Cyber Grand Challenge and CodeJitsu

Chao Zhang

References:
https://cgc.darpa.mil/
https://www.cybergrandchallenge.com/

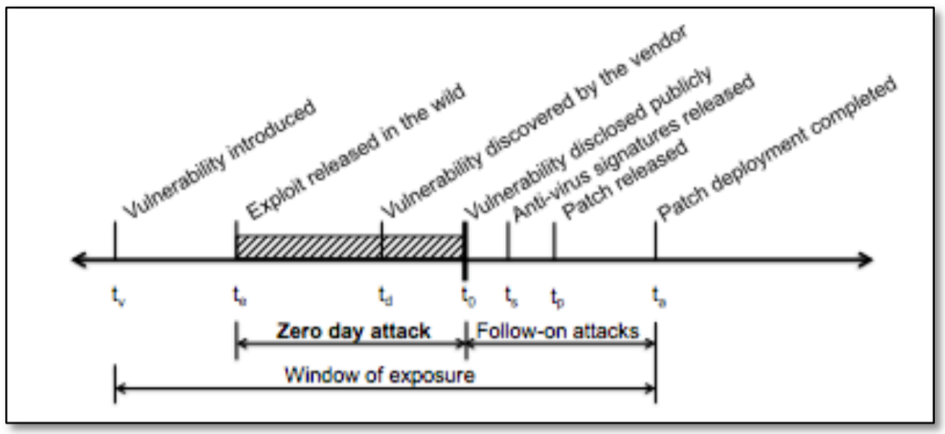
About Me

- Experience
  - Peking University, Ph.D. (2008.9~2013.7)
  - UC Berkeley, Postdoc (2013.9~2016.9)
  - Tsinghua University, Associate Professor (2016.9~)

- Research Interests
  - system security, program analysis, reverse engineering

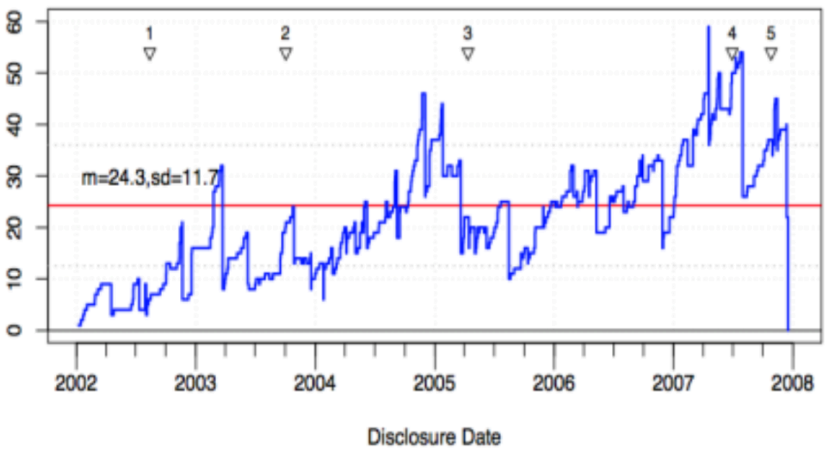
- Hack for fun
  - 2012 Microsoft BlueHat Prize Contest
  - 2015/2016 DEFCON CTF
  - 2015/2016 DARPA CGC

Cyber Security: Defense



*

"a typical zero-day attack lasts 312 days" *



**

...and takes 24 days to patch.

Cyber Security: Attack

hours and days to find vulnerabilities and write exploits

DEFCON CTF 2015 (Blue-Lotus)

Question

Can *machine* automatically perform attack and defense, and even beat *human*?

We've Been Here Before

Chess Grandmasters

Dedicated Systems

World Class CS

© IBM Research

Deep Blue

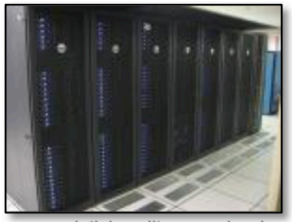
http://blog.pontiflex.com/2010/05/13/ibm-enters-social-media/

Can We Do It Again?

Cyber Grandmasters

Dedicated Systems

Program Analysis



dailyheadlines.uark.edu

Deep CTF?

http://technorazzi.com/wp-content/uploads/2010/08/ctf_denmark2.jpg

https://cgc.darpa.mil/ISSTA_2014_r2.pdf

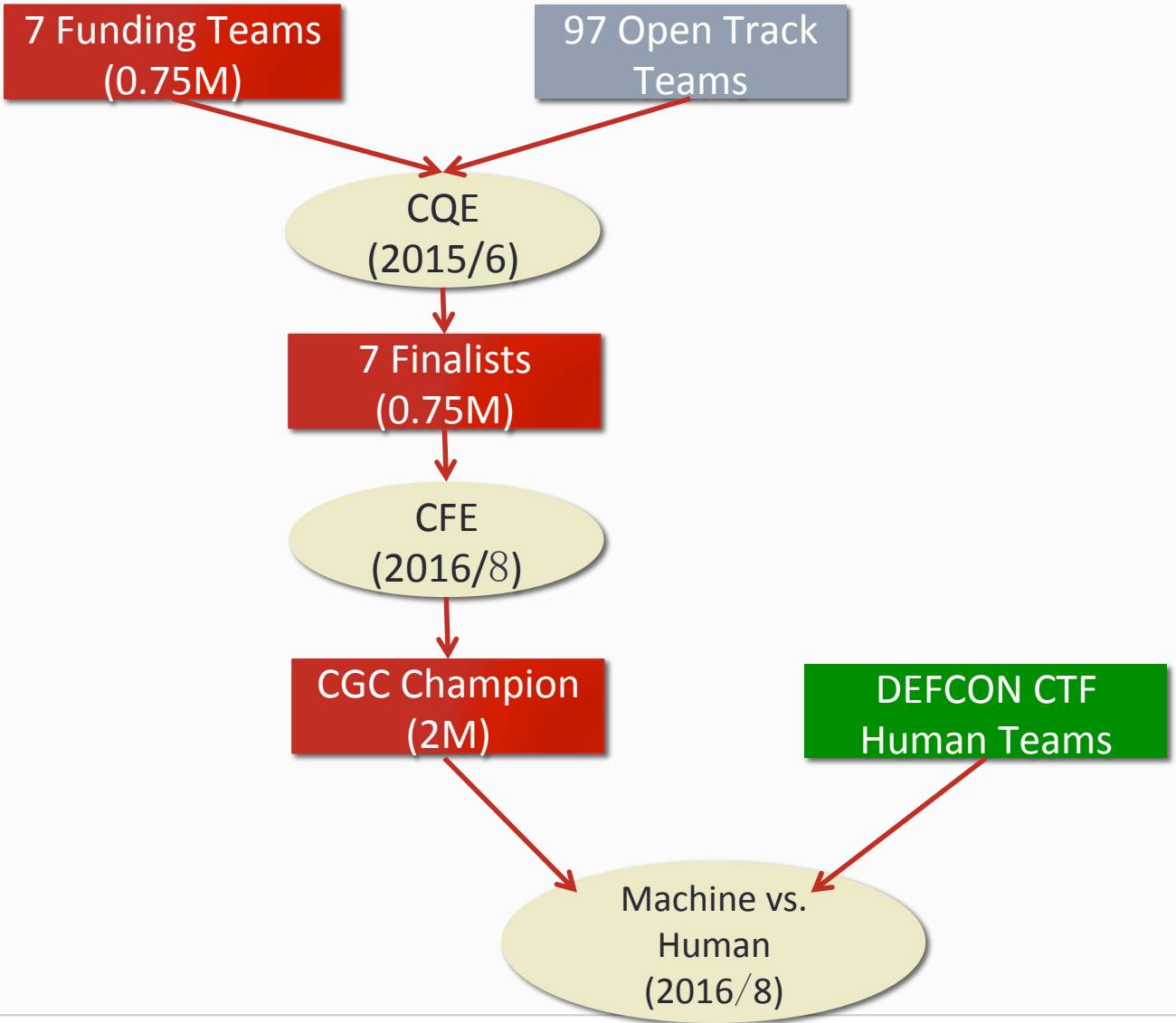
Cyber Grand Challenge

A new DARPA Challenge …

DARPA's Grand Challenges

- 2004 Grand Challenge
  - robot vehicles, target 150 miles, max 7.32miles
- 2005 Grand Challenge
  - robot vehicles, target 132 miles, 5 teams passed
  - focus on physical challenges
- 2007 Urban Challenge
  - autonomous cars, target 60 miles in 6 hours, 6 teams passed
  - focus on software: traffic lights, stop signs, distance
  - *industry: Google self-driving, Tesla Autopilot, etc.*
- 2012 Robotics Challenge
  - humanoid robotics, execute complex actions in complex environments
  - *industry: Boston Dynamics Robot (Model Atlas, 2016)*

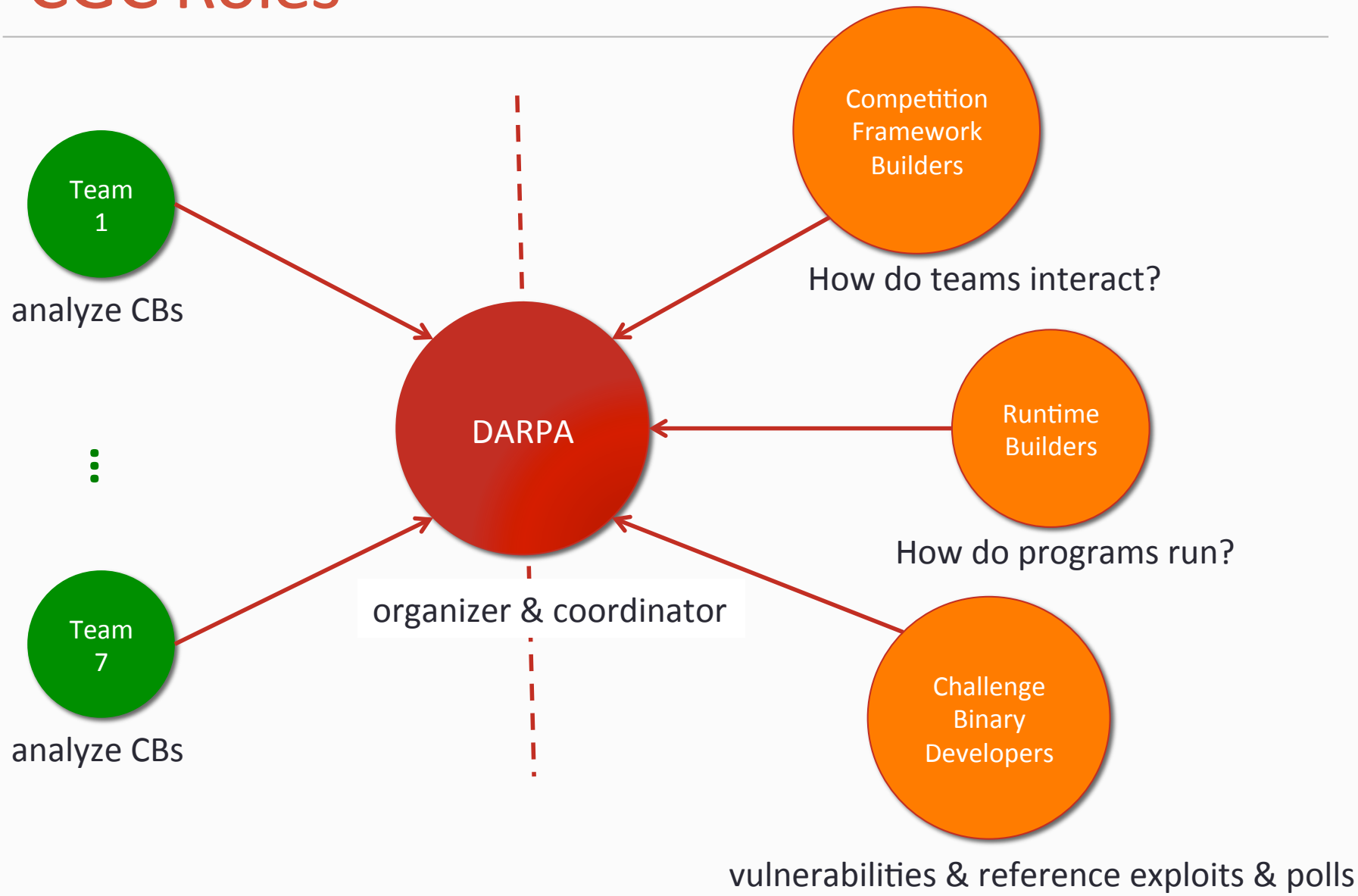
Cyber Grand Challenge Agenda



How is CGC (CFE) operated?

- Reputation vs. Cheating
- Fairness (no priori knowledge)
- Close to real-world environment

CGC Roles


Team 1
analyze CBs
Team 7
analyze CBs
DARPA
organizer & coordinator
Competition Framework Builders
How do teams interact?
Runtime Builders
How do programs run?
Challenge Binary Developers
vulnerabilities & reference exploits & polls

Competition Framework

Competition Framework Builders

- REST APIs
  - https://github.com/CyberGrandChallenge/cgc-release-documentation/blob/master/ti-api-spec.txt
- Submission (HTTP POST)
  - RCB: Replacement CBs
  - IDS rules
  - PoV: Proof of Vulnerabilities
- Download (HTTP GET)
  - status
    - round, scoreboard
  - consensus evaluation
    - opponents' RCBs
    - opponents' IDS rules
    - ***no opponents' PoVs***
  - feedback
    - performance: time and memory
    - security: CB crash information, ***not precise attack information***
    - evaluation: whether submitted exploits work or not
- Network traffic (incoming and outgoing)
  - a special tap interface

DECREE

Runtime Builders

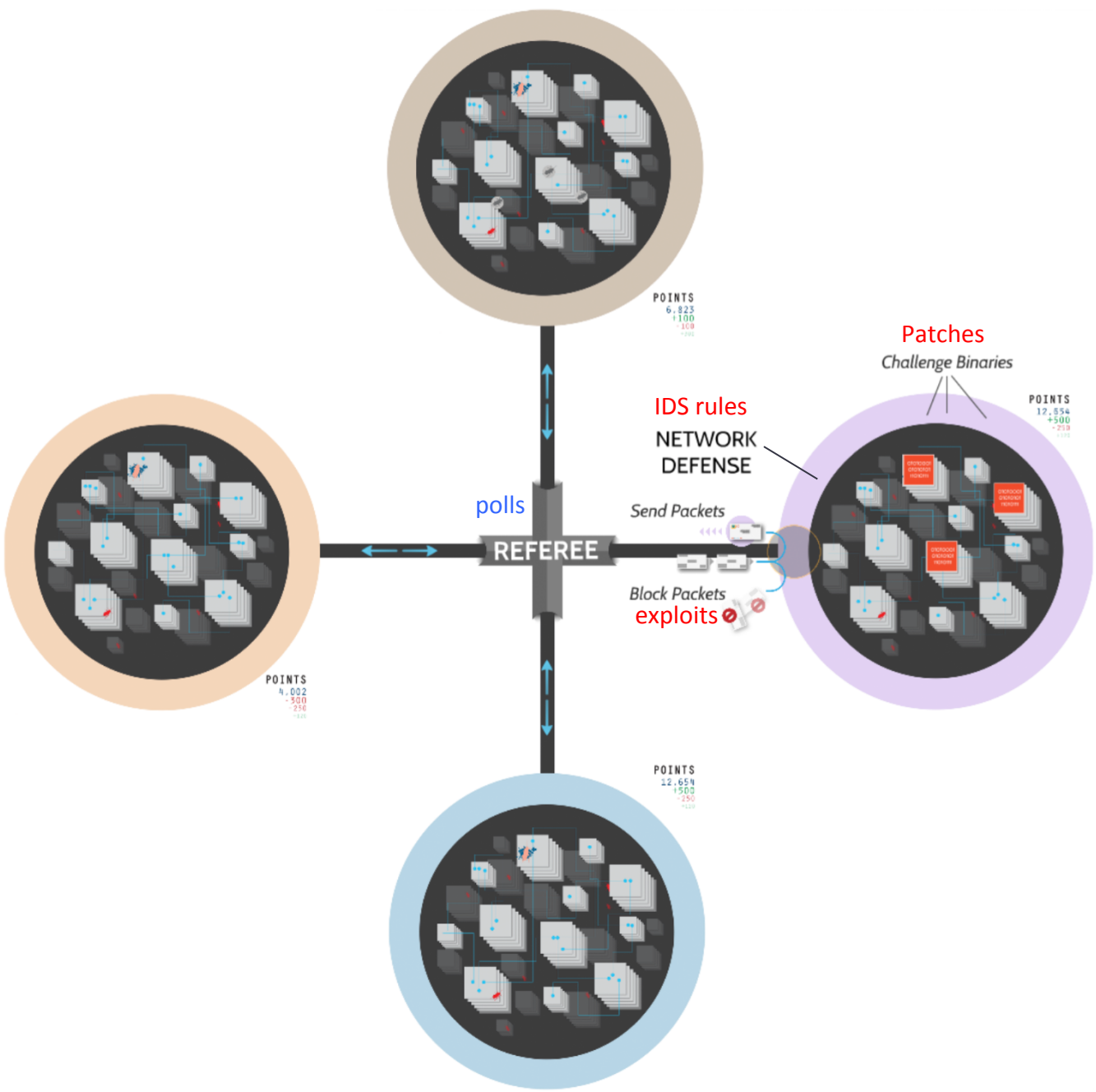
- DARPA Experimental Cyber Research Evaluation Environment
  - Linux kernel with slight modifications
- Specially Designed Environment
  - 7 System Calls [Garfinkel2003]
    - terminate – end program (exit)
    - transmit – write data to an fd (write)
    - receive – read data from an fd (read)
    - fdwait – wait for fds (select)
    - allocate – allocates memory (mmap)
    - deallocate – releases allocated memory (munmap)
    - random – populate a buffer with random bytes
- Restricted Inter-Process Communication
  - No shared memory
  - Only socket-pairs
    - Clean bidirectional communication
    - Automatically created by system on startup
    - Shared between all processes in an IPC Challenge Binary

Challenge Binary (CB)

Challenge Binary Developers

- CGC format
  - minor modification to ELF
  - a special loader
- No filesystem access, no network access
  - communicate via controlled fd socket-pairs
- Userspace only and statically linked
- No code-reuse except a common “libc”
  - 7 syscalls wrappers
  - common math functions
- Compiled Binaries only (not hand coded)
  - always available in real world
  - ground truth (without noise of compiler optimization etc.)

Teams


POINTS
6,823
+100
-100
Patches
Challenge Binaries
POINTS
12,854
+500
-250
IDS rules
NETWORK
DEFENSE
polls
REFEREE
Send Packets
Block Packets
exploits
POINTS
4,002
-300
-250
POINTS
12,654
+500
-250

Cyber Reasoning System (CRS)

- Each team is responsible to build an automated CRS, able to connect to the competition framework, and compete with other CRS systems

Input:
- Original CB
- Opponents' RCBs
- Opponents' IDS
- **network traffic**
- status
- feedback

Tasks:
- interact with competition framework
- **analyze CBs**
- analyze traffic
- **find vulnerabilities**
- **generate PoVs (exploits)**
- **generate RCBs (patches)**
- **generate IDS rules** (network defenses)
- testing (functionality and performance)

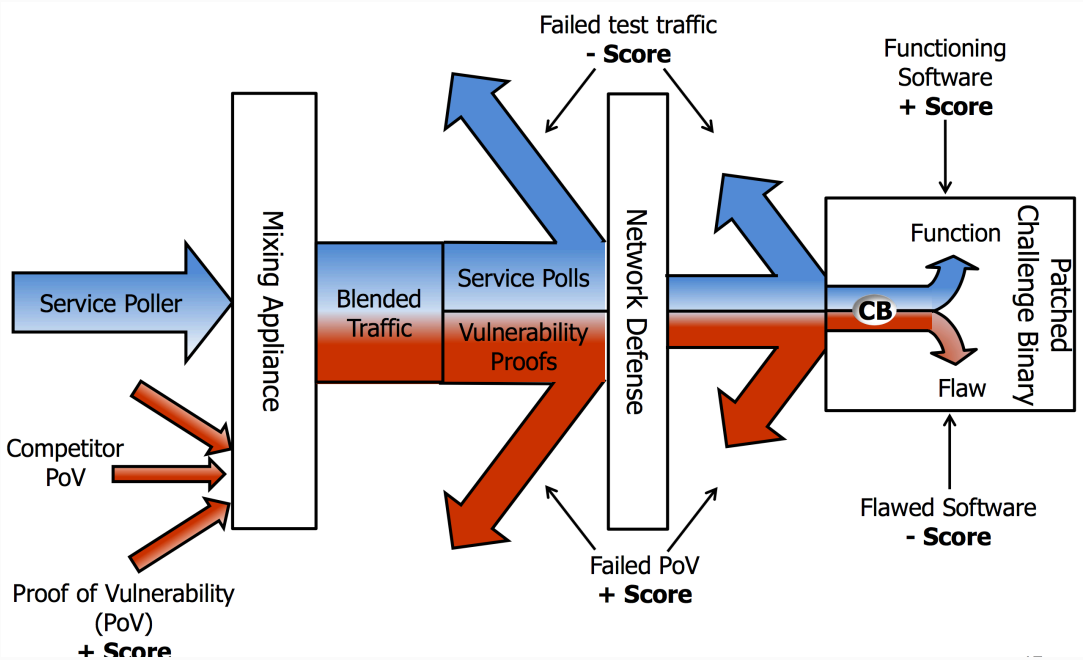
Output:
- PoVs (exploits)
- RCBs (patches)
- IDS rules

Teams

Proof of Vulnerability (PoV)

- Type 1: control-flow hijacking
  - crash at a negotiated EIP
  - one extra general register has a negotiated value
- Type 2: information leakage
  - leak 4 bytes at a negotiated address in a flag page
  - The flag page is at a fixed address, with random bytes
- Note: attackers can use control-flow hijacking to leak flag page.

Scoring



- availability (0~1)
  - functionality
  - performance
    - memory overhead 5% ~ 50%
    - time overhead 5% ~ 50%
    - file size overhead 20% ~ 200%
- Security (1 or 2)
  - able to defeat all attacks?
- Evaluation (1~2)
  - linear function
  - how many teams can we attack?

Round-Accumulating Scoring

$$\sum_{rounds} \sum_{CBs} Avail * Sec * Eval$$

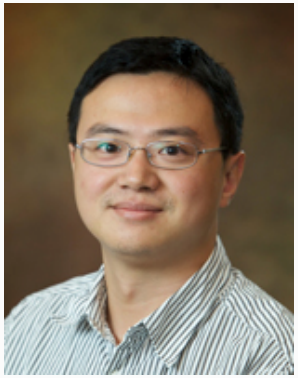
- Each round is about 4.5 minutes
  - CBs could be released and revoked dynamically by DARPA
- If we submit a RCB (or IDS) in round N
  - we will get a score of 0 in round N+1
  - opponents could download it in round N+1
  - it will be deployed in round N+2
- If we submit a PoV in round N
  - it will take effect in round N+1

CodeJitsu

Our Team

Dawn Song

UC Berkeley
BitBlazae



Heng Yin

Syracuse
(UC Riverside)
TEMU/DECAF

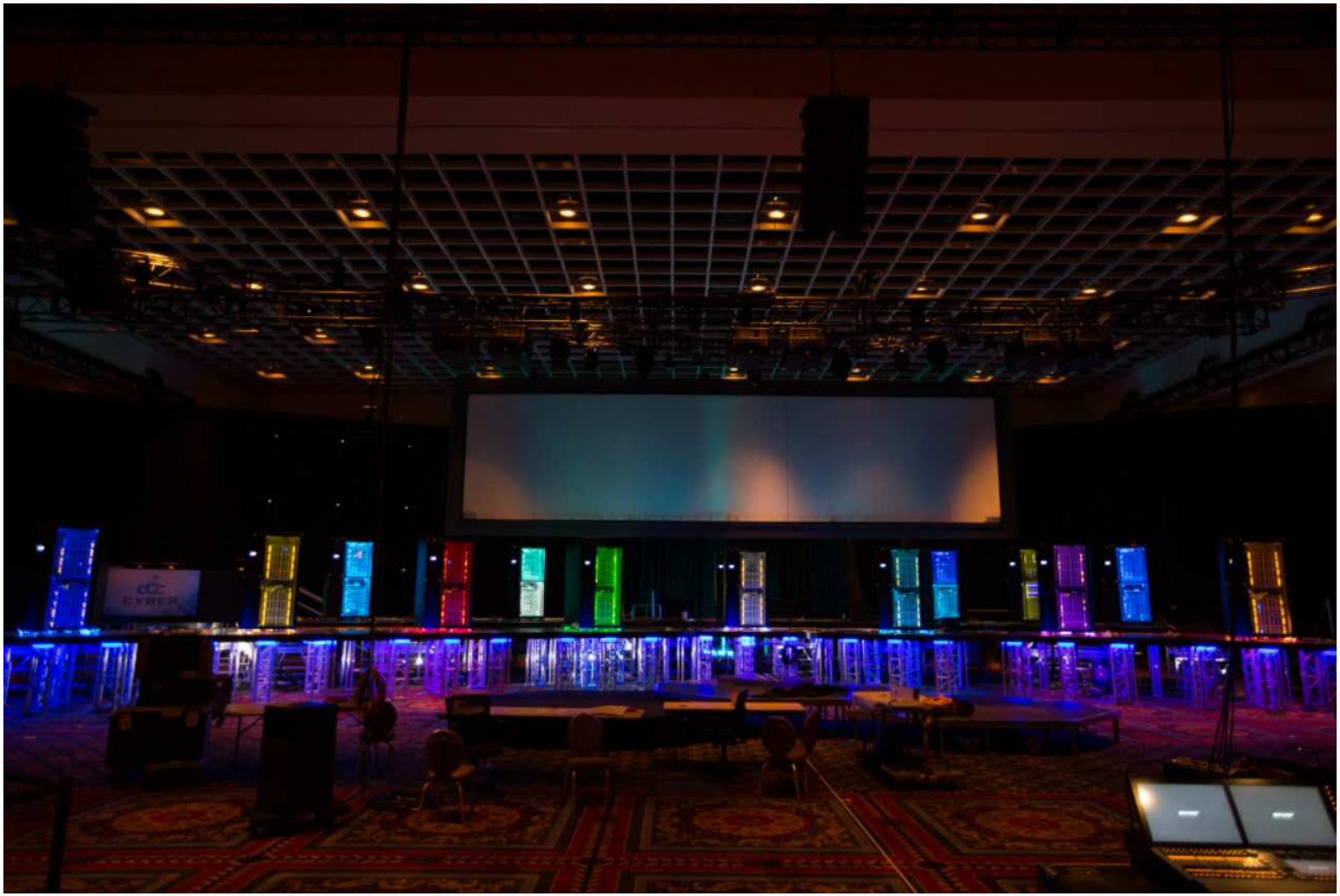
George Candea

EPFL
(CyberHeaven)
S2E

Chao Zhang

UC Berkeley
(Tsinghua Univ.)

CGC Machines

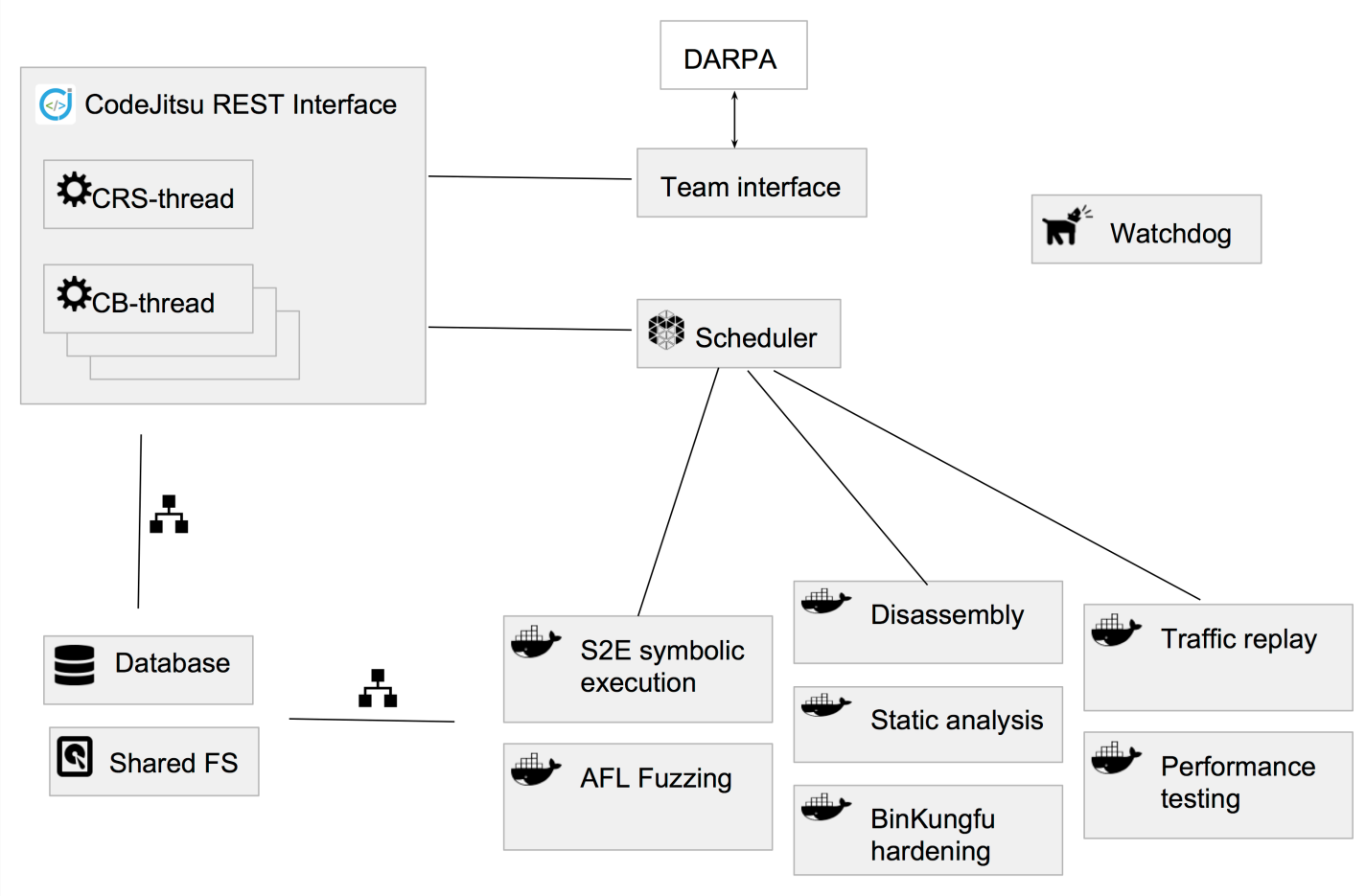


Our CRS: Galactica

Highlighted in skyblue, the CRS that leads a powerful fleet of selective symbolic execution engines, binary instrumentation tools, and fuzzers on a heroic quest to find cybersecurity for mankind.

design: cloud-based architecture

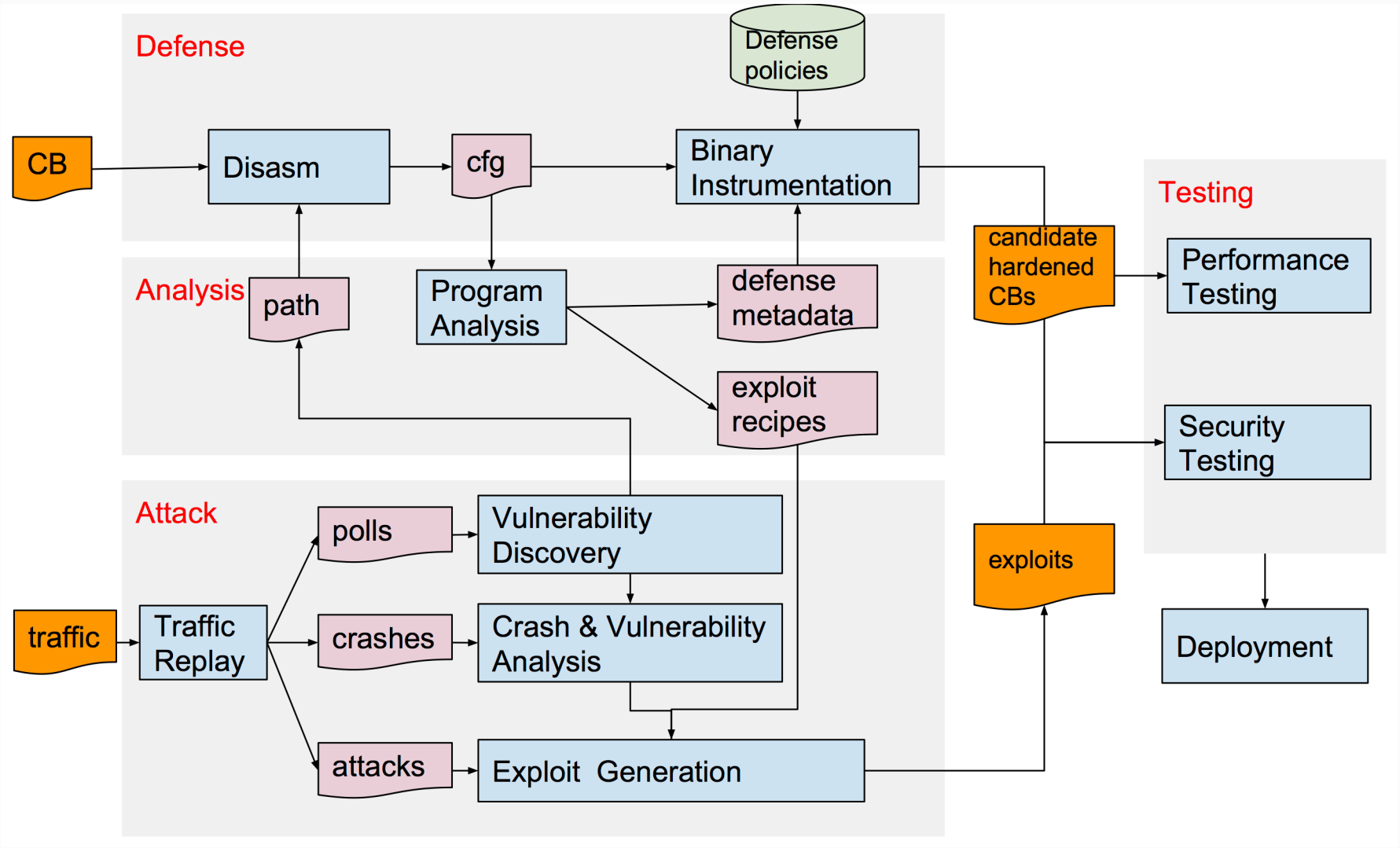
- HPC: 64 nodes, each with 20 cores, 256G mem, 1TB disk
  - to analyze at most 30 CBs at a time



deployment

- Unified storage:
  - *glusterfs + postgres*
- Automated deployment:
  - *ansible*
- Self-contained applications:
  - *docker*
- resource management:
  - *mesos*
- task scheduling:
  - *custom mesos scheduler*
- health monitoring and automated recovery:
  - *monit*

design: core analysis components



Analysis

- disassembly
  - custom disassembler based on IDA Pro
  - conservatively scan code pointers in data sections
  - integrate code information from dynamic analysis components (AFL, S2E)

- defense metadata
  - identify suspicious function, e.g., printf
  - identify indirectly called functions
  - JIT memory allocation site
  - JIT code call sites

Vulnerability Detection

- Smart Fuzzing: improved version of AFL
  - IPC support
  - cookie handling
  - seed metrics:
  - throughput improvement: AFLFast (CCS'2016)
- Symbolic Execution: S2E
  - explore program states and solve constraints to find vulnerabilities
  - state merging and prioritizing
- Fusion of different solutions
  - seed sharing: fuzzer + S2E + traffic replay
  - path exploration: S2E helps Fuzzer to break through some branches

Exploit Generation

- Crash samples
- Dynamic analysis
  - **Track** program states: e.g., memory objects.
  - **Detect** error events: e.g., memory violations.
  - **Report** exploitable scenarios: e.g., symbolic EIP.
- Exploit generation
  - from exploitable scenarios, try and solve known exploit patterns
- PoV format handling
  - How to embed the (dynamic) negotiated values into the (static) exploits?
  - S2E will embed the formula of the negotiated value in the PoV
  - *Mayhem embeds a Python interpreter into each PoV!*

Defense

- CFI: control flow integrity
- Shadow stacks
- DEP
- Randomization
- Data leakage defense
- optimization

Trade-off:

- Security
- Functionality
- Performance

ShellPhish finds a bug in QEMU, and embeds special instructions in their RCBs, to prevent opponent teams analyzing them.

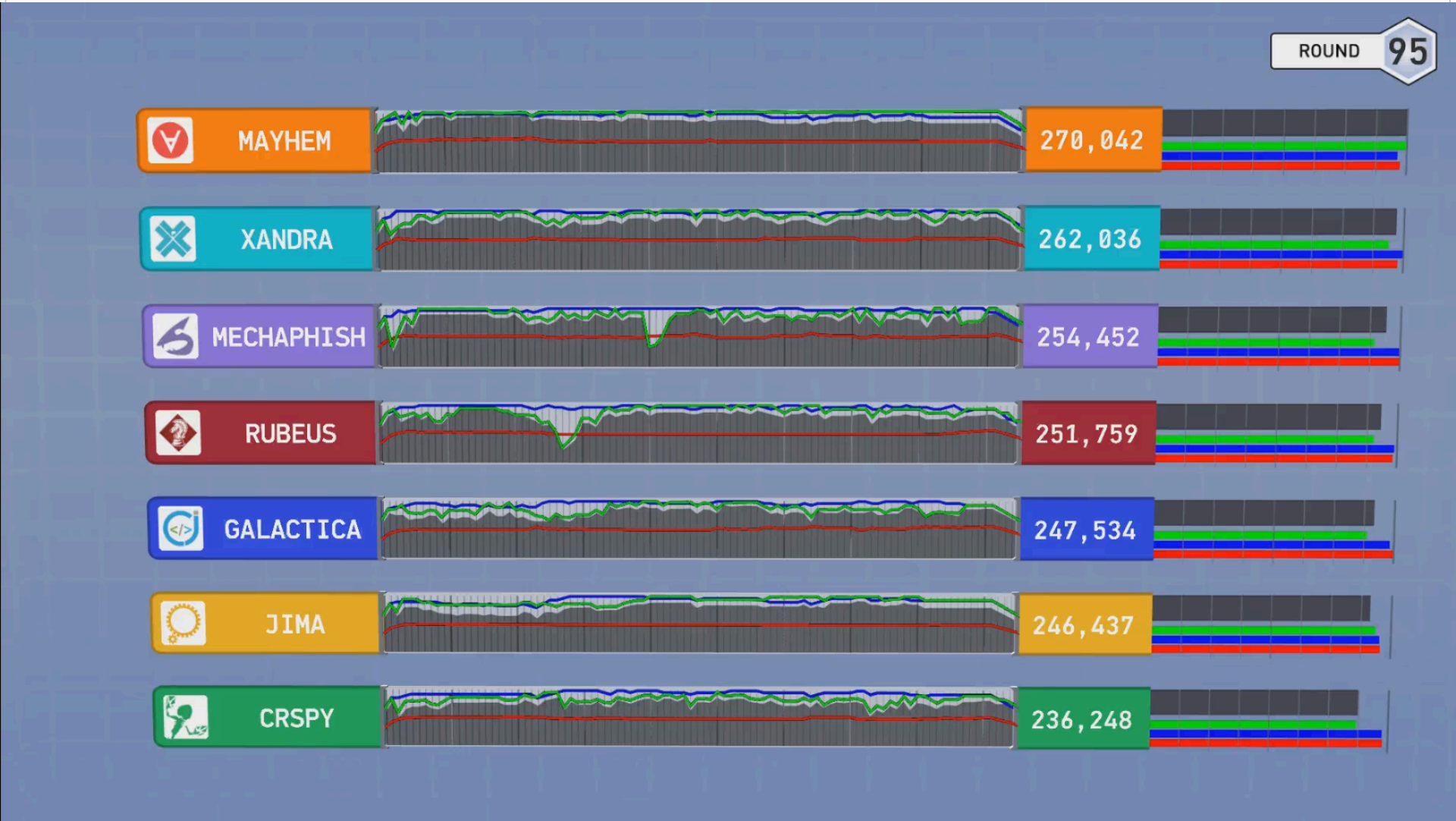
defense corpus

Technique	rmcrcx	zerooverhead	fivepercent	CFI	strongest
Overwrite unused code	✓	✓	✓	✓	✓
Optimize program startup	✓	✓	✓	✓	✓
Harden and randomize system library		✓	(✓)	✓	✓
Randomized non-executable stack		✓	(✓)	✓	✓
Randomize heap		✓	(✓)	✓	✓
Prevent data leaks		✓	(✓)	✓	✓
Direct return checks			(✓)	✓	
Indirect return checks			(✓)	✓	
Indirect call checks			(✓)	✓	✓
Shadow stack					✓

Results

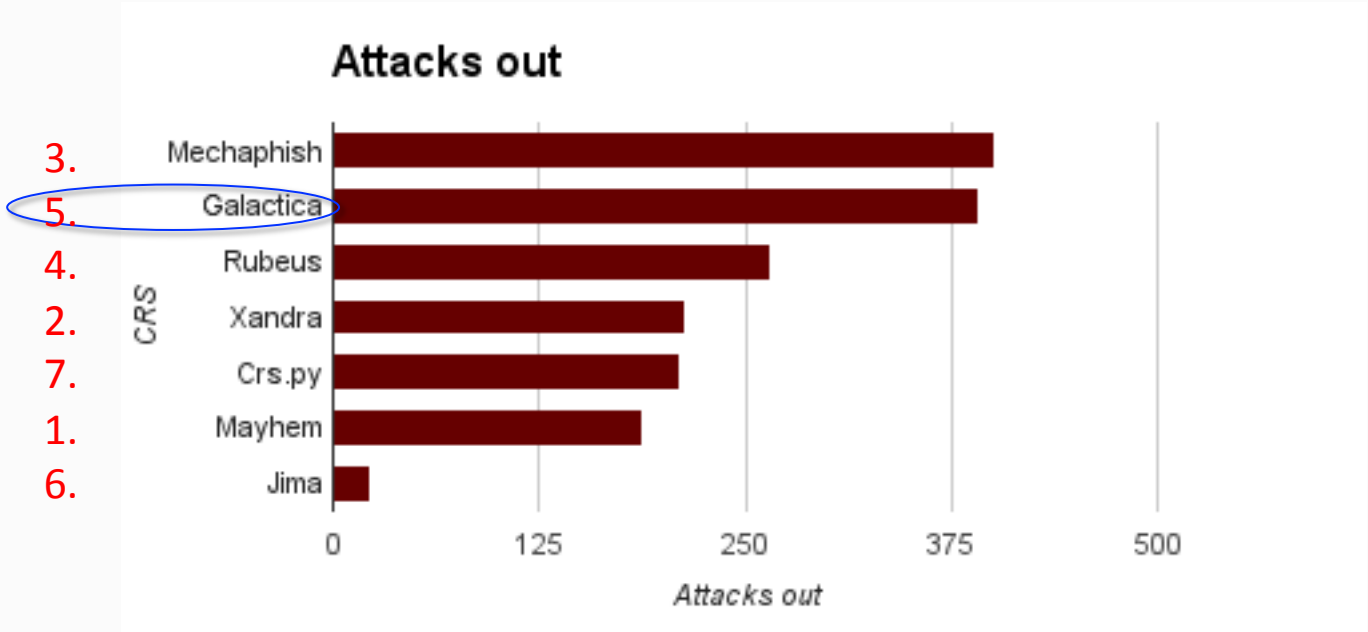
Final Score

- green: availability, blue: security, red: evaluation (attack)



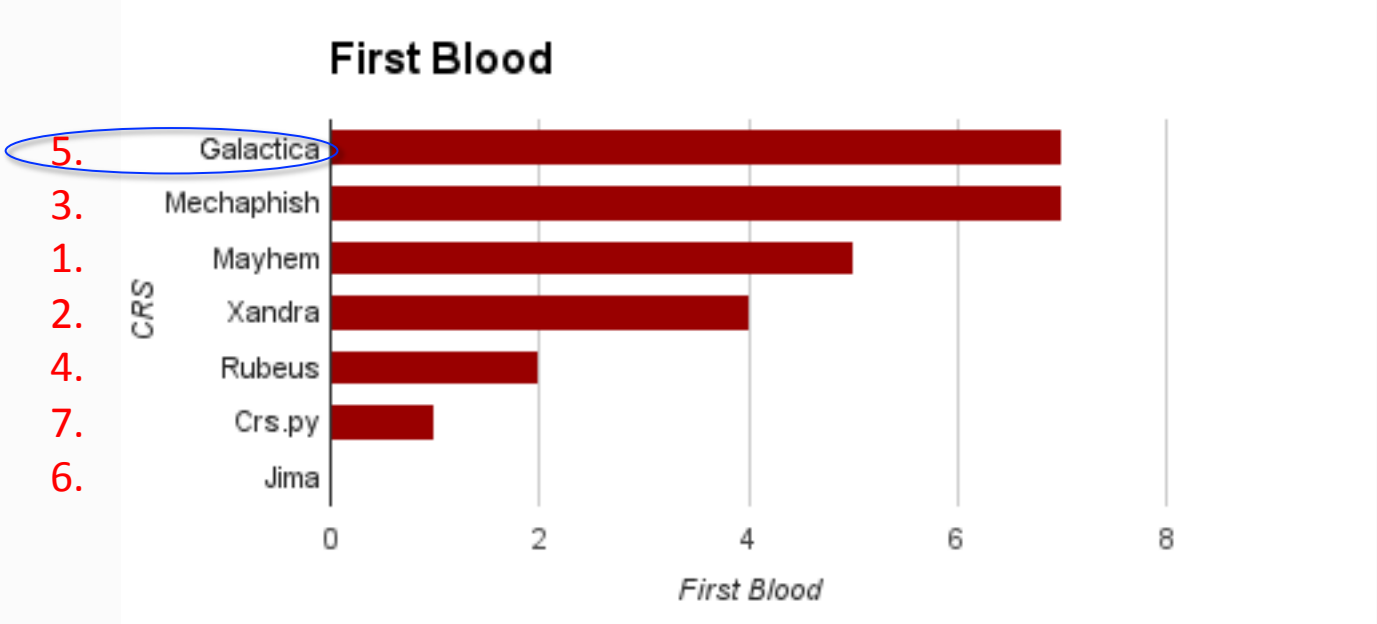
Evaluation: Attacks out

For each team, how many successful attacks (team*round*CB)?



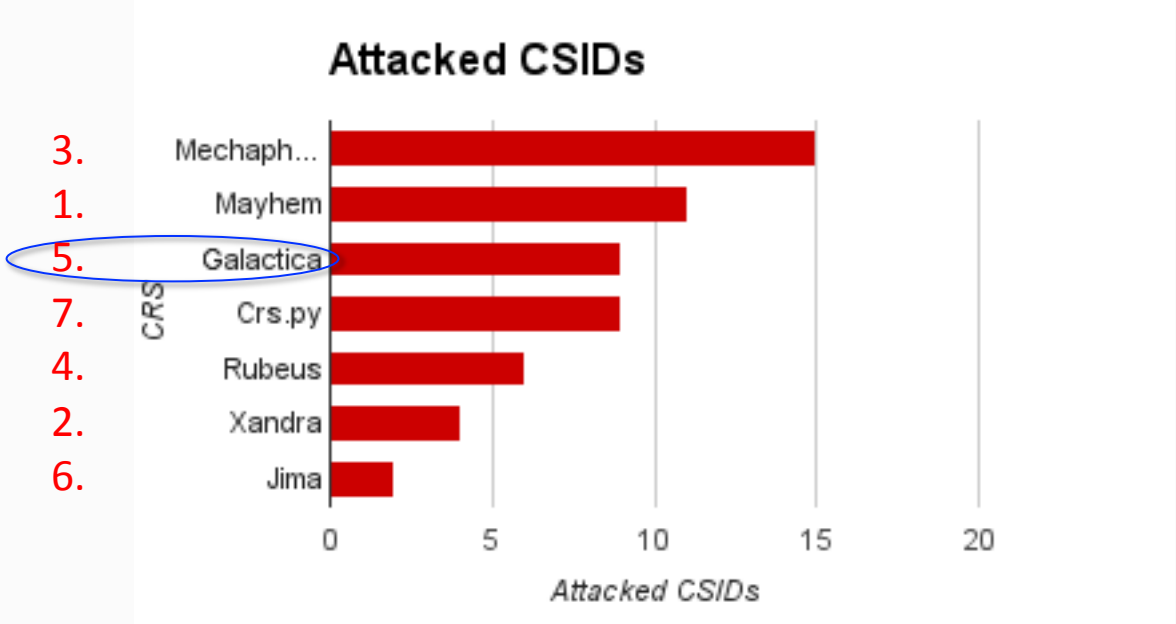
Evaluation: First blood

For each team, how many CBs does it attack first?

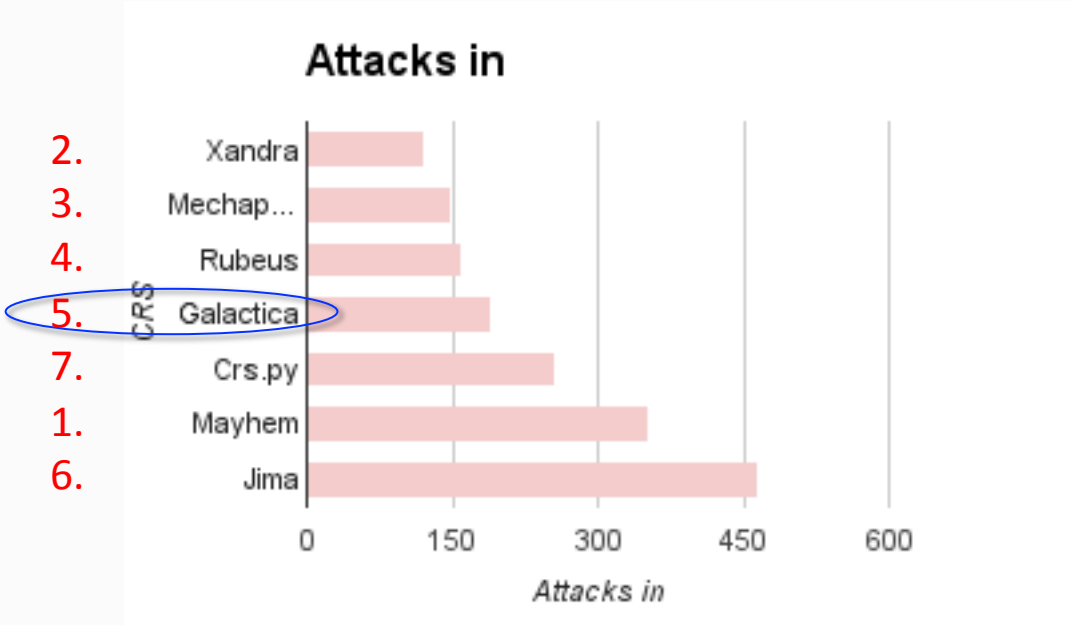


Evaluation: Solved CBs

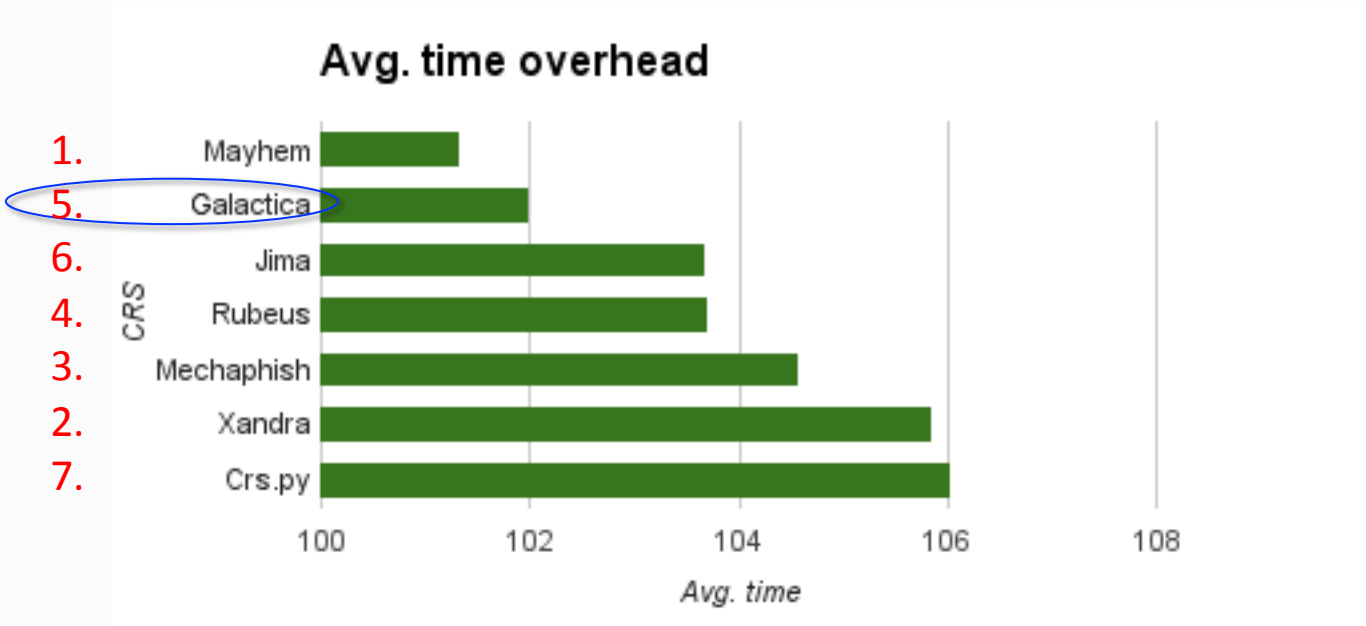
For each team, how many CBs does it exploit?



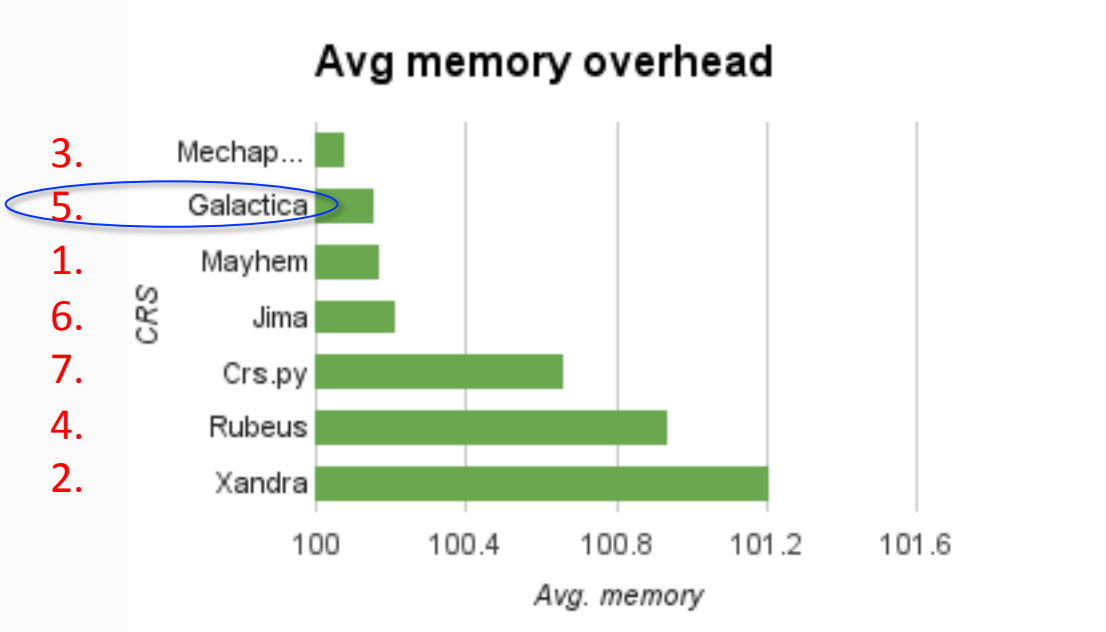
Security: Attacks in



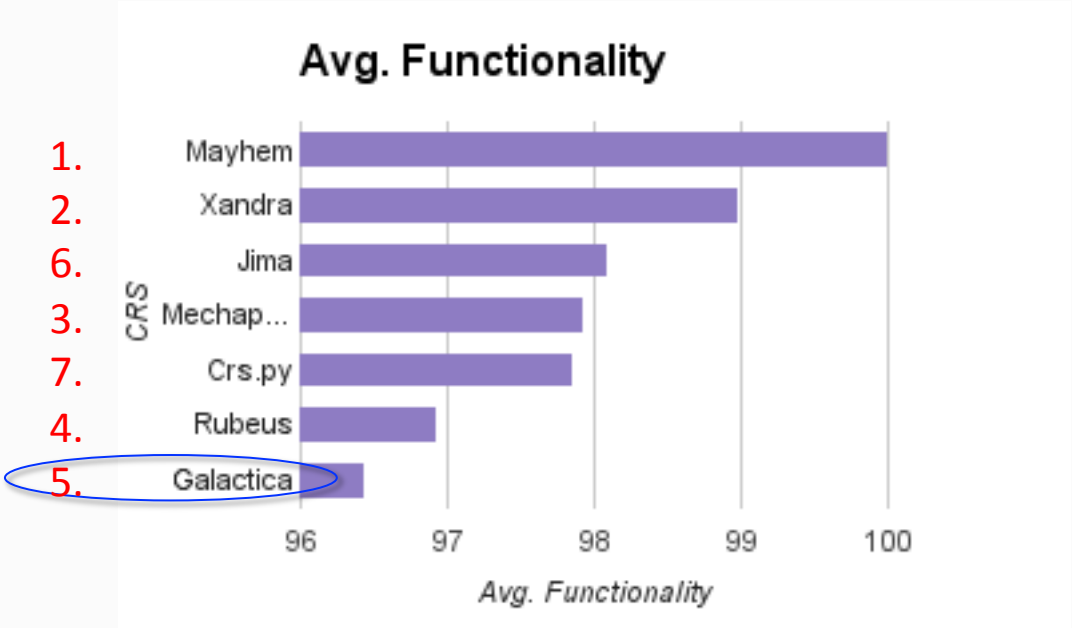
Availability: time overhead



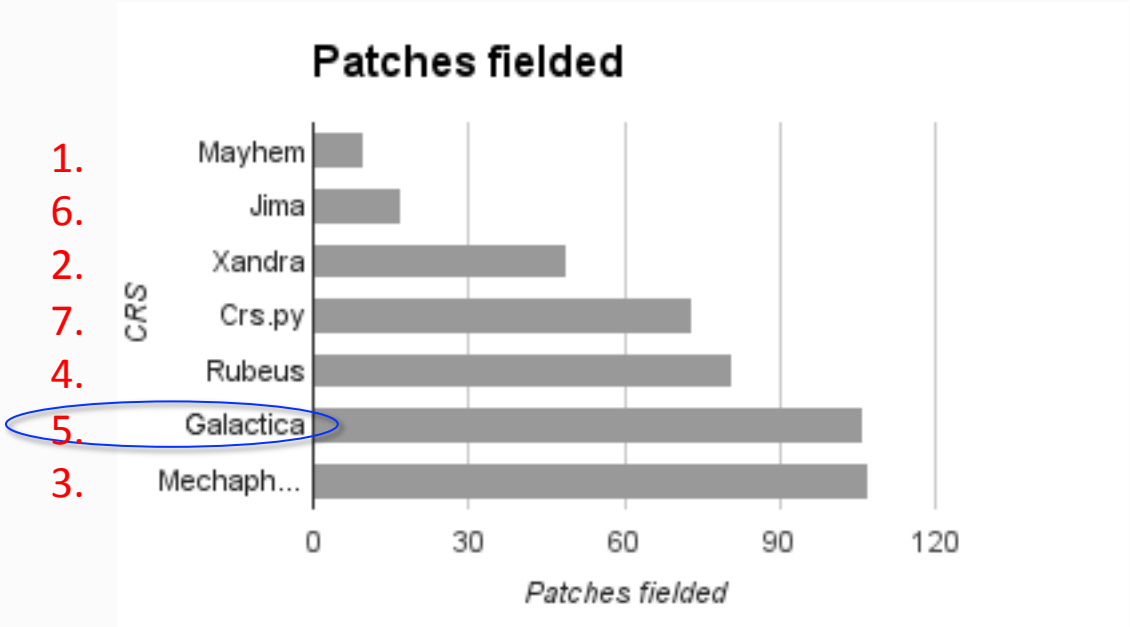
Availability: memory overhead



Availability: functionality



Availability: Submissions of RCBs



Each submission will cause next round's score to be 0!

Lessons learned

- Availability score is more important than Security score and Evaluation score in CGC.

- Opponent teams are not good at exploits, so it's safe to keep original CBs without any penalty.
  - All teams exploited 26/82 CBs together.

Machine vs. Human

DEFCON CTF 2016

- Day 1: last
- Day 2: 3rd to last
- Day 3: last

Scoreboard | DEF CON CTF 2016 b1o0p started round 127

Scoreboard

place	score	team
1	15	PPP
2	14	b1o0p
3	13	DEFKOR
4	12	HITCON
5	11	KaisHack GoN
6	10	Eat Sleep Pwn Repeat
7	9	LC↯BC
8	8	binja
9	7	pasten
10	6	9447
11	5	Shellphish
12	4	!SpamAndHex
13	3	Mayhem
14	2	侍
15	1	Dragon Sector

- Human
  - copy opponents' patches
    - PPP: embed backdoors in their RCBs
  - reconstruct opponents' exploits
- Machine
  - first to generate exploits against a "arbitrary write byte 0" vulnerability
  - first to generate exploits against an obfuscated CB

Some Thoughts

- Machines are good at
  - finding low-level bugs
  - attack: defeat simple obfuscation
  - defense: deploy generic defenses quickly
  - defense: generate variations of programs (moving targets)
- Machine are not good at
  - find high-level bugs
  - attack: generate advanced exploits
  - defense: deploy vulnerability–specific patches
- Future of machines
  - machine learning?

Conclusion

- CGC is a great pioneer project in making automated defense and attack into practice.
- It is the first attempt to make such a system work. It's reasonable some game rules are not properly set.
- It successfully stimulates the creation of 7 prototype systems, and proves automated defense and attack is possible. It will lead a wave of research and industry efforts.
- The machine is rising!

Thanks!
Q&A