Compelling CTF Challenges

Matthew Savage
@bluepichu

Corwin de Boor
@strikeskids

(Simplified) Condition Variables

```
void wait() {
    me = alloc_node(
        awake=false,
        id=thread());
    insert_queue(me);
    sleep_until(me.awake);
    free_node(me);
}
```

```
void signal() {
    poll_queue();
    sleeper.awake = true;
    awaken(sleeper.id);
}
```

(Buggy) Condition Variables

Thread: Signaler	Thread: Waiter
	`me = alloc_node(...);` `insert_queue(me);`
`poll_queue();` `sleeper.awake = true;`	
	`sleep_until(me.awake);` `free_node(me);`
`awaken(sleeper.id);` 💥	

Plaid Party Planning

```
ubuntu@strikeskids:~$ ./partyplanning
Before we start, where would you like to help out the most? 0
And second most? 0
That's too far for us to help out
That's too far for us to help out
We bucket people into at most 31 bins
Who is helping out today?
Person 1: Waituck
What a bucket e43dc40
Person 2: Ben
What a bucket 3edf5301
Person 3: Matt
What a bucket 1d14f73c
Person 4: Zach
What a bucket c15840c3
Person 5: Carolina
What a bucket 4544be93
Waituck: I really hate planning parties
Carolina: Can't wait to get going
Zach: Can't wait to get going
Zach: What kind of music is your favorite? pwn
Matt: Can't wait to get going
Ben: Can't wait to get going
Carolina: Now, that's my kind of music.
Matt: We really should have food be easy to pick up
Ben: This party should really happen in Pittsburgh
Matt: What food from around Pittsburgh do you want? indian
Carolina: Now what kind of decorations with a pwn DJ and indian to eat?
Carolina: I'll get the person making food to help me out
Matt: I hate decorating!!!!
```

An easy way to introduce concurrency is through **division of labor**: planning a party

Bug-First Design

- Start with an **interesting bug**
  - Tricky to exploit
  - Appears in an unexpected way
  - Requires creativity to find
- Build up a service (and a story) around it

What makes a challenge compelling?

Challenges Should Be

- Fair
- Realistic
- Entertaining
- Educational

Challenges Should Be » Fair

- CTFs are a **competition**
  - Players will get annoyed if there are obvious signs of unfairness
- Relevant concerns
  - Does the reward for completing this challenge reflect its difficulty?
  - Does the challenge favor certain teams over others for reasons other than knowledge and ability?
  - Is it possible for one team to influence the availability or difficulty of the challenge for other teams?

Challenges Should Be » Realistic

- CTFs challenges should provide a scenario **grounded in reality**
  - Where's the value in understanding a system that couldn't really exist?
- Relevant concerns
  - Does the challenge provide an actual service, or is it just a thin wrapper around a bug?
  - Is the story behind the problem extremely contrived, or could this service really be written this way?

Challenges Should Be » Educational

- Most players play CTFs to **learn something** about security
  - Many players, especially new ones, will stop playing if your challenges fail to deliver on this front
- Relevant concerns
  - Does this challenge provide a new experience to the average competitor?
  - Is the knowledge gained from completing this challenge useful?
  - Could the concepts in this challenge apply more generally?

Challenges Should Be » Entertaining

- CTFs are also a source of fun
  - If players don't enjoy playing challenges, **they will probably stop playing the CTF**
- Relevant concerns
  - Is the service itself special or funny?
  - Does the challenge have a sense of novelty to it?
  - Would competitors enjoy the service outside of a CTF?

Bug-First Design » The Bad

- Focusing on the bugs can leave the rest **unrealistic**
- Coming up with an **entertaining** story is hard
  - But easy to post-select

Bug-First Design » The Good

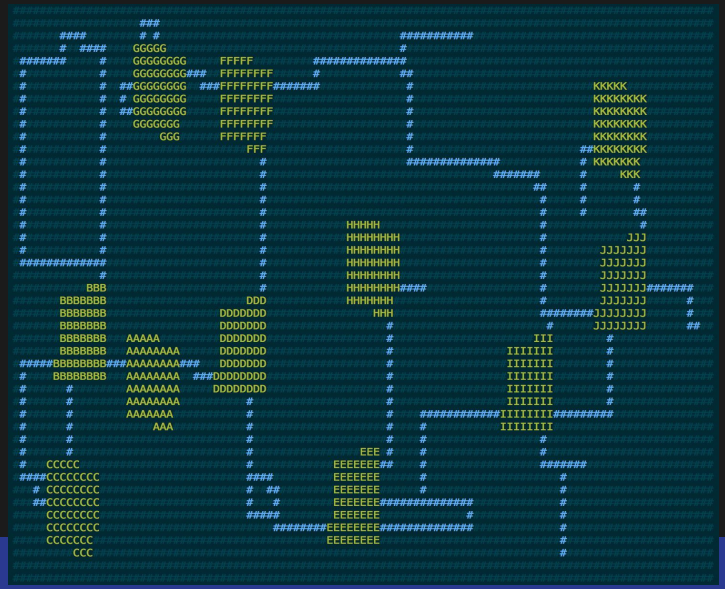
- Players **learn** because the bugs are chosen first
- Writers can spend more time on the bugs and exploits

Toaster Wars: Going Rogue

and Service-First Design

Toaster Wars

- Started as *Dungeonkit*, an attempt at writing a Mystery Dungeon-inspired roguelike web game

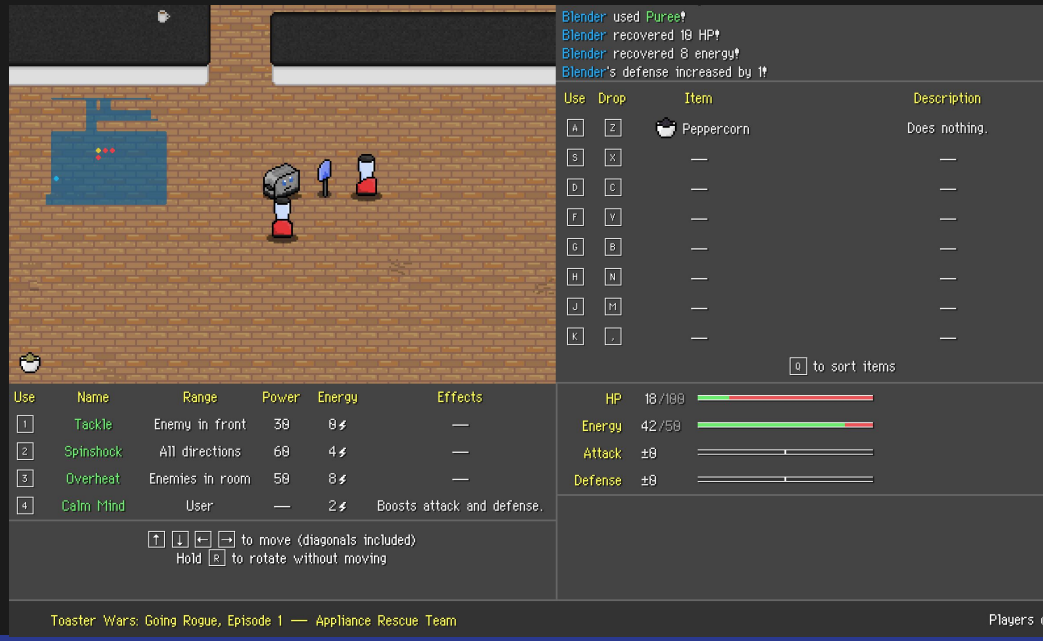


Toaster Wars » From Game to Problem

- While working, I would often **think about potential methods of breaking the game**
  - How much information would it take to get the PRNG state, and what could be done with it?
  - Is publicly identifying users by socket ID safe?

Toaster Wars » picoCTF 2017

- The easiest of these concepts became **picoCTF** problems
  - TW1 — JS source leak
  - TW2 — Logic bug in AI pathfinding
  - TW3 — ID collision in item pickup logic
  - TW4 — Trigger a race condition to skip floors



You can actually still play these challenges if you search "toaster wars going rogue" on Google!

Toaster Wars » PlaidCTF 2017

- The harder ideas became **PlaidCTF** problems
  - Light Flag — using a bug in how overworld interactions are handled, start a new game while already inside a dungeon

Toaster Wars » PlaidCTF 2017

- The harder ideas became **PlaidCTF** problems
  - Blazing Flag — exploit a race condition in Socket.IO's upgrade from long-polling to websocket, made possible by exposed socket IDs

Toaster Wars » PlaidCTF 2017

- The harder ideas became **PlaidCTF** problems
  - Stormy Flag — use a PRNG leak and several controlled PRNG advances to manipulate the PRNG to make the boss miss repeatedly

Service-First Design

- Start with a service, and turn bugs into problems **as they arise naturally**
  - *Toaster Wars: Light Flag* was an actual bug that I caught in testing
  - *Toaster Wars: Blazing Flag* arose from confusion in development about exposing Socket.IO socket IDs to the client

Service-First Design

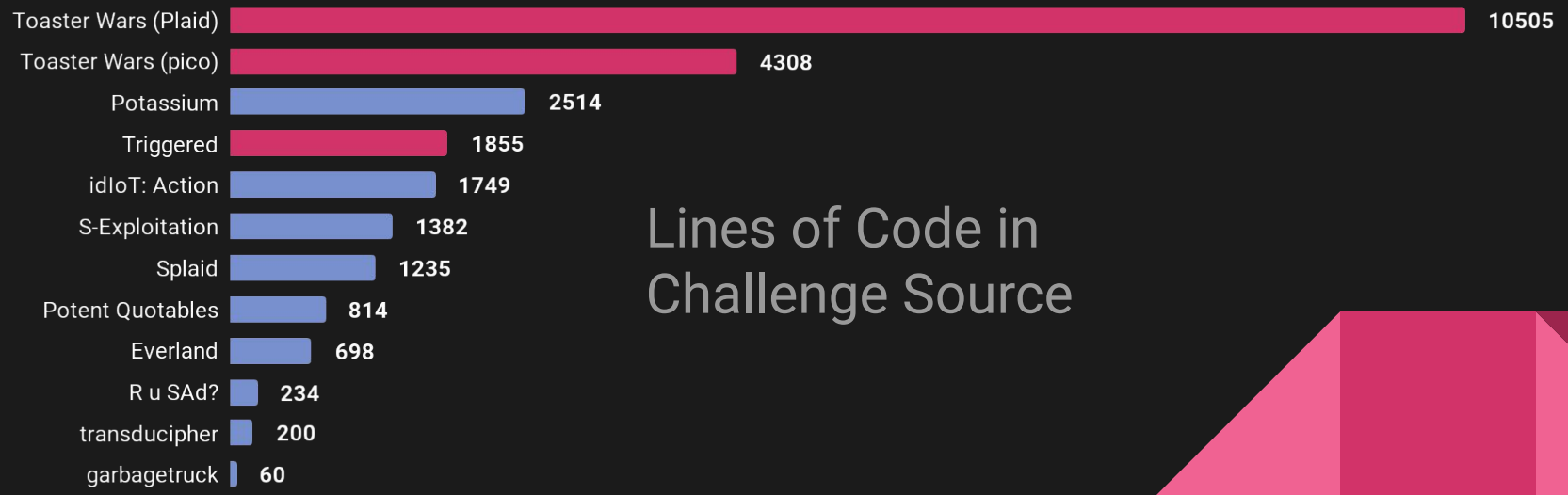
- If no exploitable bugs arise naturally, you can also **tweak interesting just-out-of-reach bugs** to produce a problem
  - *Toaster Wars: Stormy Flag* was not originally exploitable, so a weaker PRNG was introduced

Service-First Design » The Good

- Easy to write **realistic** challenges
  - Because the service stands on its own, it has an obvious intended purpose
- With a little extra effort, easy to write **entertaining** challenges
  - Via the service's purpose (e.g., a game)
  - Via the service's method of operation (e.g., a PL/pgsql webserver)

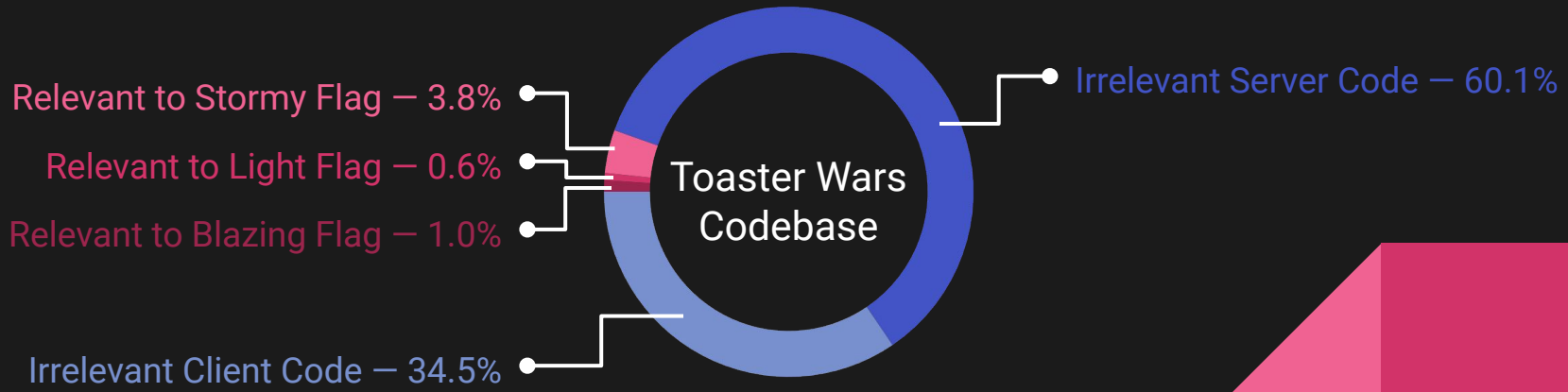
Service-First Design » The Bad

- Writing a complete, self-contained service from scratch requires **a lot of effort**



Service-First Design » The Bad

- Writing a complete, self-contained service from scratch requires **a lot of effort**
  - And a significant amount of effort is unrelated to the bugs!



Service-First Design » The Ugly

This method of writing problems can fail outright!

What happens if an exploitable bug never materializes?

```
162   strcat(dest, user_buffer);
163   memset(&format, 0, 0x3E8u);
164   v6 = sub_804A5E0(dest);
165   strcpy(&format, v6);
166   strcat(&format, "\n");
167   if ( v17 == 1 )
168     fputs(&format, stream);
169   else
170     fprintf(stream, &format);
171   fclose(stream);
```

```
37   v20 = (const char *)sub_8049F70(v4 + 12);
38   v21 = (FILE *)sub_80498D0((_DWORD *)v4);
39   fprintf(v21, v20);
```

```
16   *(_DWORD *)(a1 + 4) = strdup(s);
17   socket_printf(fd, "Welcome to 5 card draw ");
18   socket_printf(fd, *(char **)(a1 + 4));
19   socket_printf(fd, "\n");
20   v2 = 0;
```

General Considerations

Guessing

- Required guessing in a problem is **net-negative in value**
  - ✘ Fair (rewards a team for aspects other than security skill and knowledge)
  - ✔ Realistic (guessable passwords are a real-world risk)
  - ✘ Educational (you learn nothing by cold-guessing a password)
  - ✘ Entertaining (random guessing is not fun)
- Introducing guessing as a gatekeeper can **prevent teams from playing your problem**

Unintended Bugs

- Bugs not created intentionally by the author can lead to
  - Denial of service by competitors
  - Overvalued problems
- An issue **applicable to any problem-writing approach**
  - More likely to arise in the service-first approach due to code volume

Unintended Bugs » Prevention

- Testing!
  - Give testers the **same problem context** as competitors
    - Otherwise you get *Potent Quotables*
  - Make sure you **understand** the tester's potentially-unintended solution
    - Otherwise you get *SPlaid Birch*
  - Have testers look extra closely for **trivial bugs**
    - Otherwise you get *PPPIII*

Shared Environments

- Problems can arise when multiple teams operate in a single environment
  - Teams can read **other teams' data** (including exploits!)
  - Teams can prevent other teams from **persisting data**
  - Teams can **exhaust resources** required by all teams

Denial of Service

- Does solving the challenge allow a team to **make the service unresponsive or slow for others**?
  - Most prevalent in the web category
  - Can be caused by non-malicious activity
- Major impact on **fairness**
  - Teams can prevent others from solving

Denial of Service » Prevention

- **Isolate teams** to whatever extent is reasonable
  - Sandbox by IP
  - Sandbox by user account
- Place **proof-of-work** checks in front of expensive processes
  - Computation-based PoW (hashing)
  - Interaction-based PoW (captcha)

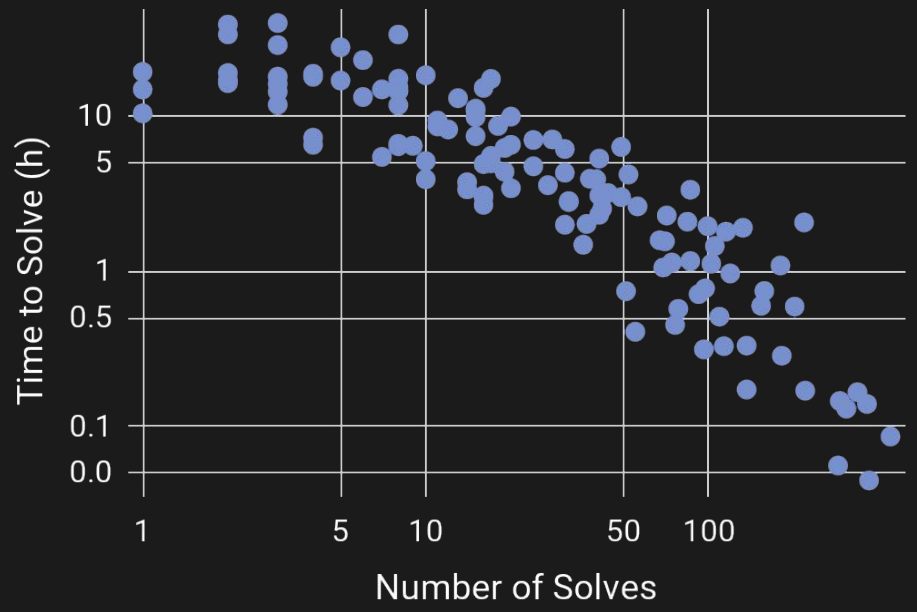
Point Values

- Having good point values is primarily about **fairness**
  - Your big effort for small reward
  - Others' small effort for large reward
- Point values should track the **difficulty** of challenges

Dynamic Scoring

- Determine point values from **the competition state**
  - What function?
- Use a **proxy metric** for difficulty
  - Time needed to solve the challenge
  - Total number of solves of the challenge

Dynamic Scoring » Proxy Metrics



- Time to solve
  - Approximates difficulty
  - Very gameable
- Number of solves
  - Gameable by creating teams
  - Approximates difficulty?

Static Scoring

- Choose values based on **perceived difficulty** before the competition
  - Start with a broad range
  - Tweak with feedback from testers
  - Maintain a realistic relationship with other problems

Scoring Systems

- Dynamic Scoring
  - ✔ Easy to implement
  - ✔ Unintended bugs are handled automatically
  - ✘ Scores decay over time
  - ✘ Competitors have less information up-front
  - ✘ Problem values depend on release time
  - ✘ Undervalues common skills
  - ✘ Gameable
- Static Scoring
  - ✔ Competitors know values up-front
  - ✔ Not gameable
  - ✔ Multipart problems
  - ✘ Misvaluation risk
  - ✘ High setup cost
  - ✔ Forces you to test problems

Scoring Systems » Example

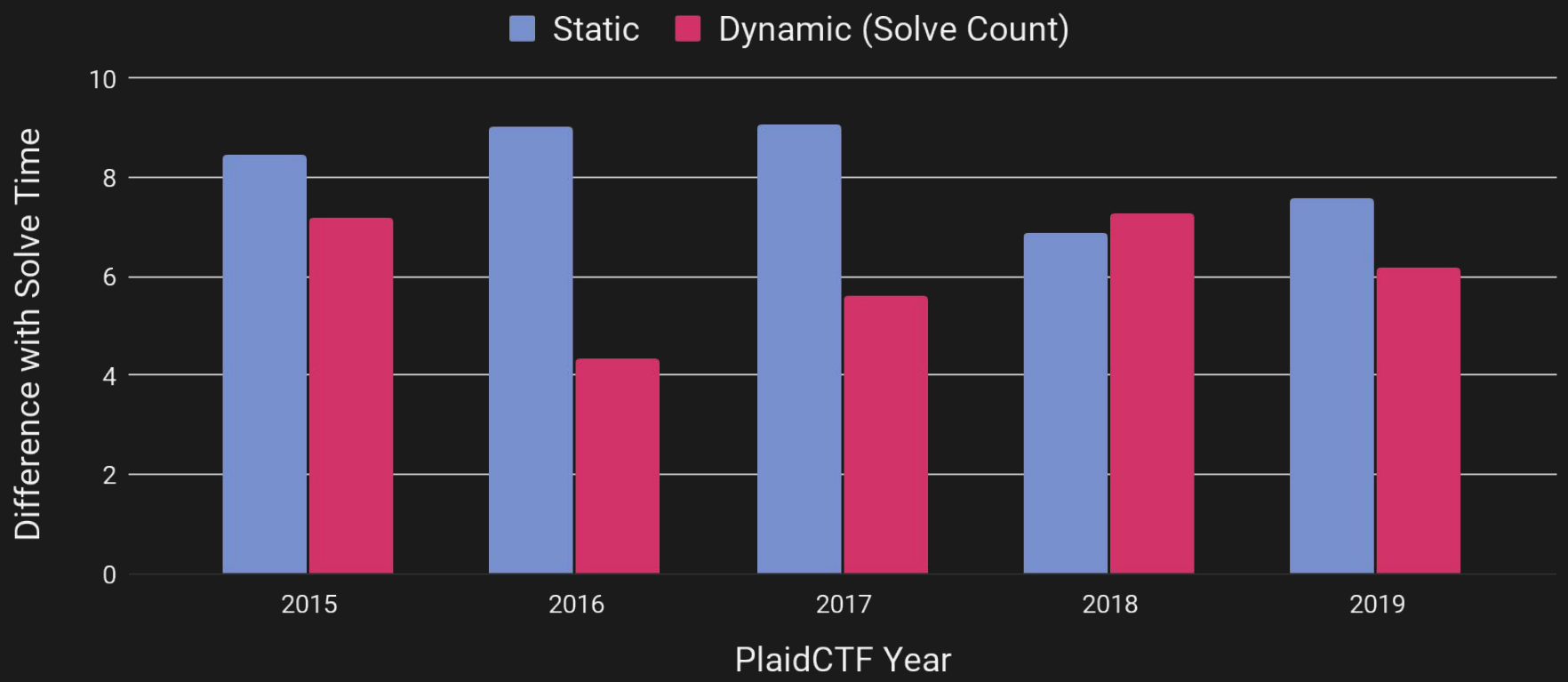
Dynamic (Solve Count)			Dynamic (Solve Time)			Static		
1	5912	217	1	5498	217	1	5840	217
2	4815	Tea Deliverers	2	4499	Tea Deliverers	2	4990	Tea Deliverers
3	4774	seoulplusbadass	3	4438	seoulplusbadass	3	4907	seoulplusbadass
4	4504	5BC	4	4155	5BC	4	4707	Balsn
5	4504	Balsn	5	4026	Sauercloud	5	4660	5BC
6	4452	A0E	6	3962	Balsn	6	4657	Sauercloud
7	4336	Sauercloud	7	3944	KaisHackGoN	7	4607	KaisHackGoN
8	4324	KaisHackGoN	8	3867	A0E	8	4357	A0E
9	3540	OpenBlue	9	3035	Dragon Sector	9	3857	Shellphish
10	3512	Dragon Sector	10	2980	OpenBlue	10	3690	Dragon Sector
12	3407	Shellphish	12	2963	Shellphish	14	3241	OpenBlue

Scoring Systems » Difference Metric

Measure how much scoreboards differ

$$\sum_{t \in \text{teams}} \left| \lg(r_1(t)) - \lg(r_2(t)) \right|$$

Scoring Systems » Example



Conclusion

- Design CTF challenges...
  - By having an interesting *bug first*, and enclosing it in a service
  - By building a realistic *service first*, and discovering bugs
- ...that are **fair**, **realistic**, **entertaining**, and **educational**...
- ...and get them tested.
  - To find *unintentional bugs*
  - To suggest *point values*

Any Questions?