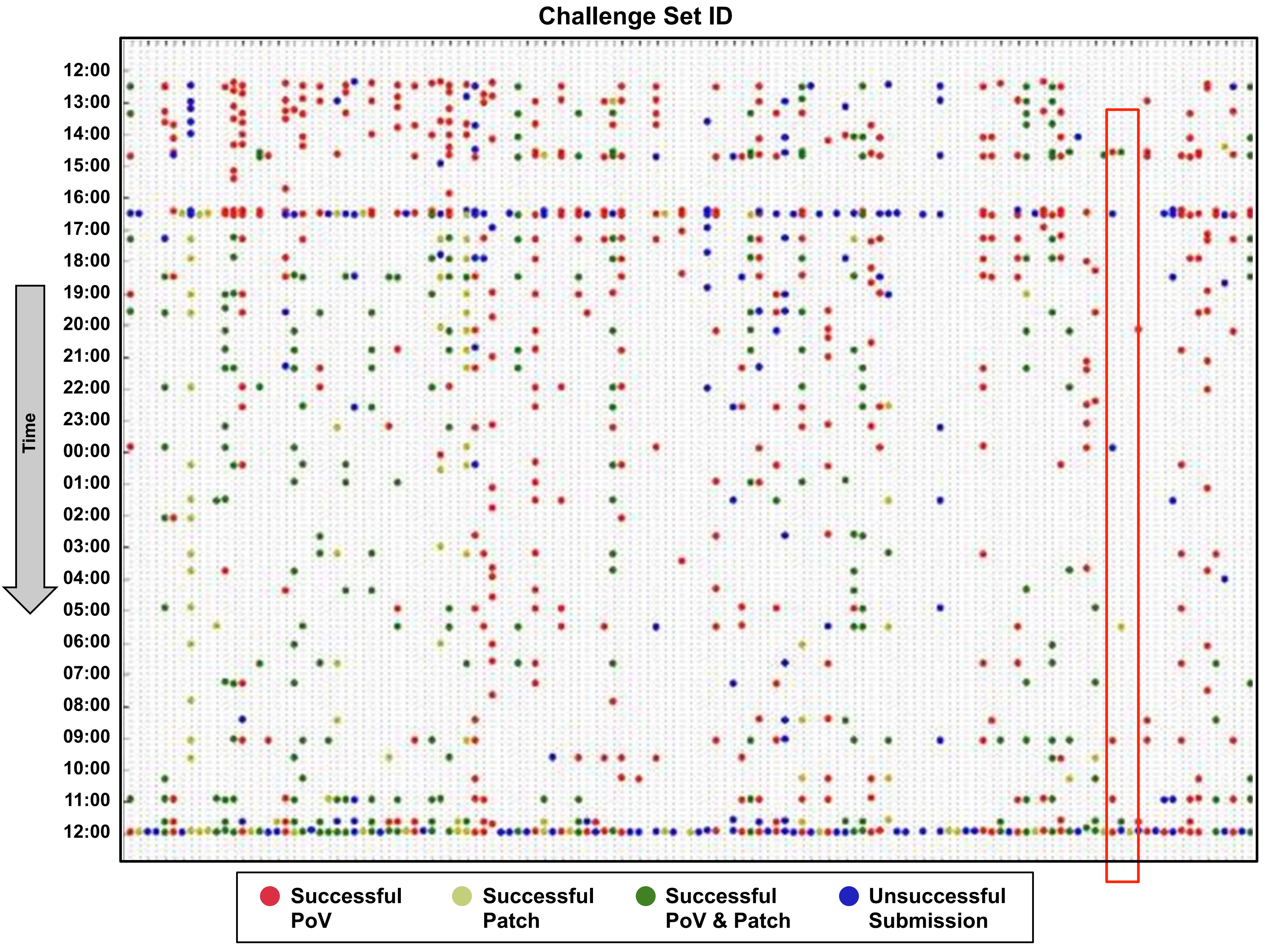

Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission



Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission



Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission


Challenge Set ID
Time
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
Successful PoV
Successful Patch
Successful PoV & Patch
Unsuccessful Submission

Example Challenge: YAN01_00012

- **A simple stack-based machine that uses 32-bit words**
  - **3 bits for opcode**
  - **29 bits for immediate values**

- **Vulnerability is a missing check on writes to a heap-allocated buffer**

- **This challenge was designed to test a CRS' ability to**
  - **Monitor the heap (`allocate` and `deallocate` system calls)**
  - **Support bit-wise operations for dependency analysis**
  - **Identify VM instructions that can be used to cause a VM stack overflow and patch them**

YAN01_00012 Instruction Set

Opcode	Instruction	Description
00	PUSH n	Push specified 32-bit number onto the stack
01	POP	Pop a 32-bit number from the stack
02	PUSHPC	Push program counter onto the stack
03	JMPZ	Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value
04	SWAP n	Swap nth stack entry with the top one
05	DUP n	Duplicate nth stack entry and push it to the top of the stack
06	ADD	Pop top two numbers off of the stack, add them, and push the sum back onto the stack
07	SUB	Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack
FFFFFFFF	RET	End of instruction stream – the topmost value on stack is the return value

YAN01_00012 Instruction Set

Opcode	Instruction	Description
00	PUSH n	Push specified 32-bit number onto the stack
01	POP	Pop a 32-bit number from the stack
02	PUSHPC	Push program counter onto the stack
03	JMPZ	Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value
04	SWAP n	Swap nth stack entry with the top one
05	DUP n	Duplicate nth stack entry and push it to the top of the stack
06	ADD	Pop top two numbers off of the stack, add them, and push the sum back onto the stack
07	SUB	Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack
FFFFFFFF	RET	End of instruction stream – the topmost value on stack is the return value

YAN01_00012 Instruction Set

Opcode	Instruction	Description
00	PUSH n	Push specified 32-bit number onto the stack
01	POP	Pop a 32-bit number from the stack
02	PUSHPC	Push program counter onto the stack
03	JMPZ	Pop two 32-bit values off the stack; if the first one is equal to 0, jump to the second value
04	SWAP n	Swap nth stack entry with the top one
05	DUP n	Duplicate nth stack entry and push it to the top of the stack
06	ADD	Pop top two numbers off of the stack, add them, and push the sum back onto the stack
07	SUB	Pop the top two numbers off of the stack, subtract them, and push the difference back onto the stack
FFFFFFFF	RET	End of instruction stream – the topmost value on stack is the return value

First Proof of Vulnerability Attempt (Unsuccessful)

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE pov SYSTEM "/usr/share/cgc-docs/replay.dtd">
<pov>
    <cbid>YAN01_00012</cbid>
    <replay>
        <delay>500</delay>
        <write>
            <data>
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                ...
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
            </data>
        </write>
    </replay>
</pov>
```

4096 bytes

Second Proof of Vulnerability Attempt (Successful)

```
<write>
  <data format="hex">
    000000000
    000000000
    000000000
    ffffff7f
    000000000
    050000000
    000000000
    000000000
    030000000
    ...
    ffffffff
    ...
  </data>
</write>
```

Resulting Program

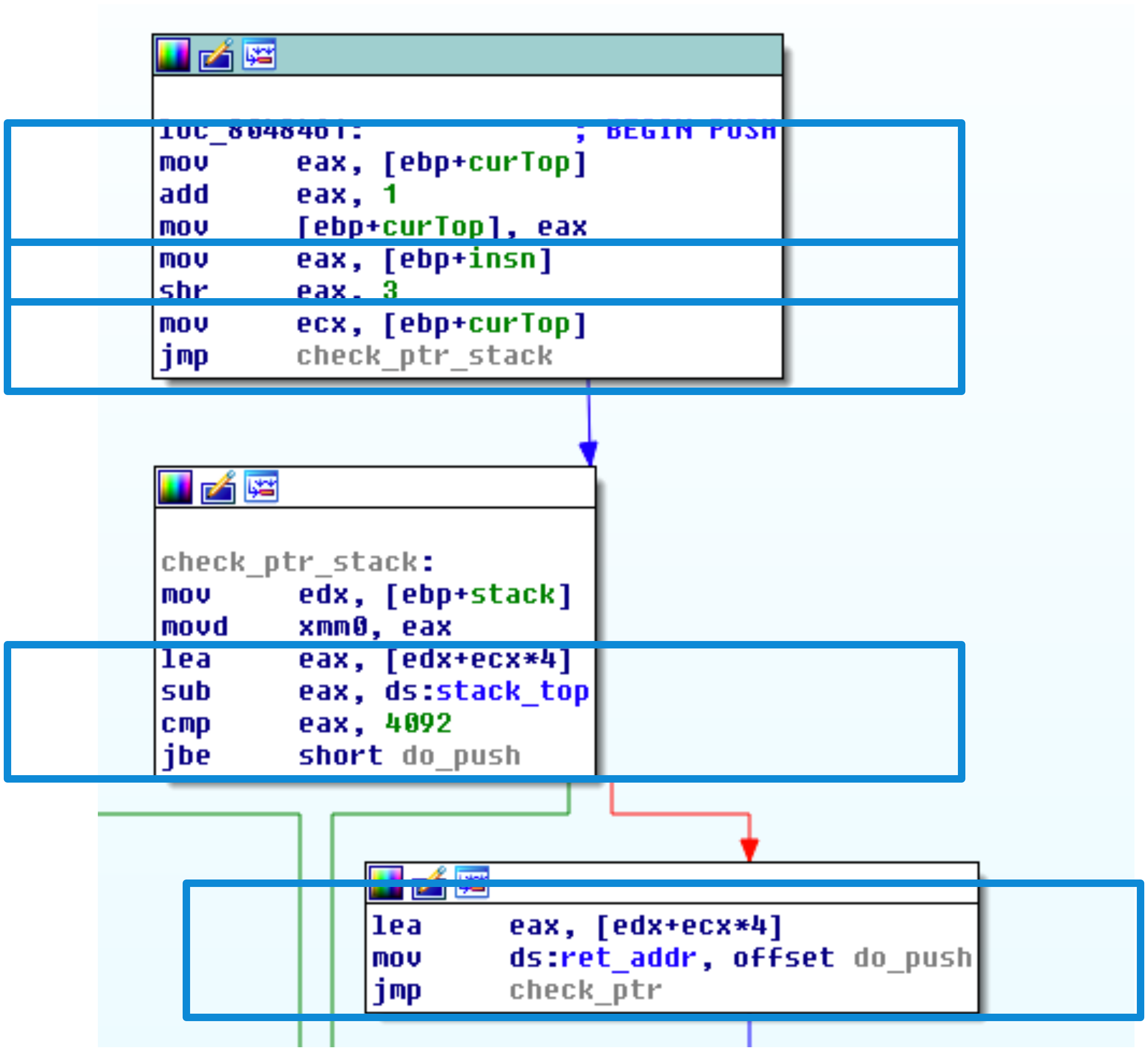
```
0. PUSH 0
1. PUSH 0
2. PUSH 0
3. SUB
4. PUSH 0
5. DUP 0
6. PUSH 0
7. PUSH 0
8. JMPZ
...
RET
```

GOTO Line 0

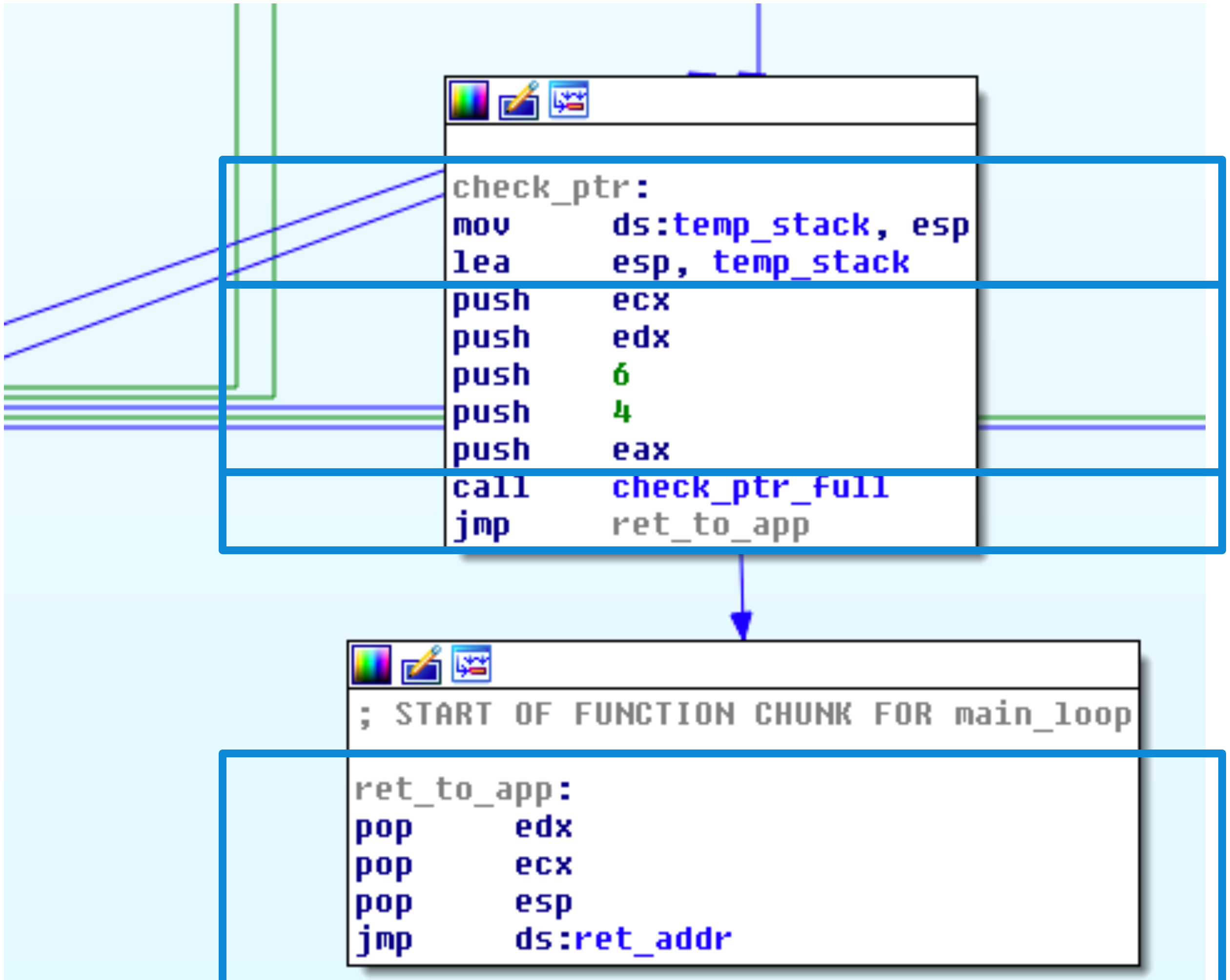
Original YAN01_00012: PUSH Instruction

```
loc_8048461:                    ; case PUSH
mov     eax, [ebp+curTop]
add     eax, 1
mov     [ebp+curTop], eax
mov     eax, [ebp+insn]
shr     eax, 3
mov     ecx, [ebp+curTop]
mov     edx, [ebp+stack]
mov     [edx+ecx*4], eax
jmp     loop_start
```

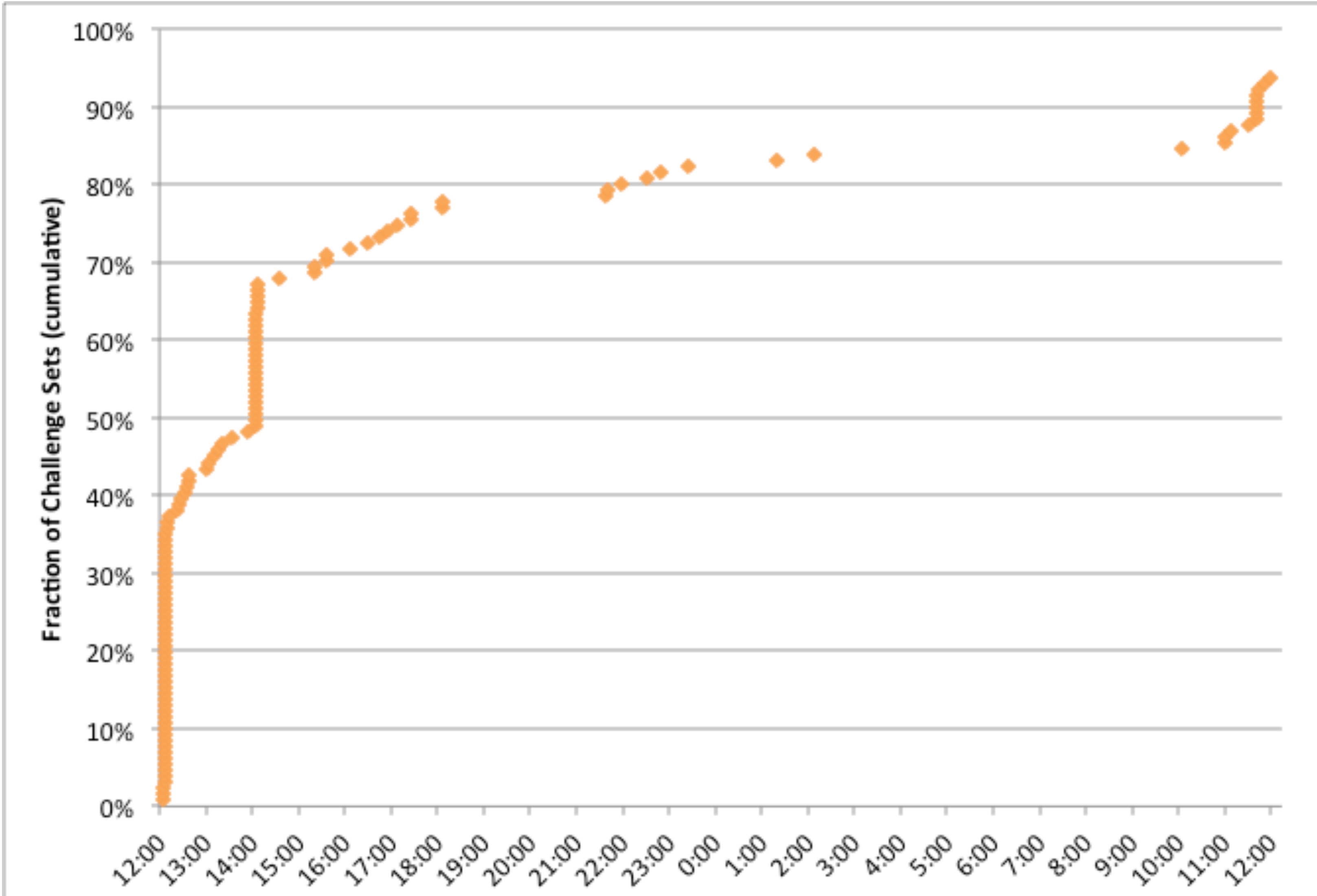
Defended YAN01_00012: PUSH Instruction (1/2)



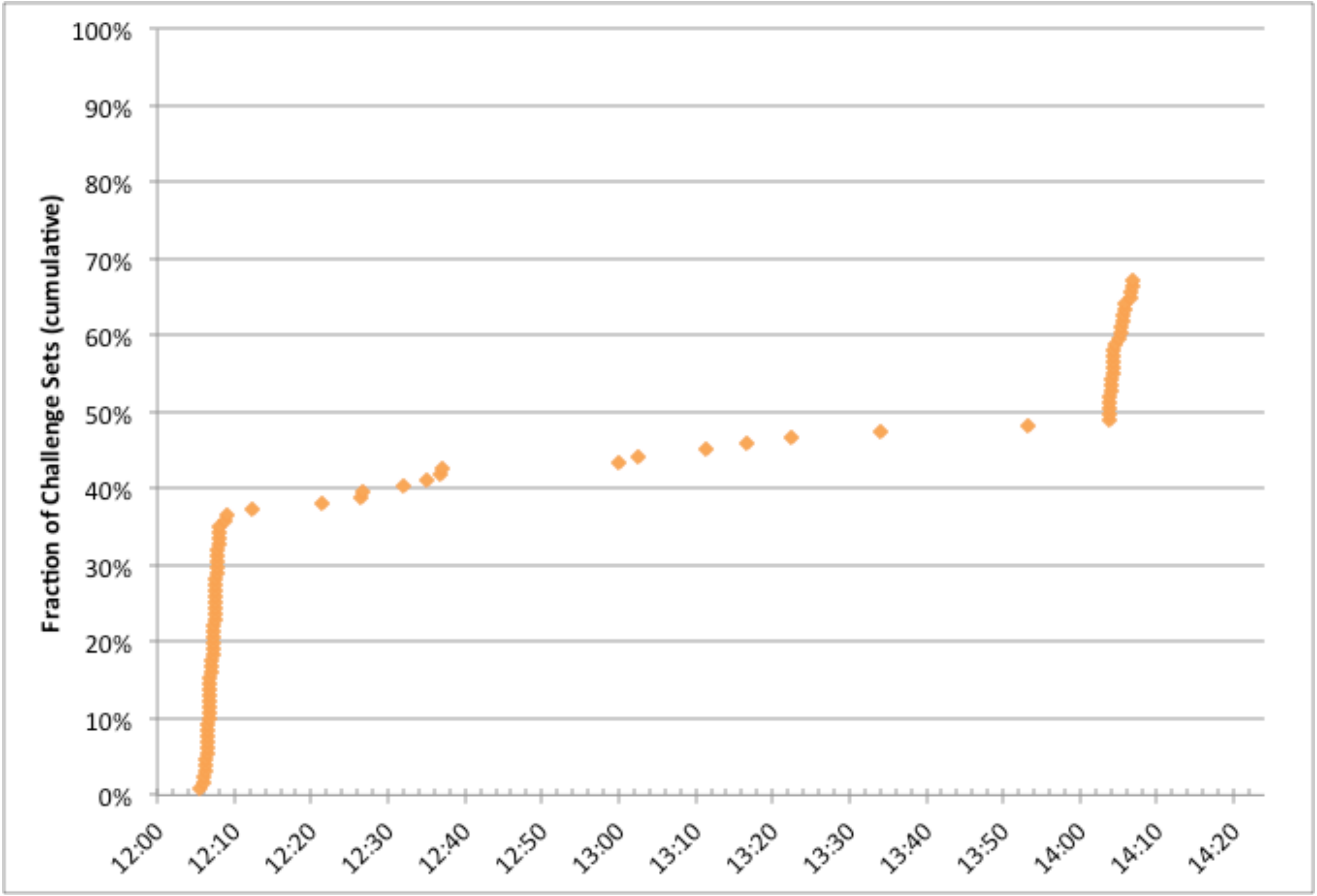
Defended YAN01_00012: PUSH Instruction (2/2)



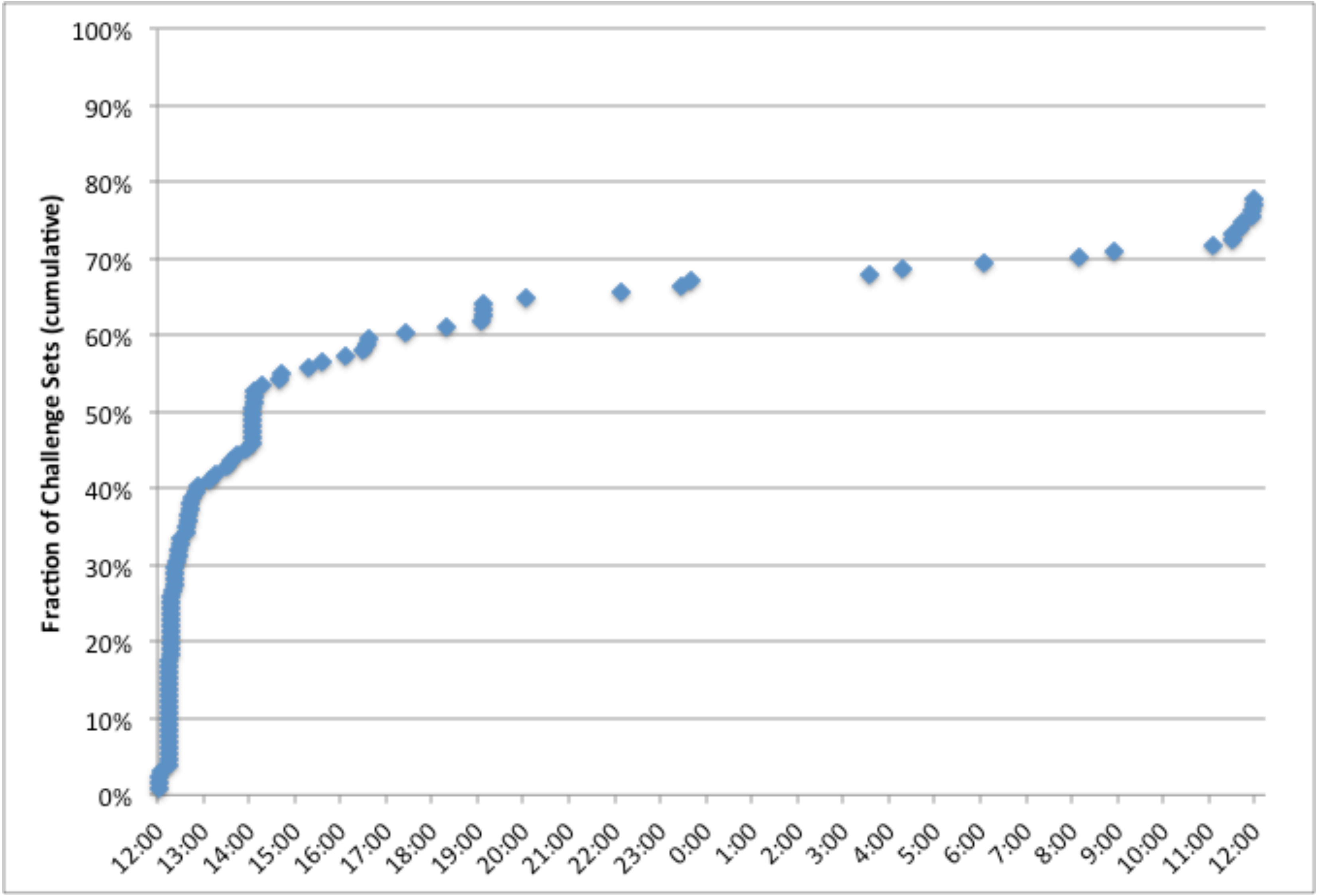
Time to First Defended Solution



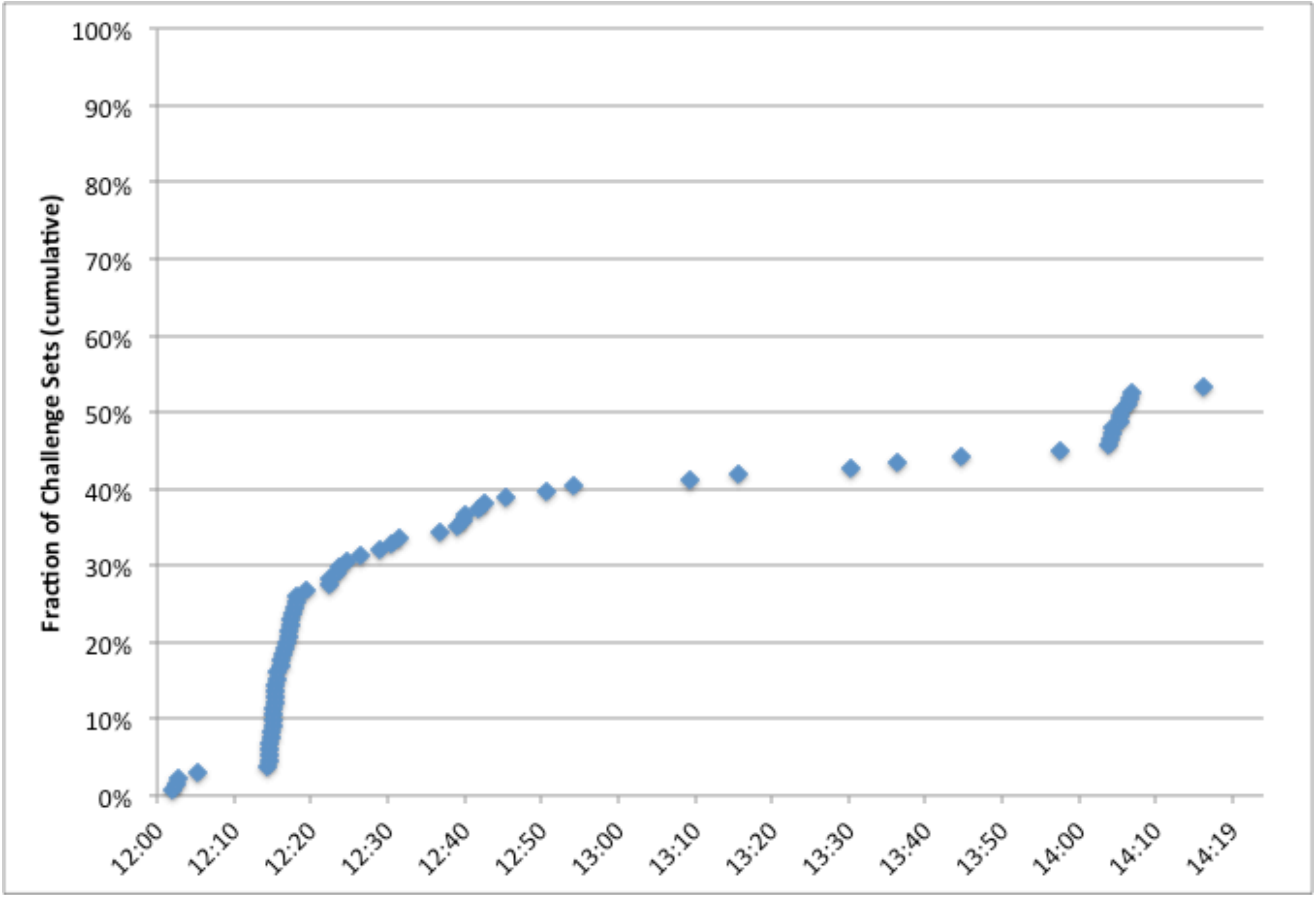
Time to First Defended Binary



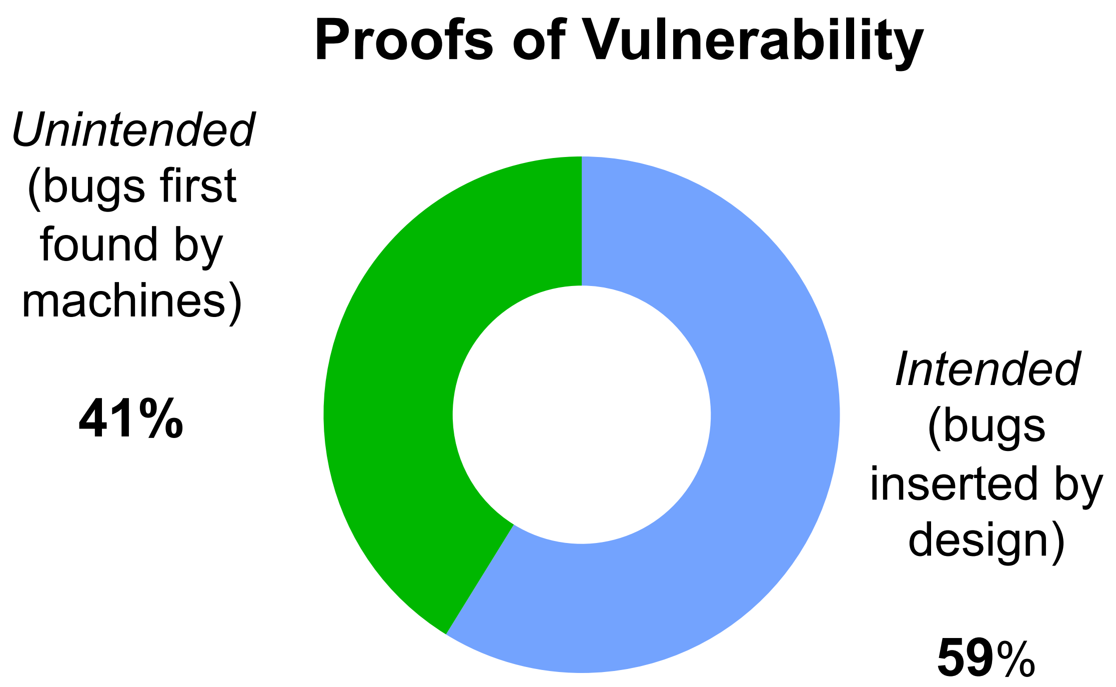
Time to First Successful PoV



Time to First Successful PoV



Machines Think Differently



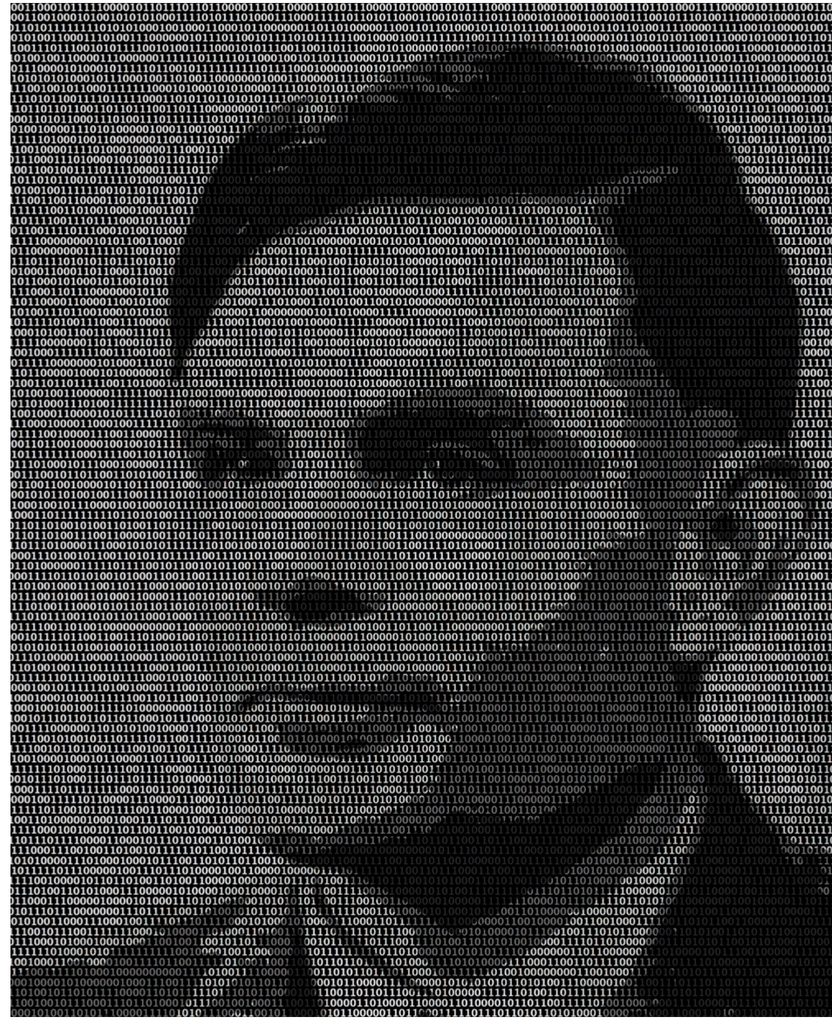
For example – it turns out *not reading* from a socket can cause a buffer overflow if writer doesn't check available buffer space

June 3, 2015: In the Beginning…

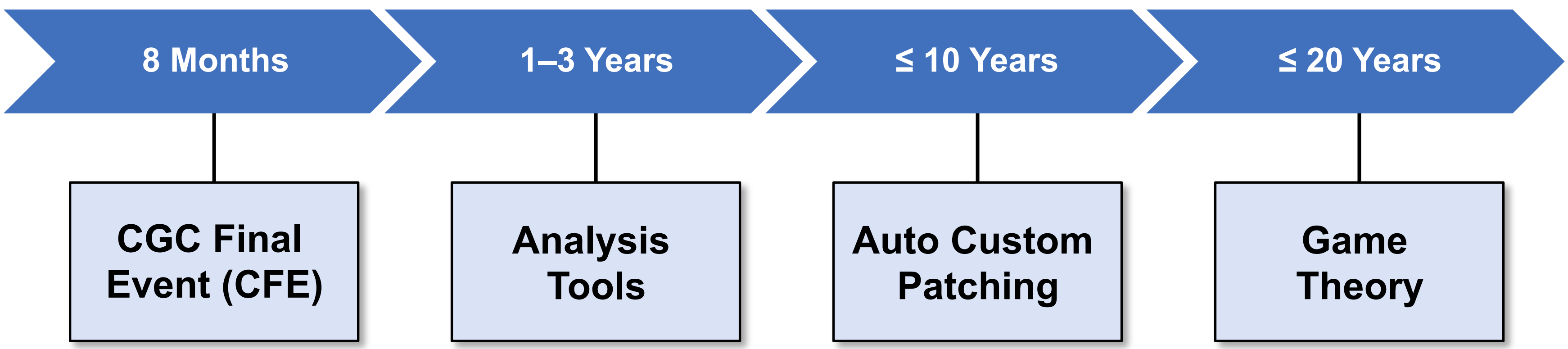
CRS received perfect scores on 18% of challenges

“We can only see a short distance ahead, but we can see plenty that needs to be done.”

~ Alan Turing



Envisioned Road Map



Short Term: Assisting the Software Analyst

- Automated unpacking
- Vulnerability discovery
- Taint tracing
  - Functional equivalency of standard routines
- Anomaly Detection
  - Can compare with specification/expectations and look for divergence (e.g., old vs new variants of program)
- Currently available tools
  - Mcsema
  - angr (management)
  - BAP
  - BitBlaze

Mid-term: Custom Patching

- On-demand custom patching
  - Reduced time to patch
  - Not dependent on vendor
  - Tailored to specific workload/inputs
  - Update unsupported legacy software

- Use CRS to remove/modify functionality
  - Remove remote tracking
  - Don’t load images in email client
  - Don’t turn URLs into links

Side Effect: software diversity prevents widespread attacks

Automated 3rd-Party Repairs Are Close

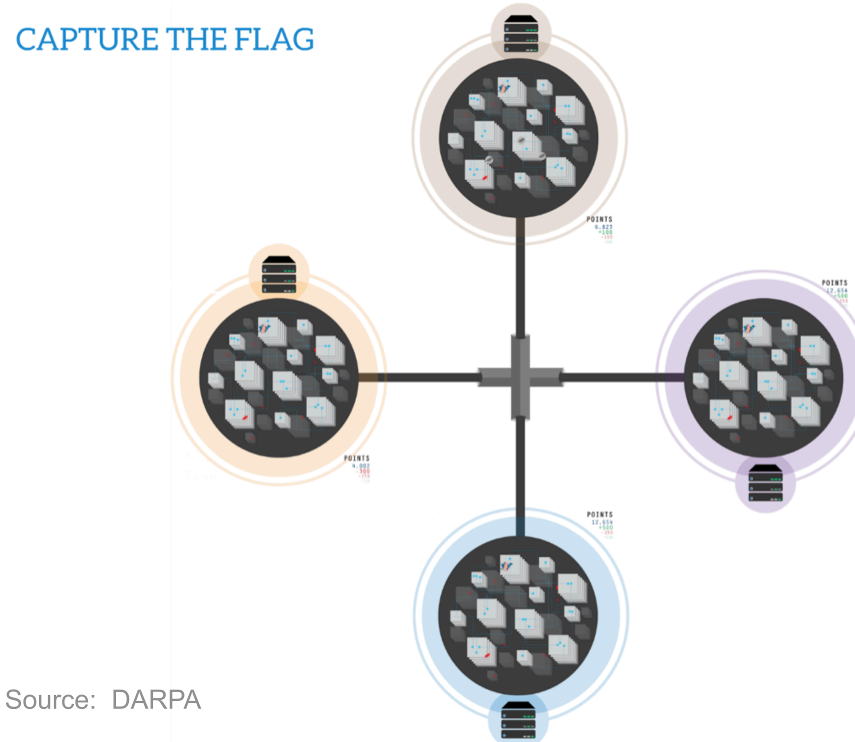
- Fun With Shellshock: http://blog.regehr.org/archives/1187 (Oct. 11 2014)
  - "We simply inserted an exploit that attempted to cat a "passwd" file into a GET request"

```
GET /appstore/index.php HTTP/1.1
User-Agent: () { :;}; /bin/cat /home/mitll/passwd > /tmp/hello.txt
Host: 155.98.38.76:7701
Accept: */*
```

  - A3 able to remove bash functionality and mitigate vulnerability
  - "…A3's mandatory mediation blocked the attack …"
  - "A3 took ~2 minutes to find a repair …"
  - "A3 took an additional ~1.5 minutes to find a source code repair …"

Long term:
CRS Interactive with Opponent

- **Machine vs machine competition adds complexity and a 'Game Theoretic' aspect, where CRS may:**
  - **Make decisions on what type of patch to deploy**
  - **Learn what kind of analysis is being used**
  - **Intentionally misinform opponent**
  - **Set up weaker defenses to see how opponent reacts**



Source: DARPA

- **Could a CRS build an adversary profile based on visible artifacts?**
- **"Interactive honeypot" backed by a CRS?**

Envisioned Road Map

8 Months	1–3 Years	≤ 10 Years	≤ 20 Years
CGC Final Event (CFE)	Analysis Tools	Auto Custom Patching	Game Theory

```
ENTER _start
      call main
      pushl %eax
      call _terminate
END   _start
```

Lesson 1

Relevant solutions require real-world constraints

- **Security solutions cannot break functionality**
- **Significant performance degradation will not be tolerated**
- **Security solutions must mitigate attacks**

Lesson 2

There's no substitute for the real event

- **Integration and scale issues are hidden until you 'go live'**
- **Practice like you play**

Lesson 3

Beware the cracks in the abstraction layer

- **Low-level artifacts can affect determinism of higher-level behavior**
- **Resources are finite**

Lesson 4

Don't trust; verify

"Be conservative in what you do, be liberal in what you accept from others."

[RFC 793]

Lesson 4

Don’t trust; verify

“Be conservative in what you do, be ~~liberal~~ *extremely* conservative in what you accept from others.”
[CGC mantra]

- **Be explicit in specification, validate ruthlessly**
- **Solve the halting problem (watchdog timer)**

Postel’s Robustness Principle Patch. http://langsec.org/postel-principle-patch.txt

Lesson 5

If you can't repeat it, it didn't happen
(if it's not automated, you can't repeat it)

- **Automated unit tests for everything**
- **Building and testing challenge sets**
- **Scoring cluster provisioning and push-button scoring**

Lesson 6

Give people a challenge, and they will surprise you

“Machines take me by surprise with great frequency.”
– Alan Turing

CGC Artifacts

Source Code and Walkthroughs:

https://github.com/CyberGrandChallenge

Packages, VMs, and Scoring Data:

http://repo.cybergrandchallenge.com

Acknowledgements

Meet the Finalists

ForAllSecure

Pittsburgh, PA

Deep Red

Arlington, VA

TECHx

Charlottesville, VA

CYBER
GRAND_CHALLENGE

Shellphish

Santa Barbara, CA

disekt

Athens, GA

Codejitsu

Berkeley, CA

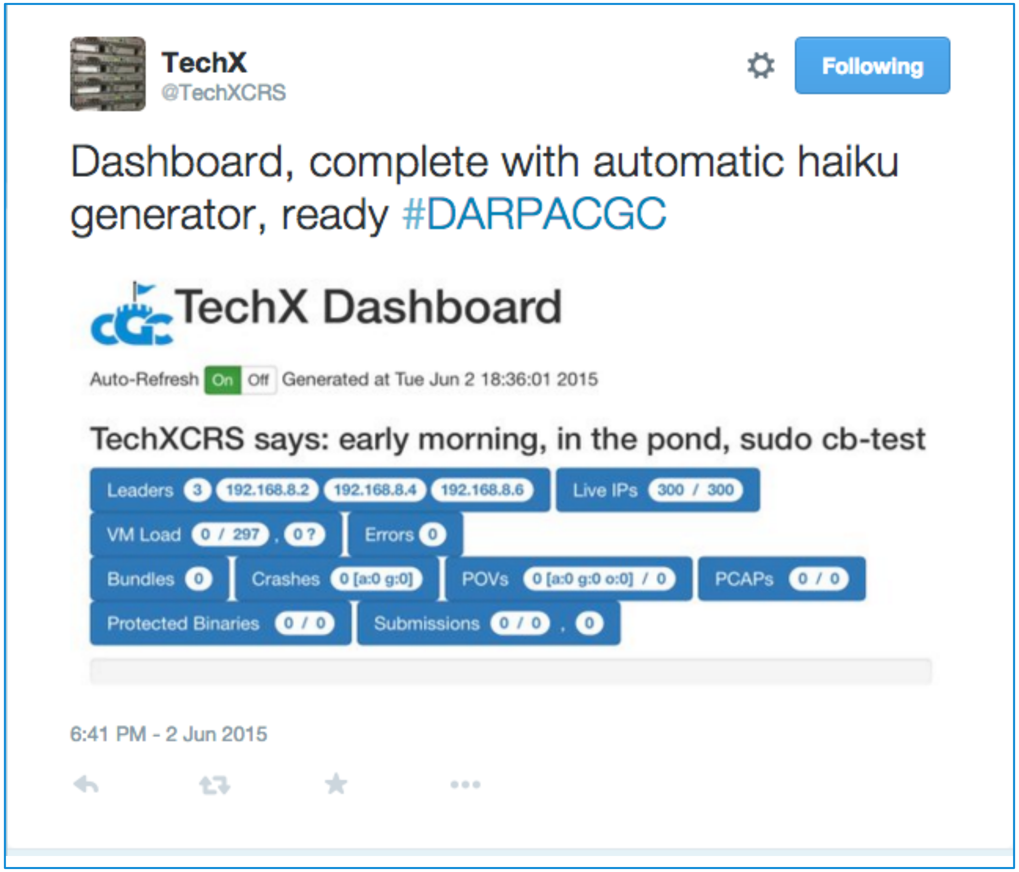
CSDS

Moscow, ID

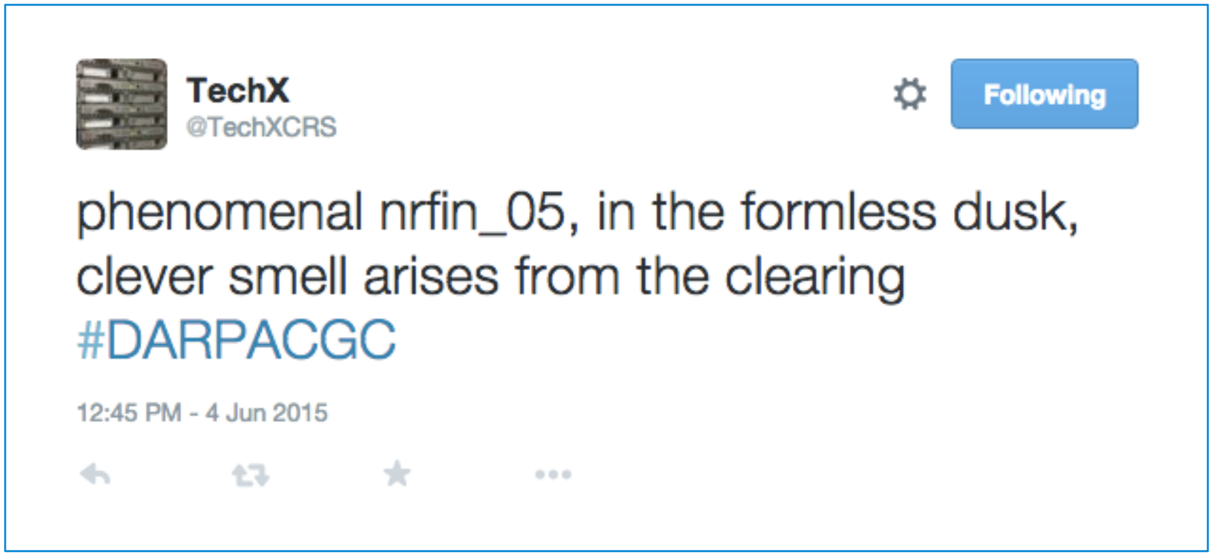
Save The Date: CGC Final Event

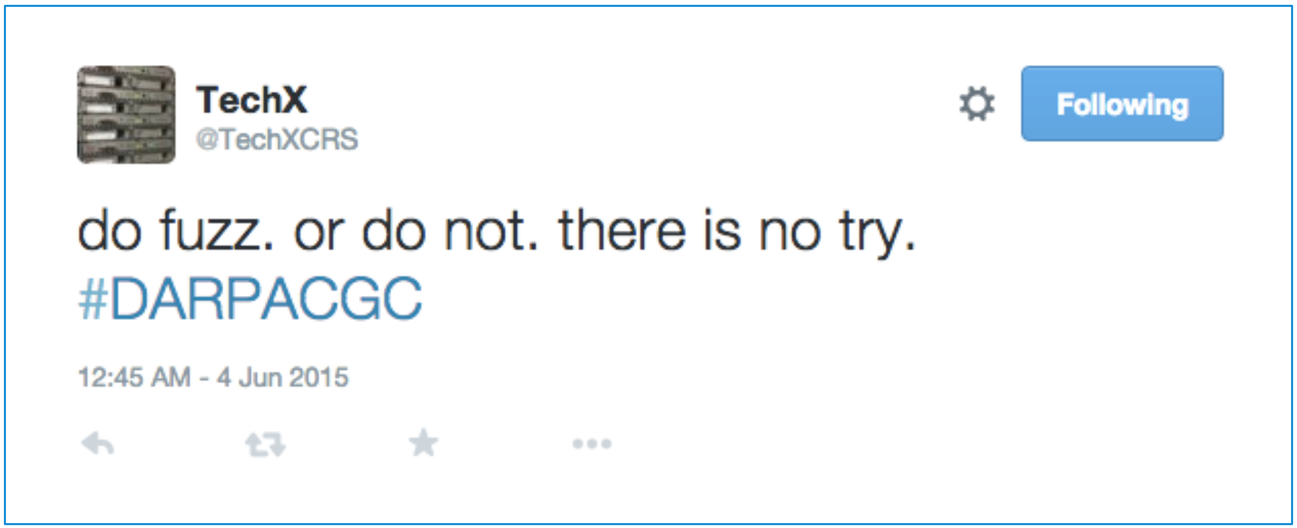
CYBER
GRAND_CHALLENGE

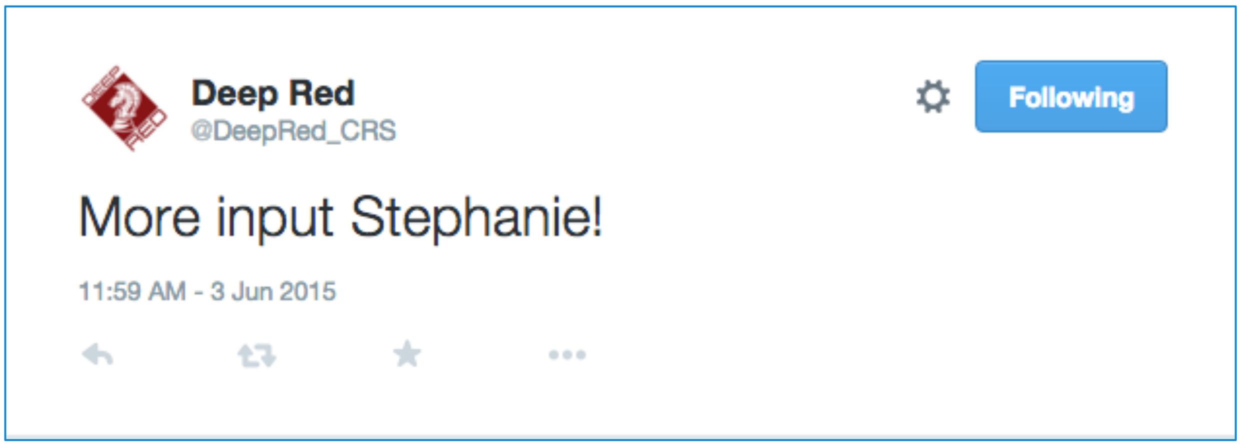
August 4, 2016
DEF CON
Las Vegas, NV

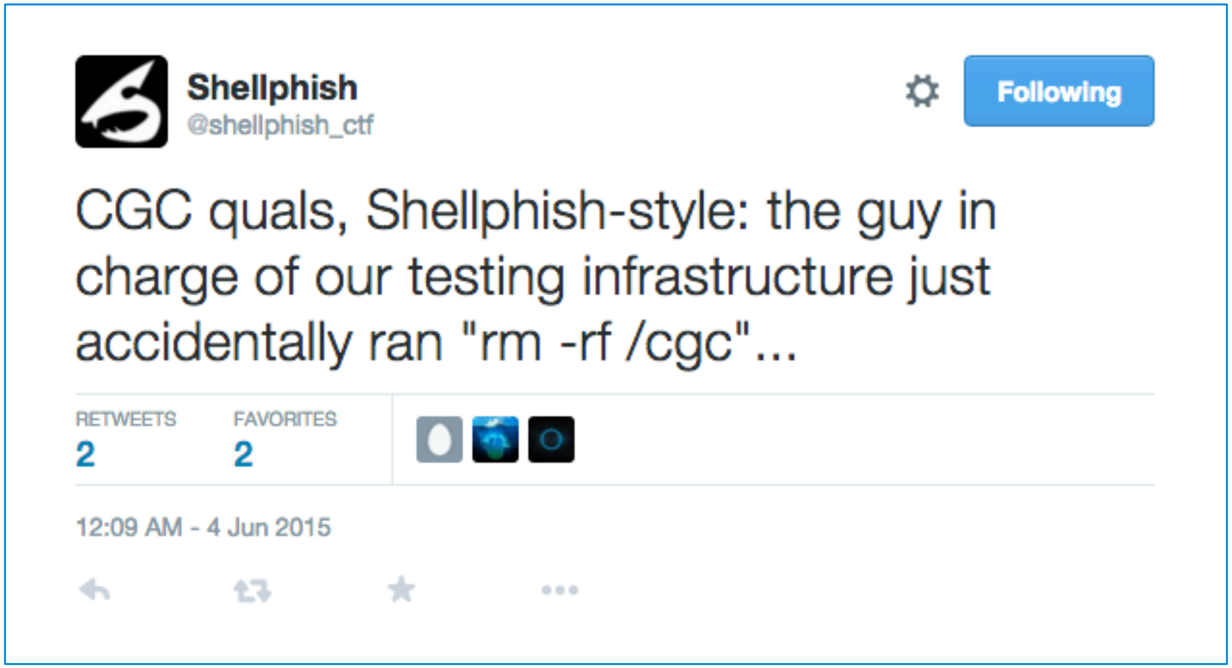


TechX
@TechXCRS
Following
Dashboard, complete with automatic haiku generator, ready #DARPACGC
TechX Dashboard
Auto-Refresh On Off Generated at Tue Jun 2 18:36:01 2015
TechXCRS says: early morning, in the pond, sudo cb-test
Leaders 3 192.168.8.2 192.168.8.4 192.168.8.6
Live IPs 300 / 300
VM Load 0 / 297 , 0 ?
Errors 0
Bundles 0
Crashes 0 [a:0 g:0]
POVs 0 [a:0 g:0 o:0] / 0
PCAPs 0 / 0
Protected Binaries 0 / 0
Submissions 0 / 0 , 0
6:41 PM - 2 Jun 2015


TechX
@TechXCRS
Following
phenomenal nrfin_05, in the formless dusk, clever smell arises from the clearing #DARPACGC
12:45 PM - 4 Jun 2015


TechX
@TechXCRS
Following
do fuzz. or do not. there is no try.
#DARPACGC
12:45 AM - 4 Jun 2015


Deep Red
@DeepRed_CRS
Following
More input Stephanie!
11:59 AM - 3 Jun 2015

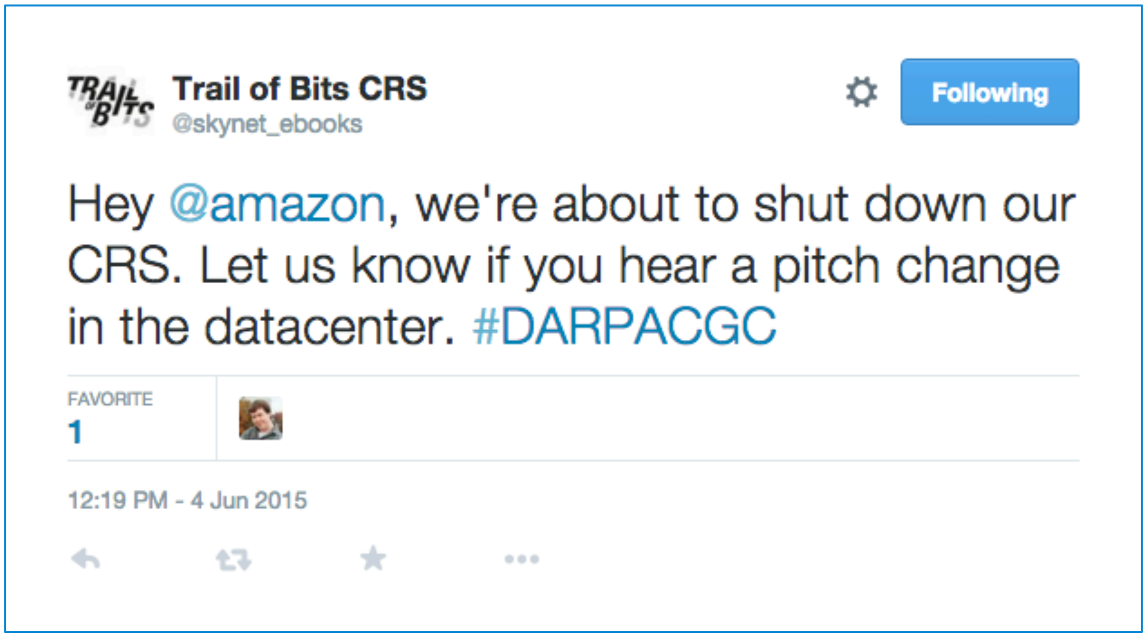

Shellphish
@shellphish_ctf
Following
CGC quals, Shellphish-style: the guy in charge of our testing infrastructure just accidentally ran "rm -rf /cgc"...
RETWEETS
2
FAVORITES
2
12:09 AM - 4 Jun 2015

TechX
@TechXCRS

Following

go ahead POV, make my day #DARPACGC

1:52 PM - 3 Jun 2015


Trail of Bits CRS
@skynet_ebooks
Following
Hey @amazon, we're about to shut down our CRS. Let us know if you hear a pitch change in the datacenter. #DARPACGC
FAVORITE
1
12:19 PM - 4 Jun 2015

Mayhem CRS
@MayhemCRS

Following

#CRSFacts We had 28 nodes in GCE, 10 in AWS, and 2 local servers for a total of 804 cores. We crashed 81 CBs and patched 128 CBs. #DARPACGC

RETWEET
1

3:39 PM - 4 Jun 2015

Save The Date: CGC Final Event

CYBER
GRAND_CHALLENGE

August 4, 2016
DEF CON
Las Vegas, NV

Meet the Finalists: CodeJitsu

CodeJitsu is based at the University of California Berkeley and led by Professor Dawn Song. The CodeJitsu cyber reasoning system is based on automated binary analysis and hardening.

Meet the Finalists: CSDS

The Center for Secure and Dependable Systems at the University of Idaho is proud to sponsor team CSDS. This self-funded team consists of Dr. Jia Song, a postdoc, and Dr. Jim Alves-Foss, director of CSDS. Although a small team, they are building from scratch a new and innovative custom tool suite to participate in CGC.

Meet the Finalists: DeepRed

Deep Red is composed of a small team of specialized engineers from Raytheon Corporation. The Deep Red team is inventing new ways to analyze software that builds on the team's uniquely rich heritage in computer security.

Meet the Finalists: disekt

disekt is a computer security team that participates in various Capture the Flag security competitions hosted by other teams, universities and organizations from around the world.

Meet the Finalists: ForAllSecure

ForAllSecure's technology is the result of more than a decade of program analysis research at Carnegie Mellon University by Professor David Brumley, Thanassis Avgerinos, and Alex Rebert.

Meet the Finalists: Shellphish

Shellphish started at the University of California Santa Barbara as the SecLab hacking team. As members graduated and moved, the team expanded to include other locations such as France, United Kingdom, and other exotic locations. Shellphish has participated in more DEF CON CTF events than any other team.

Meet the Finalists: TECHx

The TECHx team consists of leading software analysis experts from GrammaTech, Inc. and the University of Virginia. The team is led by Dr. David Melski, Professor Jack Davidson, and Professor John Knight. GrammaTech and UVA are co-developers of an automatic software-hardening technology called PEASOUP.