



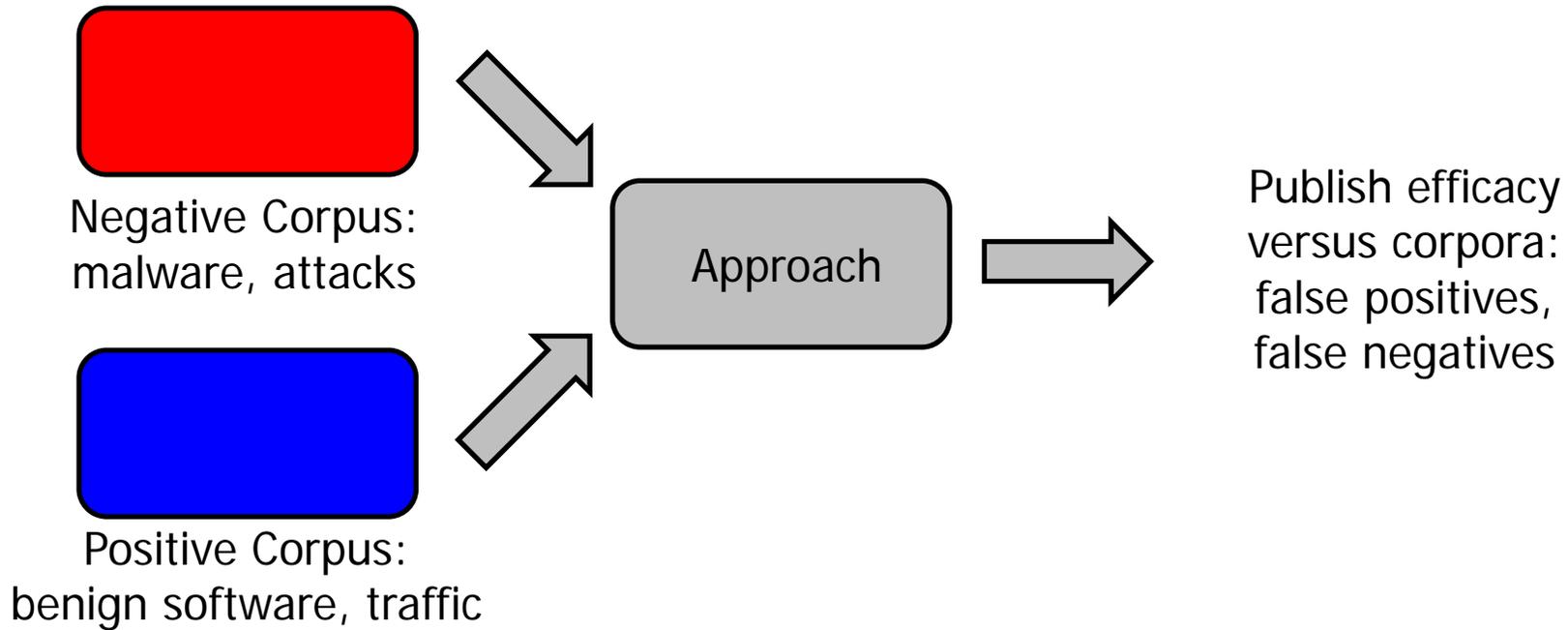
---

## Machine vs. Machine: Lessons from the First Year of Cyber Grand Challenge



“Autonomous cyber defense capabilities that combine the speed and scale of automation with reasoning abilities exceeding those of human experts.”

“During a final competition event, automated Cyber Reasoning Systems will compete against each other in real time.”

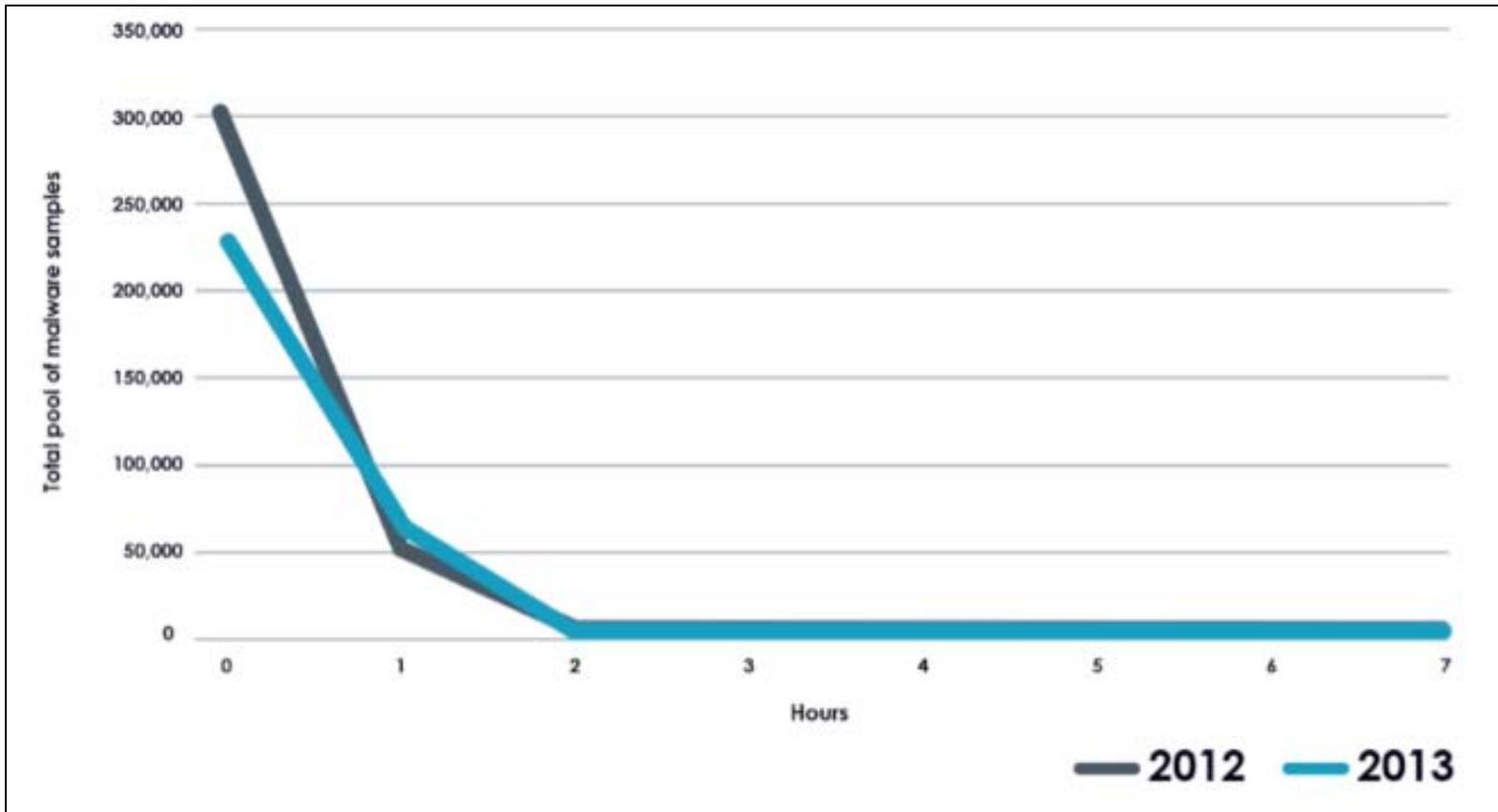




# Adversarial cycles



From FireEye:



\*

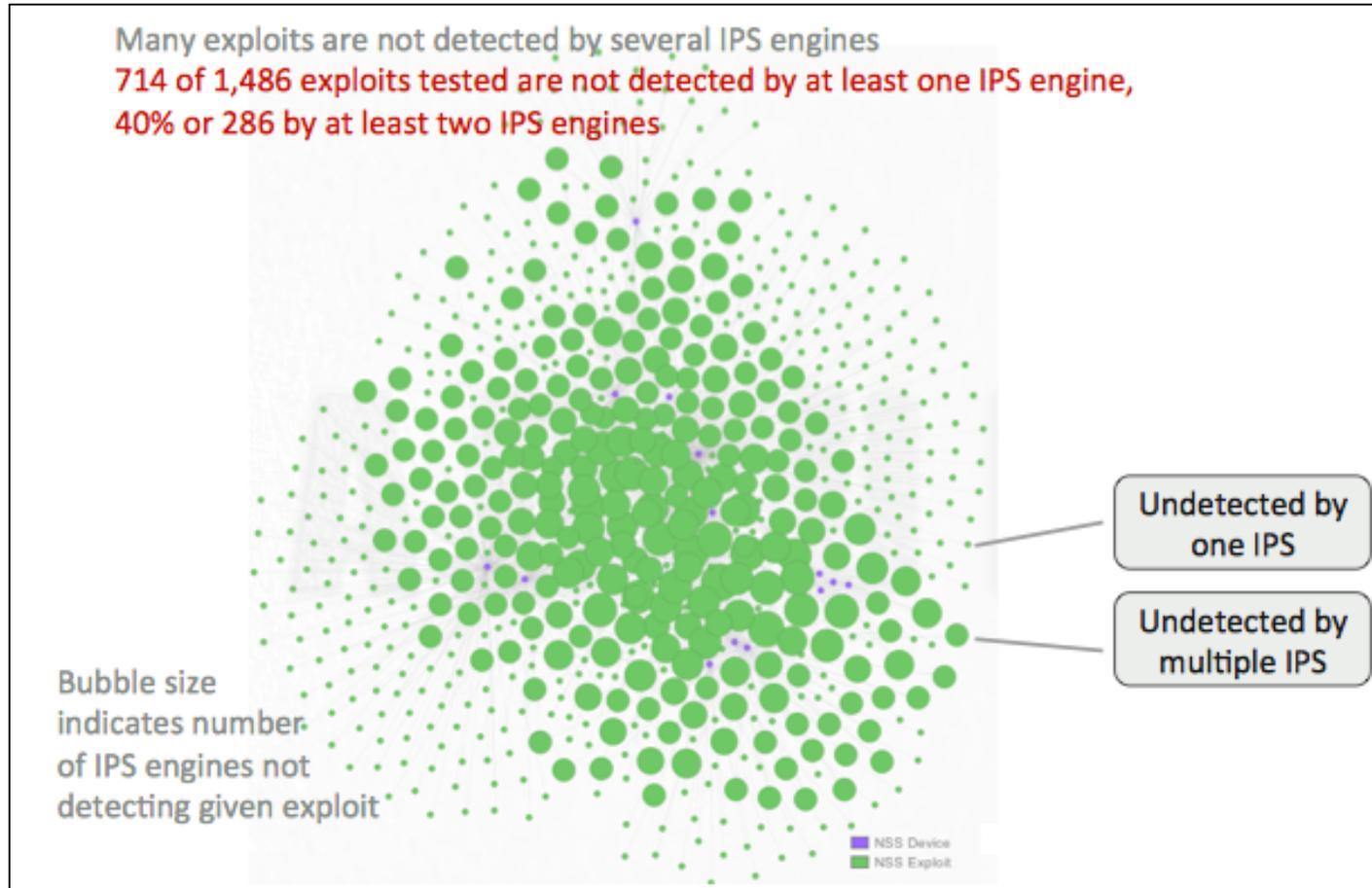
[\\* https://www.fireeye.com/blog/executive-perspective/2014/05/ghost-hunting-with-anti-virus.html](https://www.fireeye.com/blog/executive-perspective/2014/05/ghost-hunting-with-anti-virus.html)



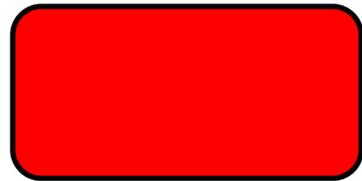
# Adversarial cycles



From NSS Labs, Frei & Artes:



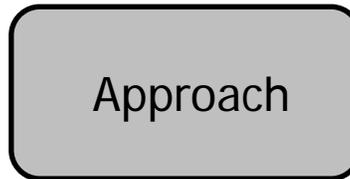
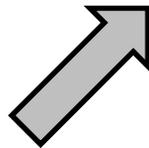
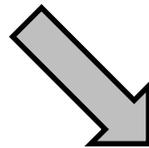
\* <https://media.blackhat.com/ad-12/Artes/bh-ad-12-cybercrime-kill-chain-artes-slides.pdf>



Negative Corpus:  
malware, attacks



Positive Corpus:  
benign software, traffic



Approach

"The enemy knows the system"  
-Shannon's Maxim

- 1) Adversary inspects approach  
"white box": full knowledge of the system
- 2) Adversary creates test cases to defeat approach



Publish efficacy in terms of  
*performance against adversary:*  
false positives,  
false negatives



“a product is a security product when it has sentient opponents”.  
- Dan Geer, 2005\*

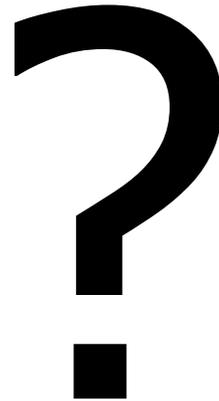
“Security involves making sure things work [...] in the face of an intelligent and malicious adversary”  
- Bruce Schneier, 2001\*\*

\*<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1514409>

\*\*[https://www.schneier.com/essays/archives/2001/05/foreword\\_to\\_security.html](https://www.schneier.com/essays/archives/2001/05/foreword_to_security.html)



## What kind of adversary?



"If your product fails because some gleeful clown discovers that he can be a superuser by typing 5,000 lowercase a's into a prompt, then said clown might not be all that sentient, but nevertheless, yours is a security product." \*

-Dan Geer

[\\*http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1514409](http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1514409)



# What kind of adversary?



\*\*

“If your product fails because some gleeful clown discovers that he can be a superuser by typing 5,000 lowercase a’s into a prompt, then said clown might not be all that sentient, but nevertheless, yours is a security product.” \*

-Dan Geer

\*<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1514409>

\*\* <https://twitter.com/uvasecdep/status/589087467065761792>



How can we build a *standard experimental model*:  
reproducible, controlled, and consistent  
That includes an **intelligent adversary**?

## Reproducible

Same level of  
adversarial inspection  
expertise every audit

## Controlled

To provide for control  
cases, adversary must  
be capable of *forgetting  
everything it knows*

## Consistent

Can't get tired or  
change its efficacy at  
10k, 100k, 10000k lines  
of code

Solution: software safety becomes the expert domain of machines



Difficulties of adversarial experimentation:

- Old, vulnerable computers in an isolated network, with real exploits:
  - Vulnerable Linux\*
  - Hacksys Extreme Vulnerable Driver\*\*
  - metasploit

Challenges:

- Old, real exploits have OS-specific interchangeable components
- Network isolation is difficult and mistakes happen
- Old, real exploits still work on many unpatched computers
- Can't safely research novel flaws, thus:
- Can't safely research defense against novel flaws
- Adaptation and counter-adaptation are basically off limits in this model

\* <http://distrowatch.com/table.php?distribution=dvl>

\*\* <http://www.payatu.com/hacksys-extreme-vulnerable-driver/>



Adversarial experimentation, with DARPA DECREE:

- “The world from scratch”:
  - Incompatible protocols from scratch
  - 131 samples of incompatible server software from scratch
  - Incompatible binary format from scratch
  - Incompatible loader from scratch
  - Incompatible ABI from scratch
  - Brokered networking
- Network isolation is totally irrelevant: run it on your desktop
- No exploit for DECREE will ever affect a real world computer
- Non-reusable: protocol code and platform code are worthless for any real world task
- Novel flaws? Novel defenses? No problem
- Adaptation and counter-adaptation are safe and easy

\* <http://distrowatch.com/table.php?distribution=dvl>

\*\* <http://www.payatu.com/hacksys-extreme-vulnerable-driver/>

Released as Open Source:

<https://repo.cybergrandchallenge.com/>

<https://github.com/CyberGrandChallenge/>

- Released as a Linux layer, portable via 7 system calls to any host OS
- Measurement:
  - Functionality loss via programmatically replayed network tests
  - Performance loss via instrumentation
  - Testable presence of vulnerability via known harmful inputs
- Experiments:
  - Bug hunting & software safety technology
  - Exploit mitigation measurement
  - Adversarial automation
  - Your approach here
- Tools:  
PIN\*, IDA Pro\*\*

\* <https://github.com/CyberGrandChallenge/cgc-release-documentation/blob/master/walk-throughs/pin-for-decree.md>

\*\* <http://idabook.com/cgc/>



DECREE

(<https://github.com/CyberGrandChallenge>)



- DARPA Experimental Cyber Research Evaluation Environment
- 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 socketpairs
      - Clean bidirectional communication
      - Automatically created by system on startup
      - Shared between all processes in an IPC CB



# Challenge Binaries



- No filesystem access, no network access
- Userspace only and statically linked [Qu2011]
- No code-reuse except a common "libc"
- Compiled Binaries only (not hand coded)
  - Always available
  - Ground truth

```

struct tun_struct *tun = ...;
struct sock *sk = tun->sk;
if (!tun)
    return POLLERR;
/* write to address based on tun */

```

"A null pointer dereference vulnerability (CVE-2009-1897) in the Linux kernel, where the dereference of pointer tun is before the null pointer check. The code becomes exploitable as **gcc optimizes away** the null pointer check [10]" [Wang2013]

RedHat 7.0	- (default Sendmail 8.11.0)	does not crash
RedHat 7.2	- (default Sendmail 8.11.6)	does not crash
RedHat 7.3 (p)	- (patched Sendmail 8.11.6)	does not crash
RedHat 7.0	- (self compiled Sendmail 8.11.6)	crashes
RedHat 7.2	- (self compiled Sendmail 8.11.6)	crashes
RedHat 7.3	- (self compiled Sendmail 8.11.6)	crashes
Slackware 8.0 (p)	- (patched Sendmail 8.11.6 binary)	crashes
Slackware 8.0	- (self compiled Sendmail 8.12.7)	does not crash
RedHat 7.x	- (self compiled Sendmail 8.12.7)	does not crash
	(p) - patched box	

"Due to the nature of the overflowed buffer declaration (static), exploitation of this issue is **highly dependent on the way compiler orders the static data** in the data segment" [LSD2003]

- Wide availability of "lifters" (these are open source x86)
  - BAP (BAP IR) - <https://github.com/BinaryAnalysisPlatform/bap/>
  - BitBlaze (VINE IR) - <http://bitblaze.cs.berkeley.edu/>
  - McSema (LLVM IR) - <https://github.com/trailofbits/mcsema/>
  - QEMU (TCG IR) - <http://www.qemu.org/>
  - Valgrind (VEX IR) - <http://www.valgrind.org/>



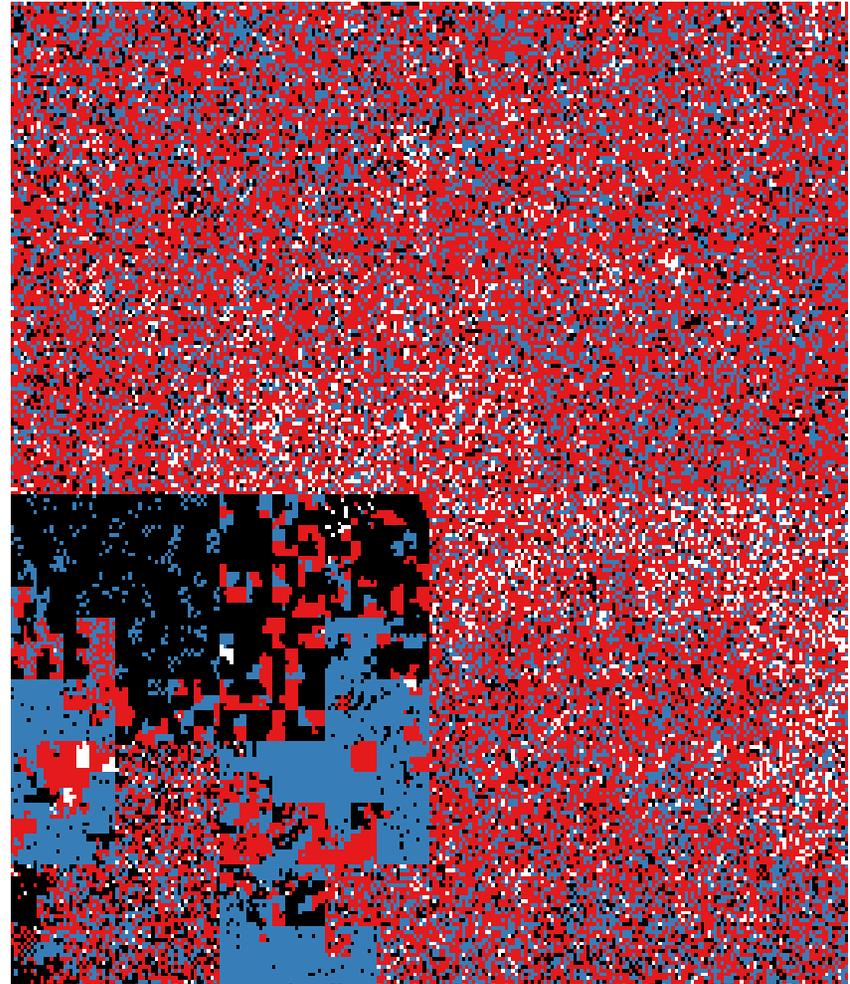
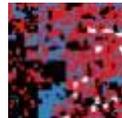
# Text vs Code of trivial program



CGC

Linux

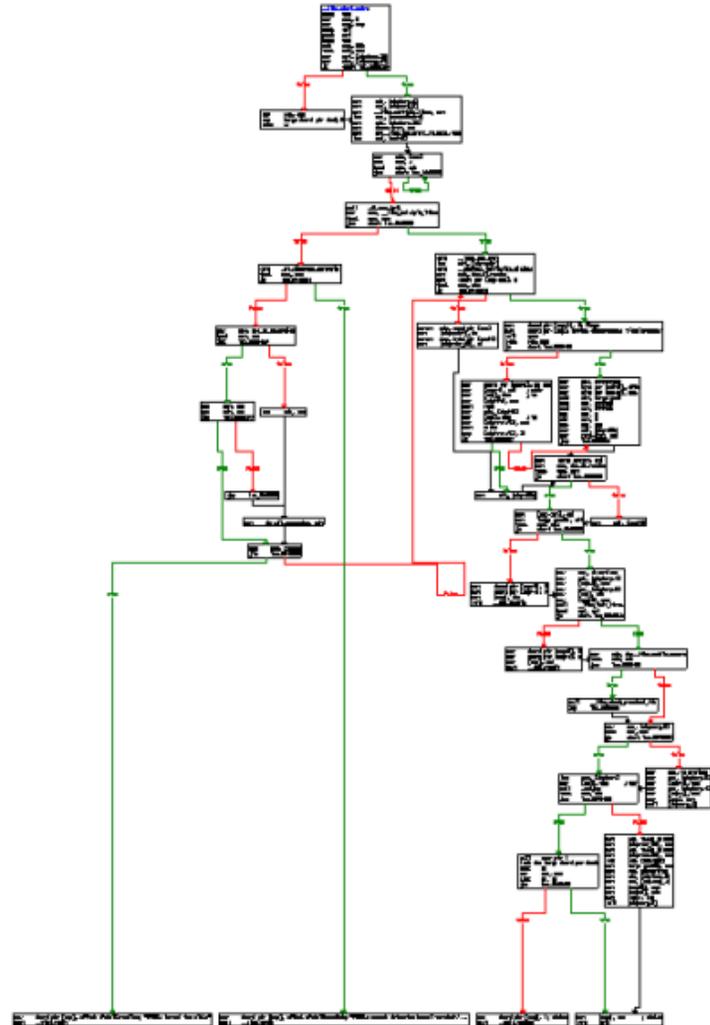
Text
Code
0xff





# CGC

# Linux





# 131 Challenges

72 CC Files

1236 H Files



# DECREE Qualifier Challenge Sets:

---



9 CC

5 IPC

# 131 Challenges

9 Tokenized

72 CC Files

1236 H Files

122 C



# DECREE Qualifier Challenge Sets:

---



131 Challenges  
**72 CC Files**  
1236 H Files  
1996 C Files



# DECREE Qualifier Challenge Sets:

---



131 Challenges

72 CC Files

**1236 H Files**

1996 C Files

> 6K Functions



72 CC Files

1236 H Files

**1996 C Files**

> 6K Functions

> 190K H LOC



# DECREE Qualifier Challenge Sets:

---



1236 H Files

1996 C Files

> **6K Functions**

> 190K H LOC

> 7K CC LOC



# DECREE Qualifier Challenge Sets:

---



1996 C Files  
> 6K Functions  
**> 190K H LOC**  
> 7K CC LOC  
> 200K C LOC



# DECREE Qualifier Challenge Sets:

---



- > 6K Functions
- > 190K H LOC
- > **7K CC LOC**
- > 200K C LOC
- 590 POVs



# DECREE Qualifier Challenge Sets:

---



> 190K H LOC  
> 7K CC LOC  
**> 200K C LOC**  
590 POVs  
> 10K Polls



## DECREE Qualifier Challenge Sets:

---



> 7K CC LOC

> 200K C LOC

**590 POVs**

> 10K Polls



## DECREE Qualifier Challenge Sets:

---



> 200K C LOC

590 POVs

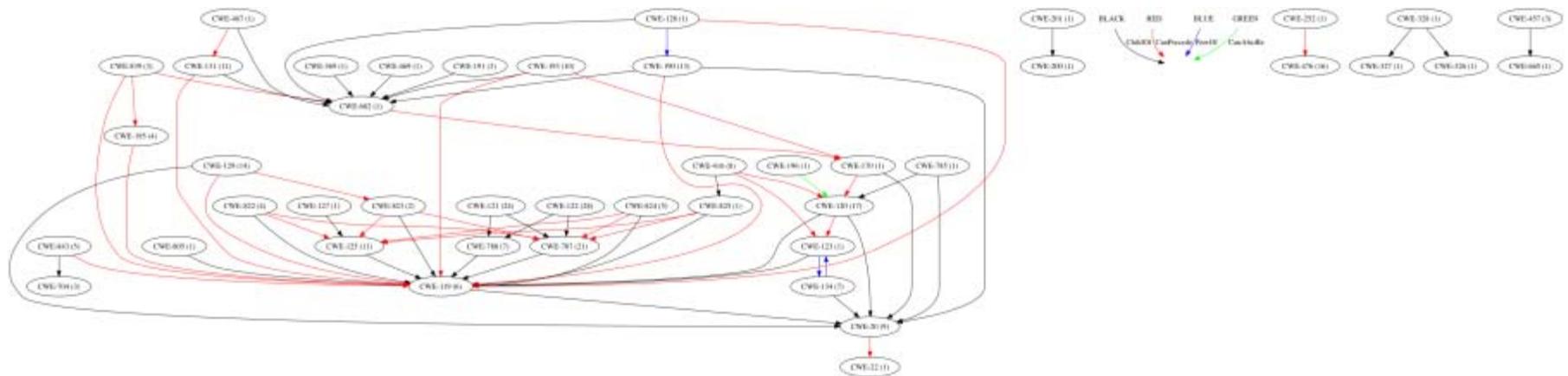
**> 10K Polls**



# Challenge Sets CWE



- 53 Different CWEs (as identified by Challenge Set authors)
- Common ones:
  - 28 CWE-122 Heap Overflow
  - 24 CWE-121 Stack Overflows
  - 16 CWE-476 Null Pointer Dereference
  - 13 CWE-190 Integer overflow or wraparound
  - 8 CWE-416 Use after Free
  - 7 CWE-134 Uncontrolled Format String





# Challenge Set Complexity - Callgraphs

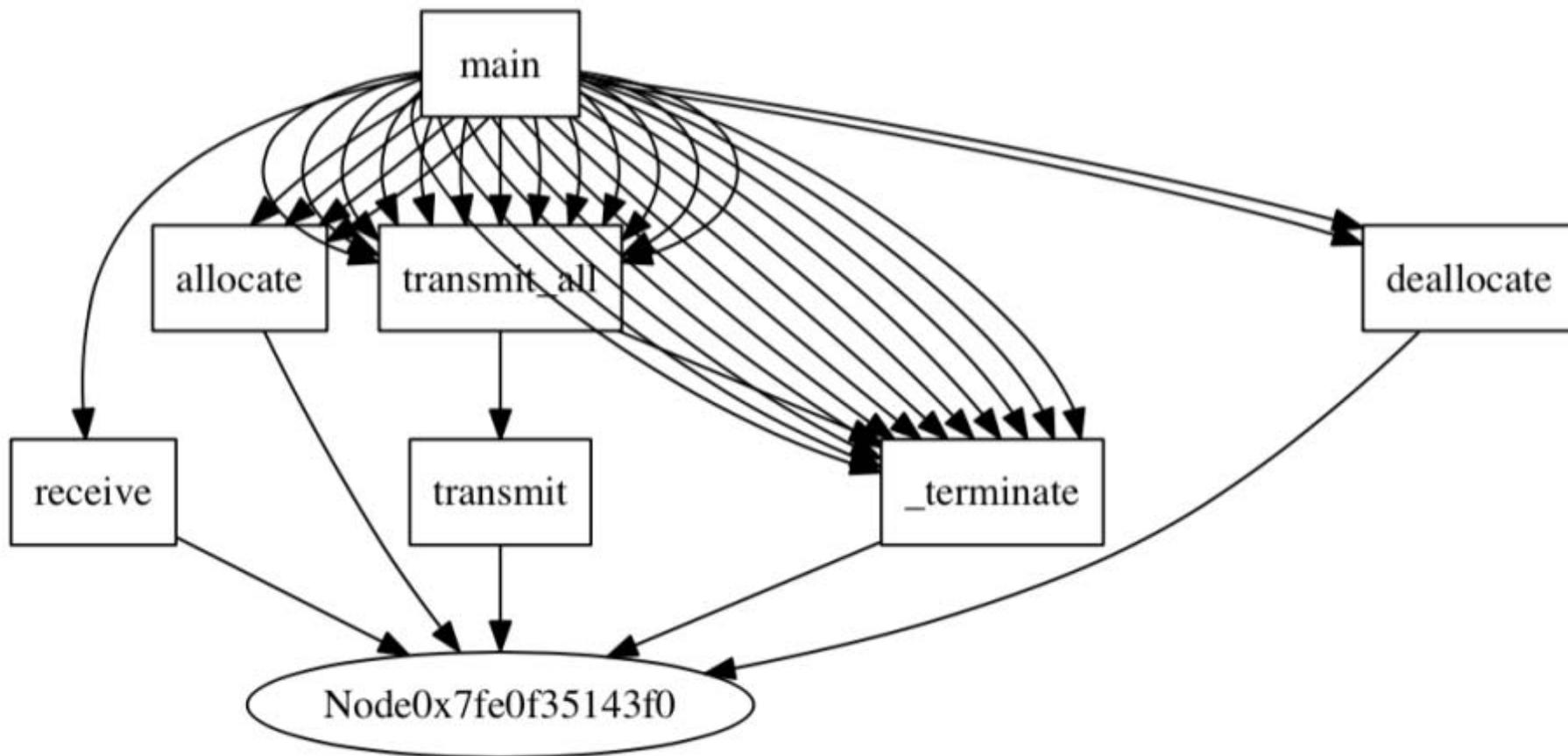
---



- Totals :
  - Total Nodes
- Simplest : YAN01\_00012
  - 8 Nodes, 12 Edges
- Most Complex: NRFIN\_00026
  - 1041 Nodes, 7290 Edges
  - NRFIN\_00032 : 240 Nodes, 1121 Edges
- Average:
  - 81 Nodes, 238 Edges

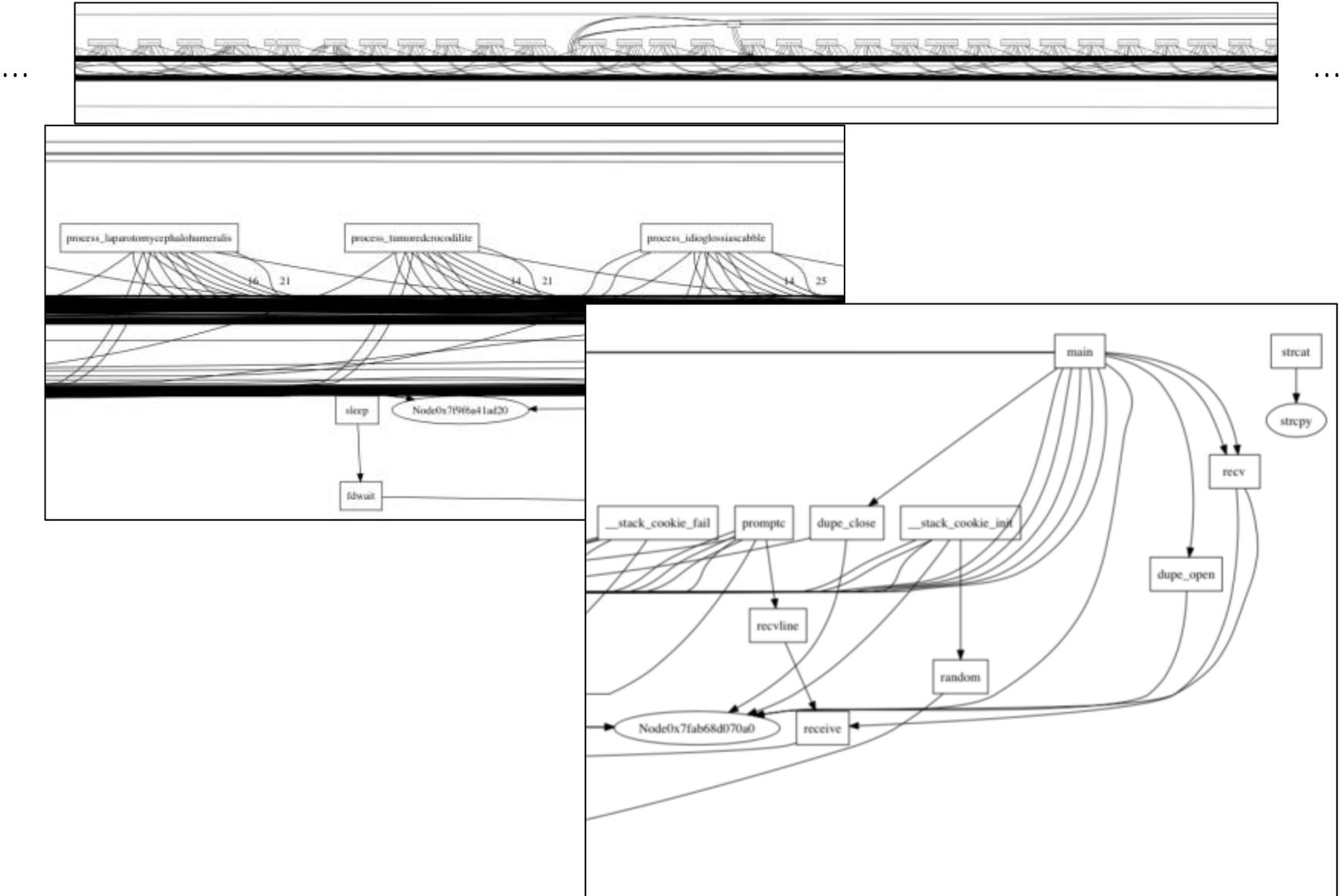


# YAN01\_00012 Callgraph



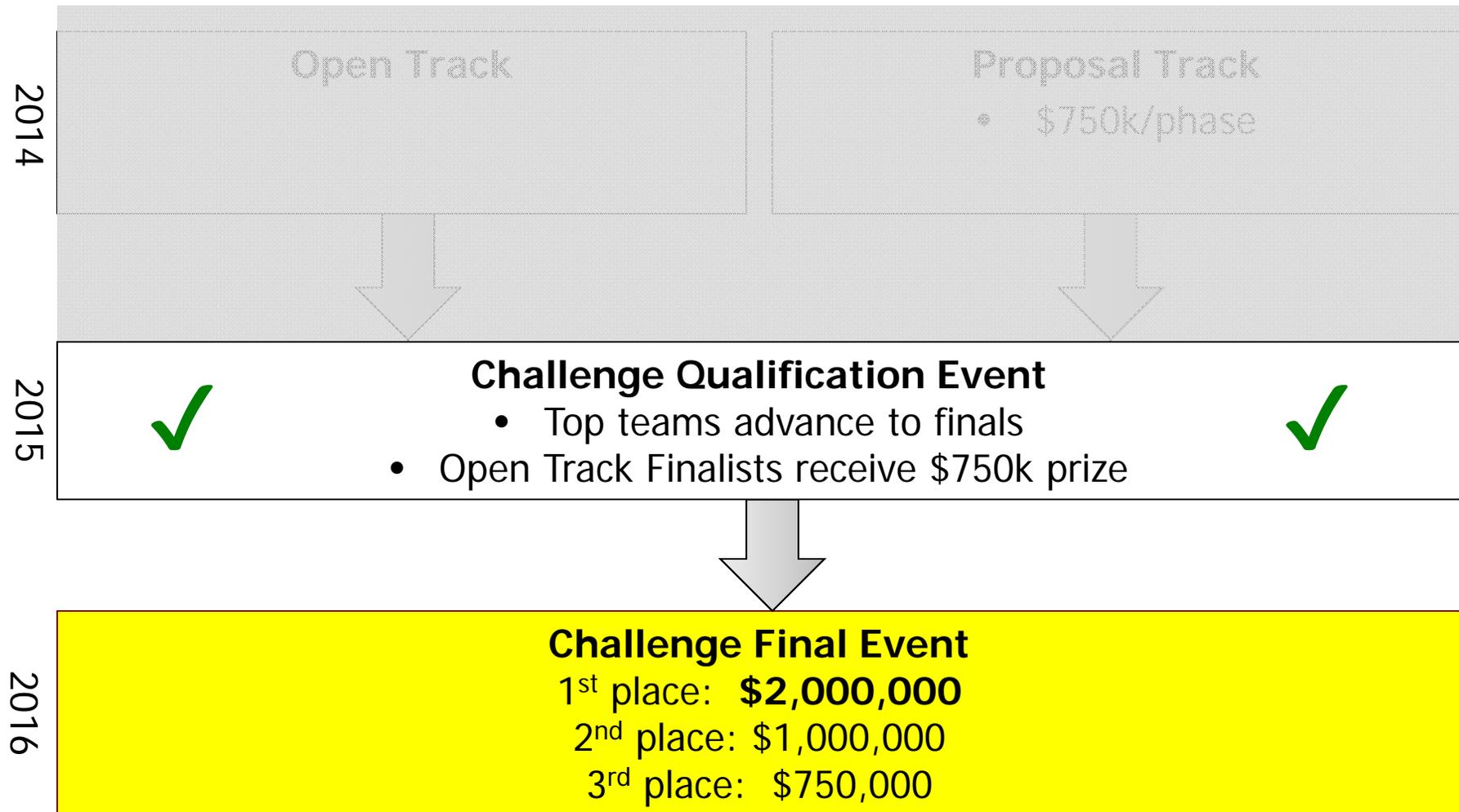


# NRFIN\_00026 Callgraph





# Cyber Grand Challenge: Scheduled Events





## Lessons Learned

---



In a nutshell :

```
char buf[64];  
receive(STDIN, buf, 128, NULL);
```

Reproducibility – Does it crash?:

1. Perform a single 128 byte “write” to CADET\_00001
2. Perform two consecutive 64 byte “writes” to CADET\_00001



In a nutshell :

```
char buf[64];  
receive(STDIN, buf, 128, NULL);
```

Reproducibility – Does it crash?:

1. Perform a single 128 byte “write” to CADET\_00001
2. Perform two consecutive 64 byte “writes” to CADET\_00001

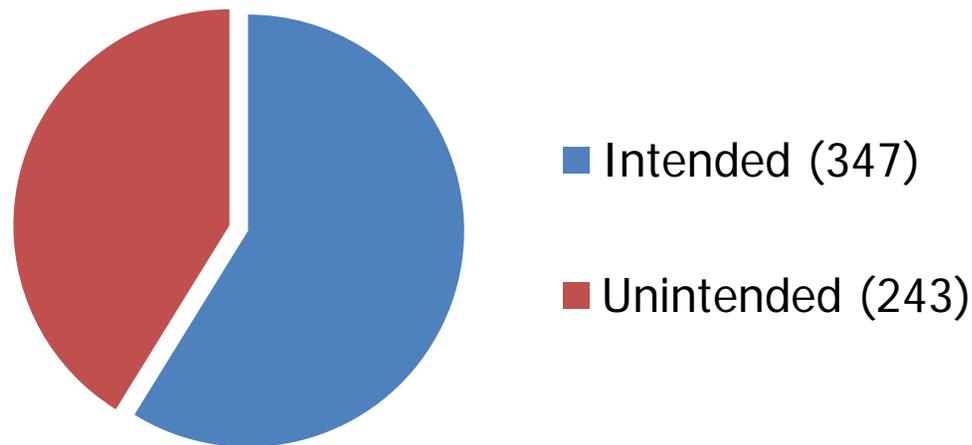
More Deterministic Version :

```
char buf[64];  
receive_delim(STDIN, buf, 128, '\n');  
...
```



- “These are all problems we should and could have detected on our own, but did not. Had they gone undetected, they could have led to security holes exploitable by hackers, particularly if they had access to source code. Our experience suggested **the use of formal methods and tools is an essential part of improving the security properties of software. Using experienced, security-conscious programmers is not enough.**” [Fagin 2013]\*

## POVS (590)



\* <http://ironsides.martincarlisle.com/ICRST2013.pdf>



Bugs are everywhere ...



Trail of Bits CRS

@skynet\_ebooks

+ Follow

[#CRSFacts](#) During development, we reported 5 XED bugs (including ones in JCXZ/JECXZ/JRCXZ), 3 IDA bugs (jump tables), and 1 0day. [#DARPAACGC](#)

RETWEETS

3

FAVORITES

2



7:16 AM - 4 Jun 2015





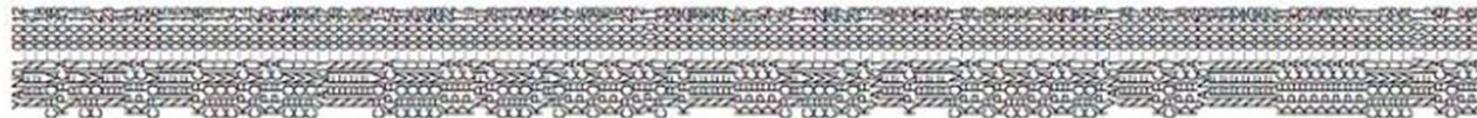
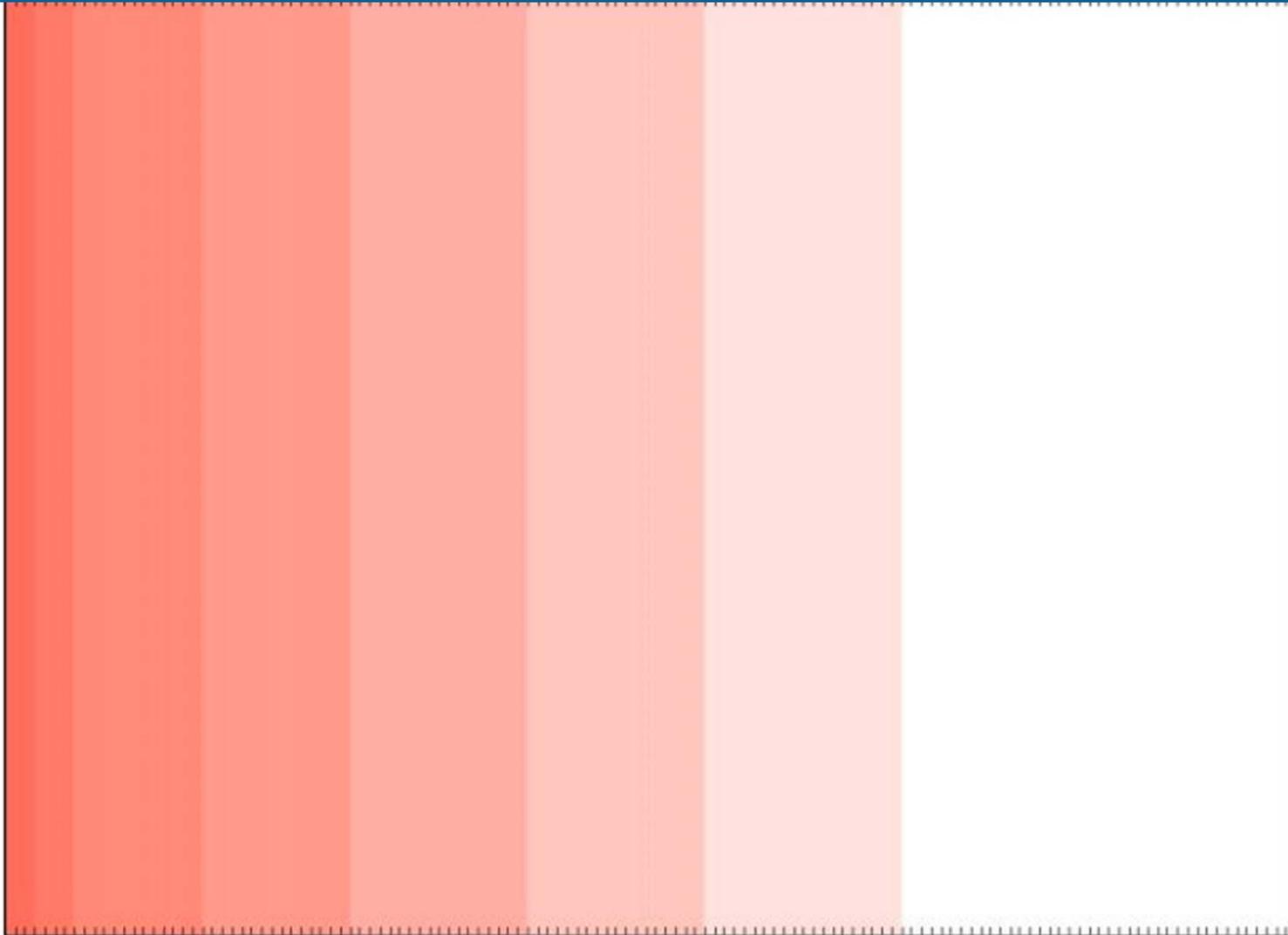
*"CS deliveries will be evaluated for their ability to differentiate competitors.*

*A Challenge Set that is solved by no competitors will receive a minimal score;  
a CS that is solved by all competitors will also receive a minimal score.*

*Sets that are able to differentiate CQE competitors in a unique way will receive high marks."*

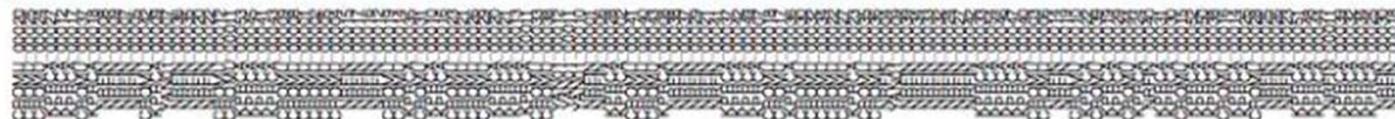
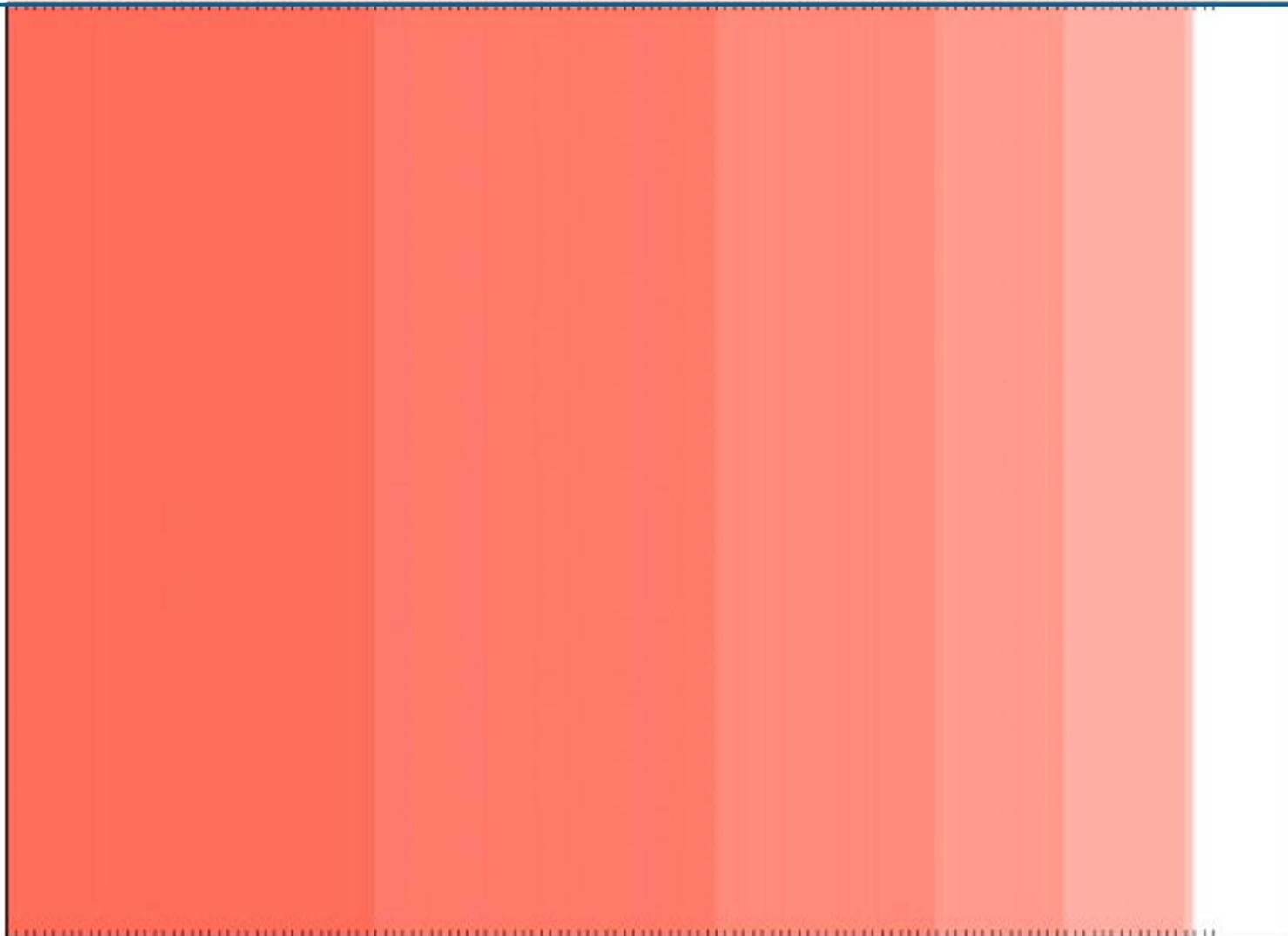


Finalists POV





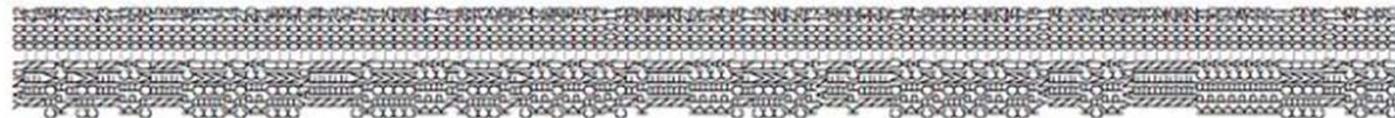
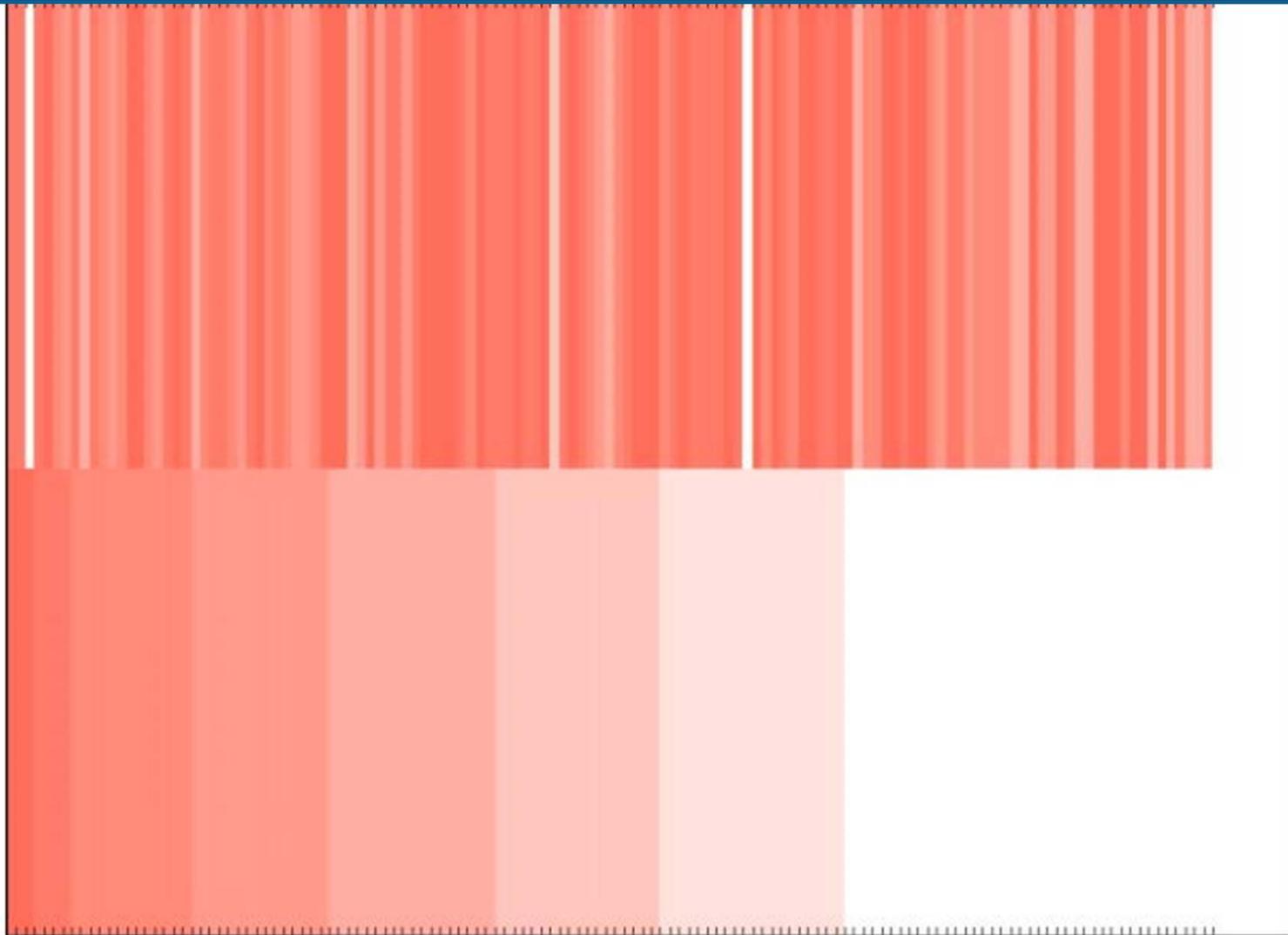
Finalists Avail



Approved for Public Release, Distribution Unlimited



Finalists - POV on Bottom, Avail on Top

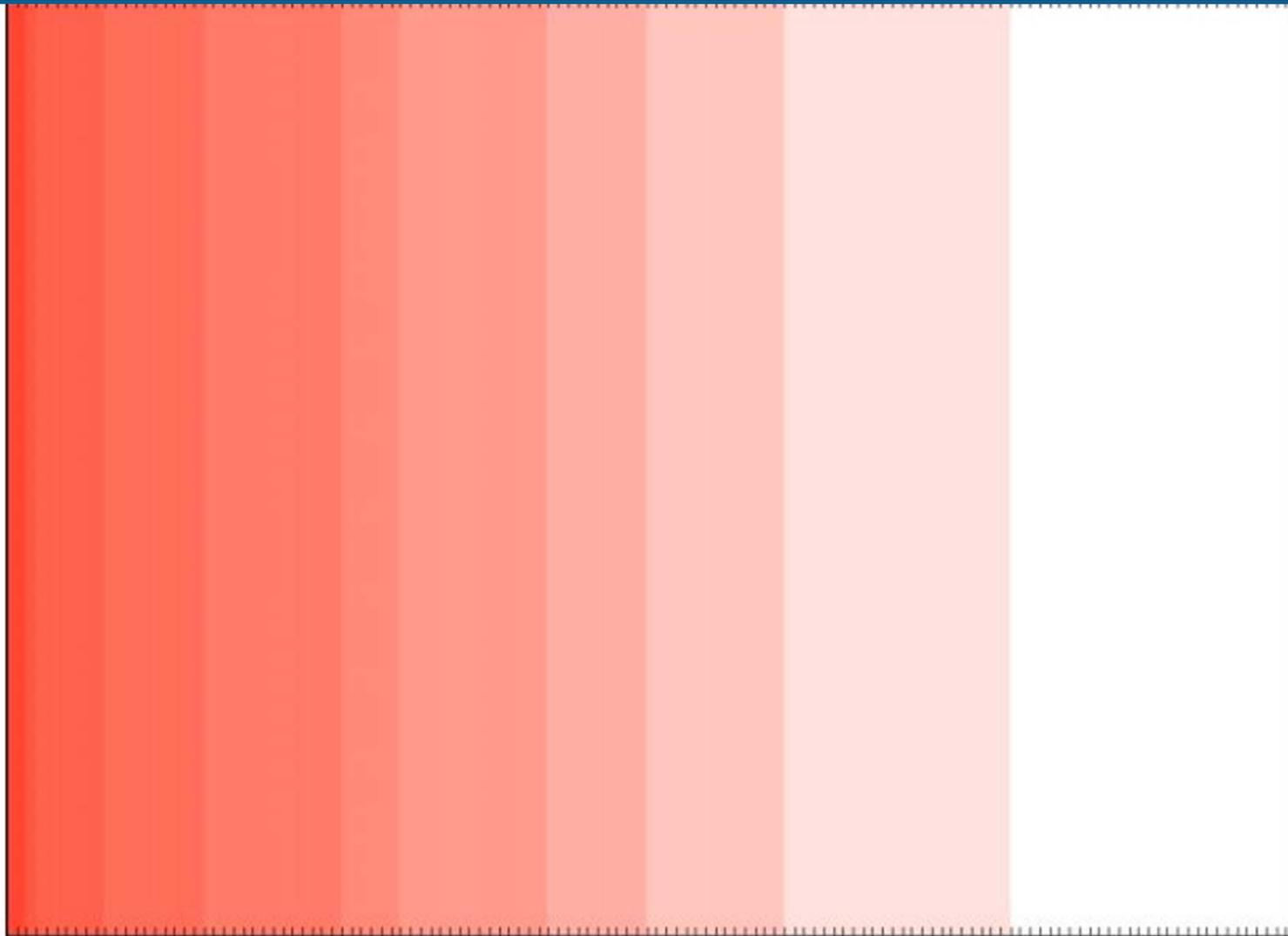




```
85  #ifdef PATCHED
86      uint8_t bytes_needed = 2 + (8 * qty) + 2;
87  #else
88      int bytes_needed = 5120;
89  #endif
90      char *buf = NULL;
91
92      buf = calloc(bytes_needed);
93
94      if (0 > (bytes_read = recv_until_delim(STDIN, buf, bytes_needed, FIN))) {
95          ret = bytes_read;
96          goto bail;
97      }
98  #ifdef PATCHED
99      if (bytes_needed != bytes_read) {
100          ret = ERRNO_RECV;
101          goto bail;
102      }
103  #endif
```



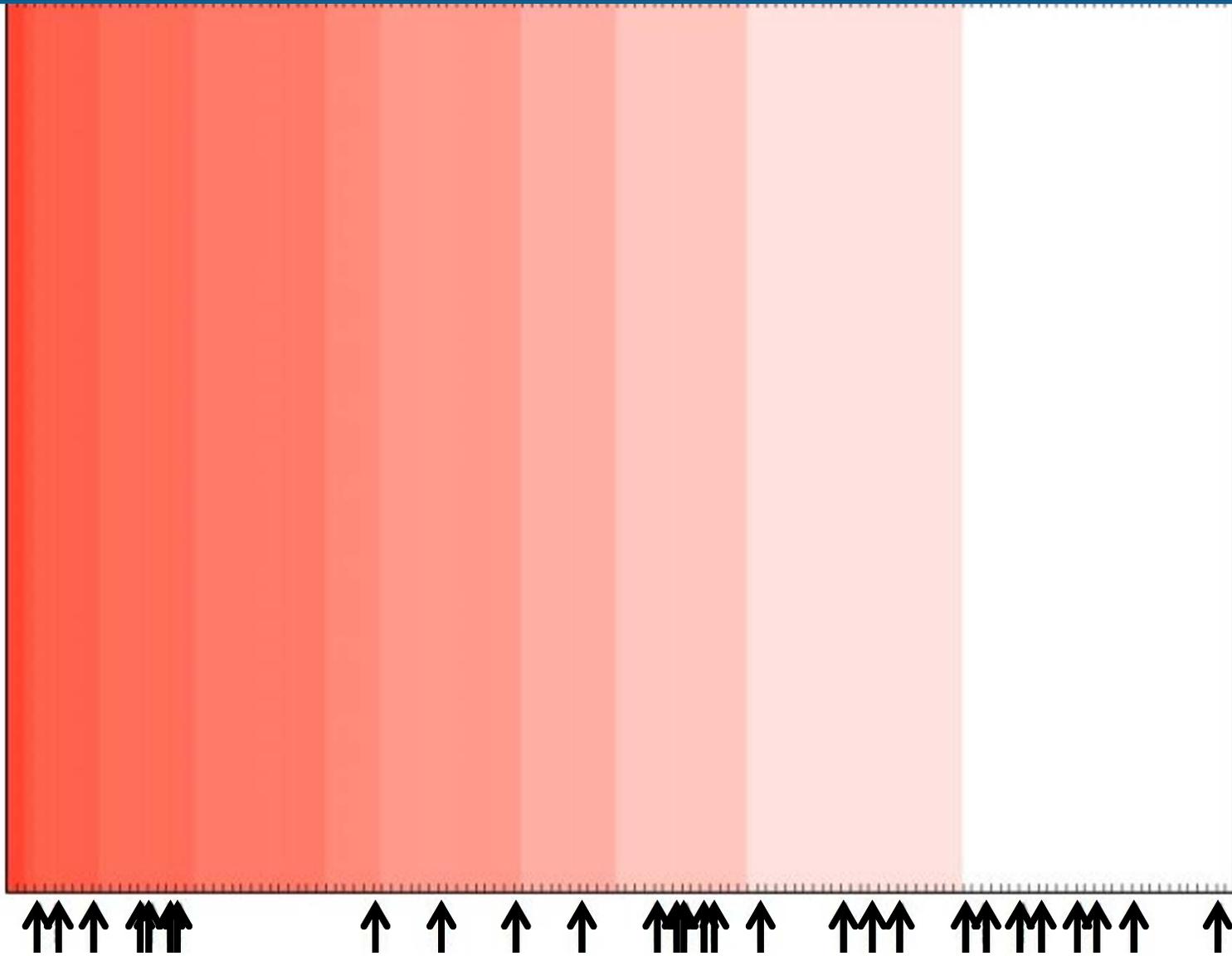
Finalists - CWE-120





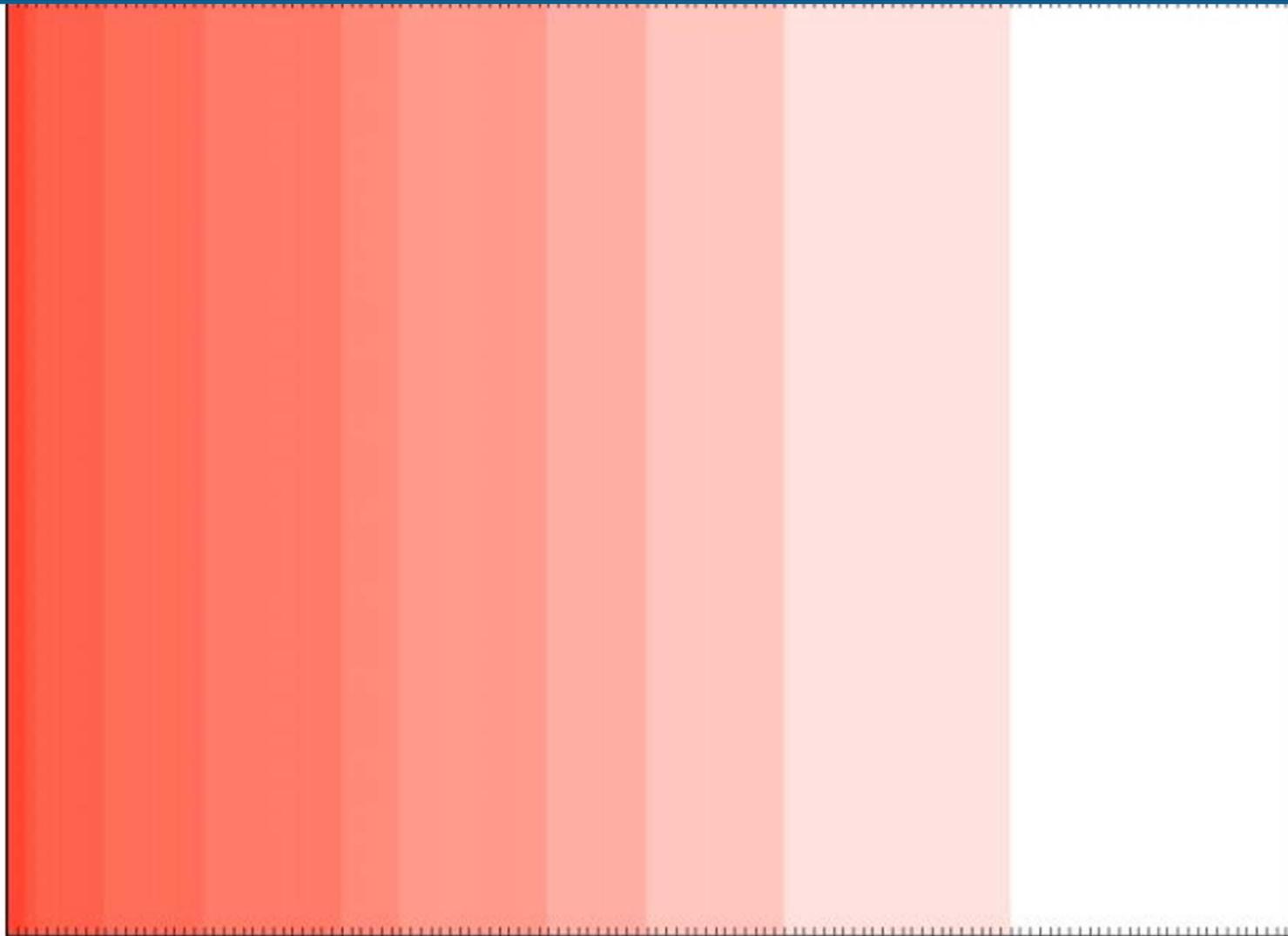


Finalists - CWE-122



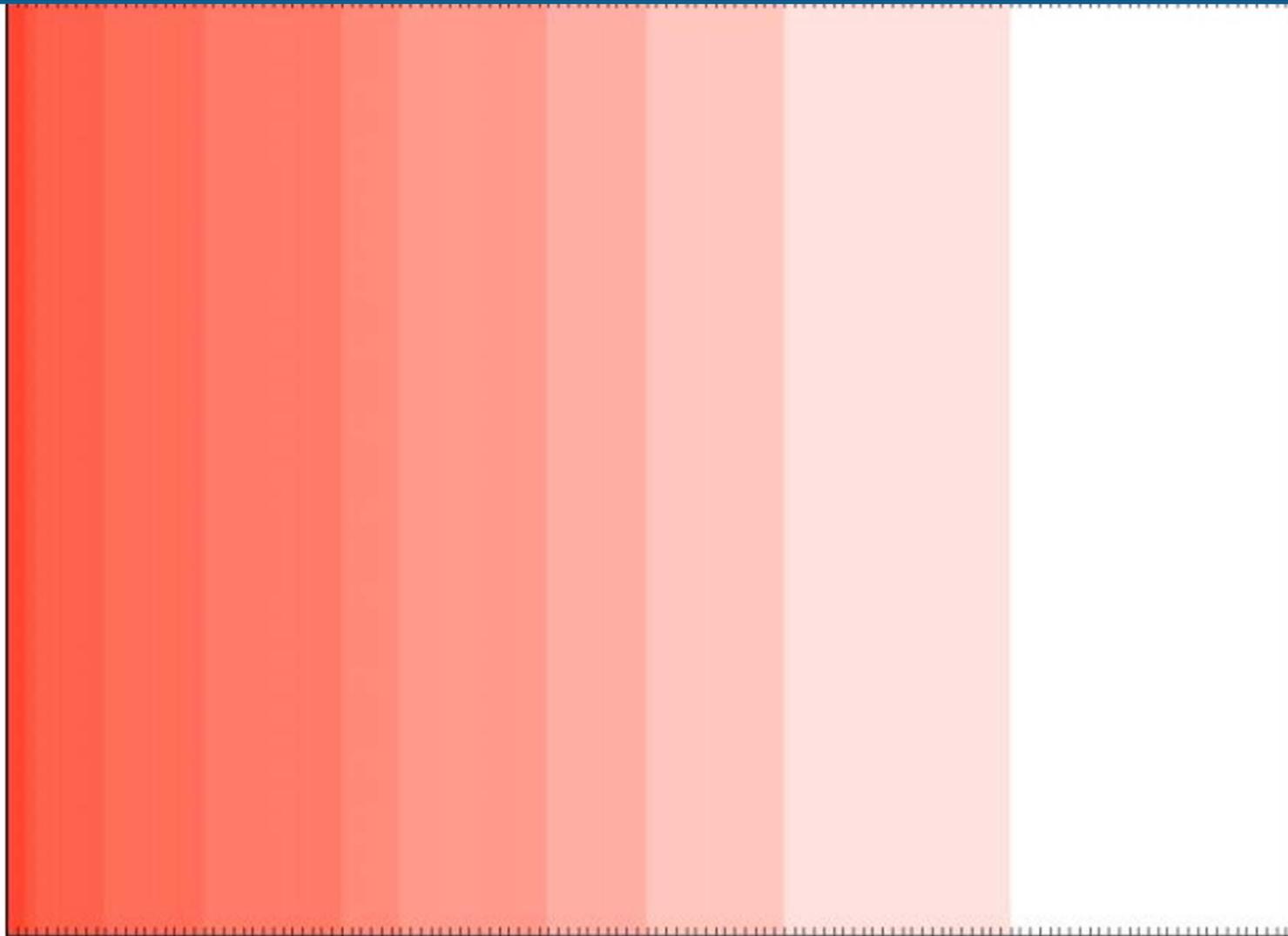


Finalists - CWE-125



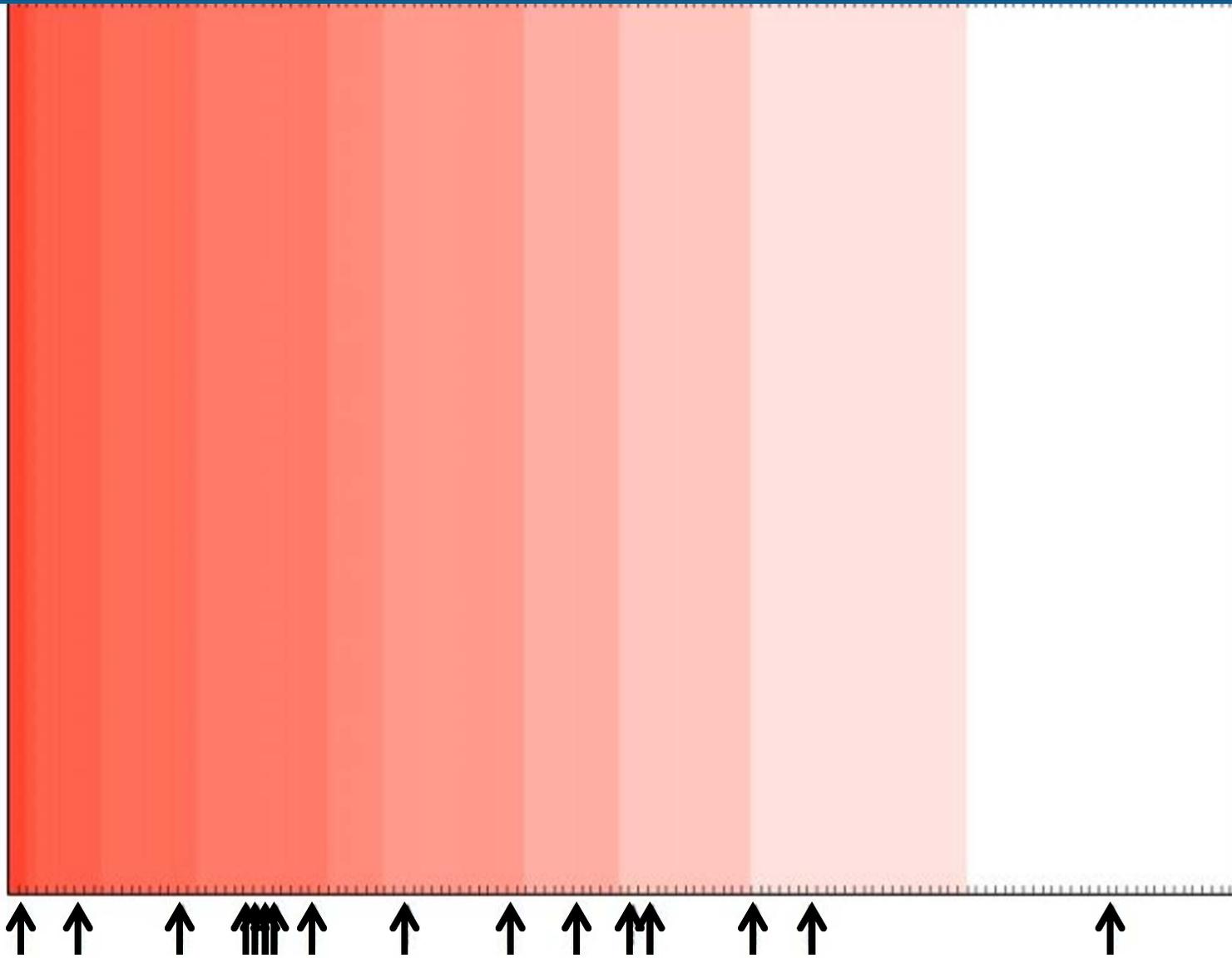


Finalists - CWE-787



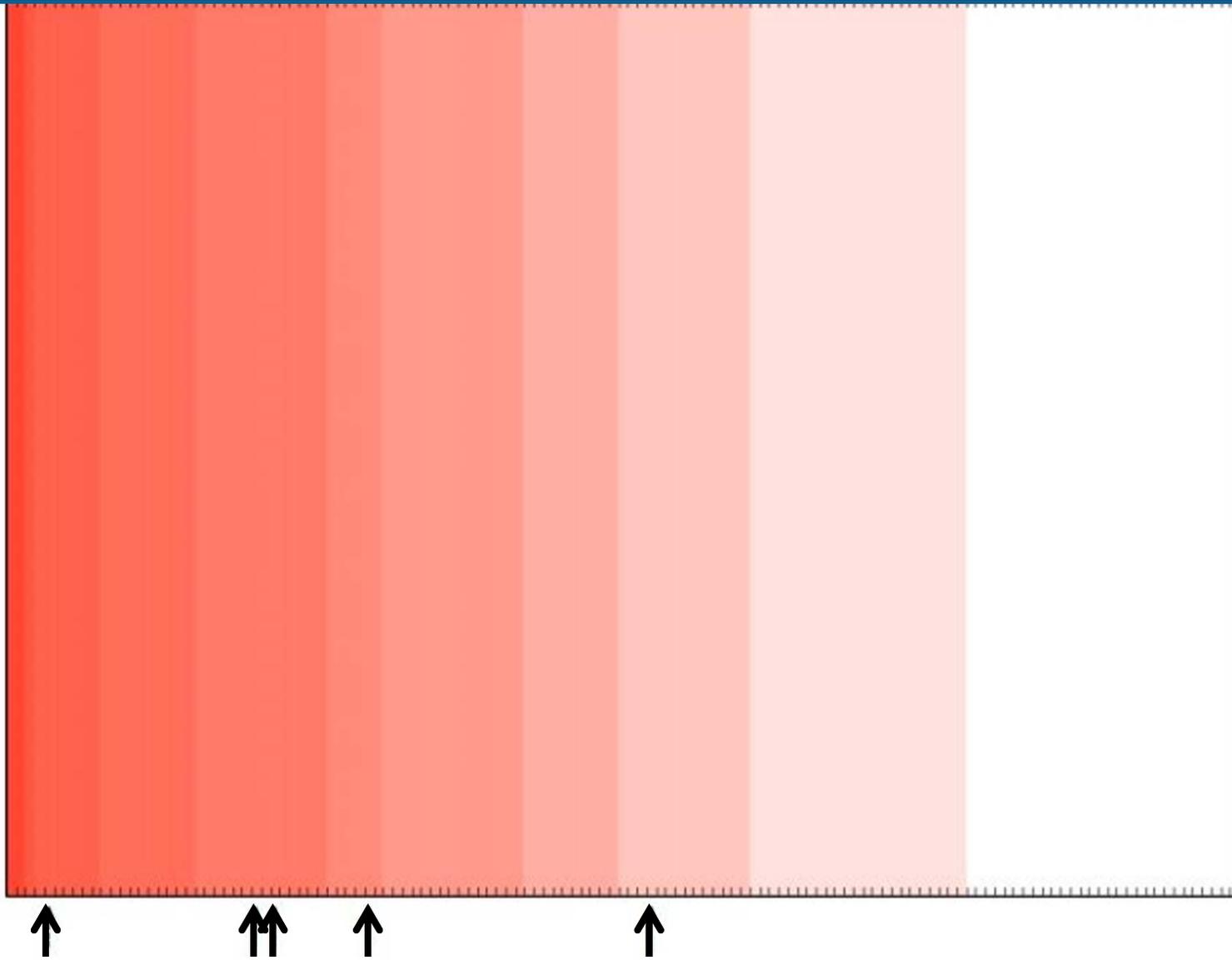


Finalists - CWE-476



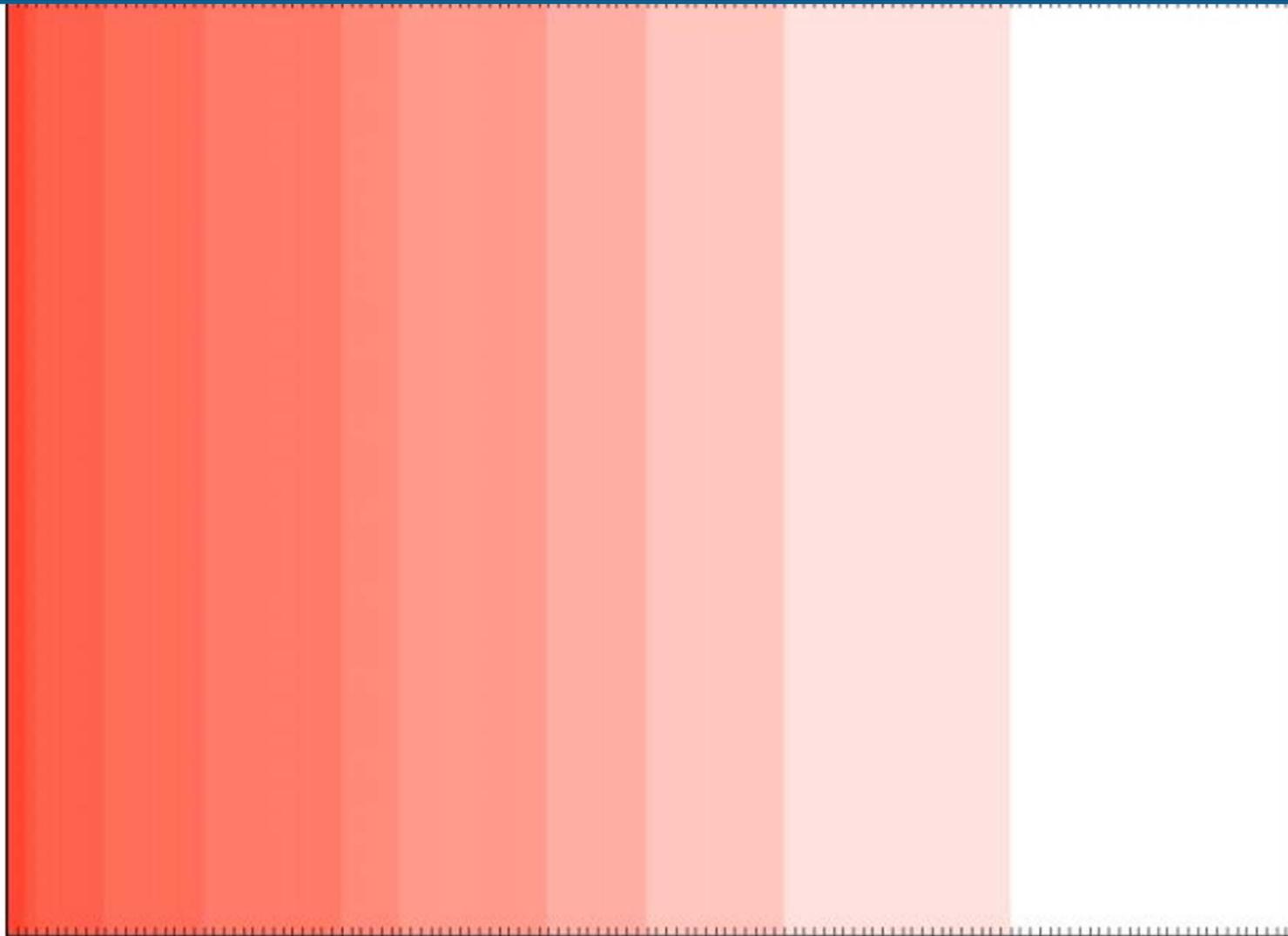


Finalists - CWE-824



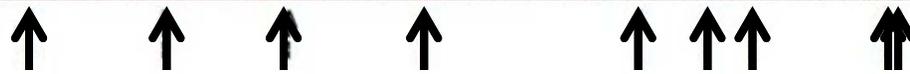
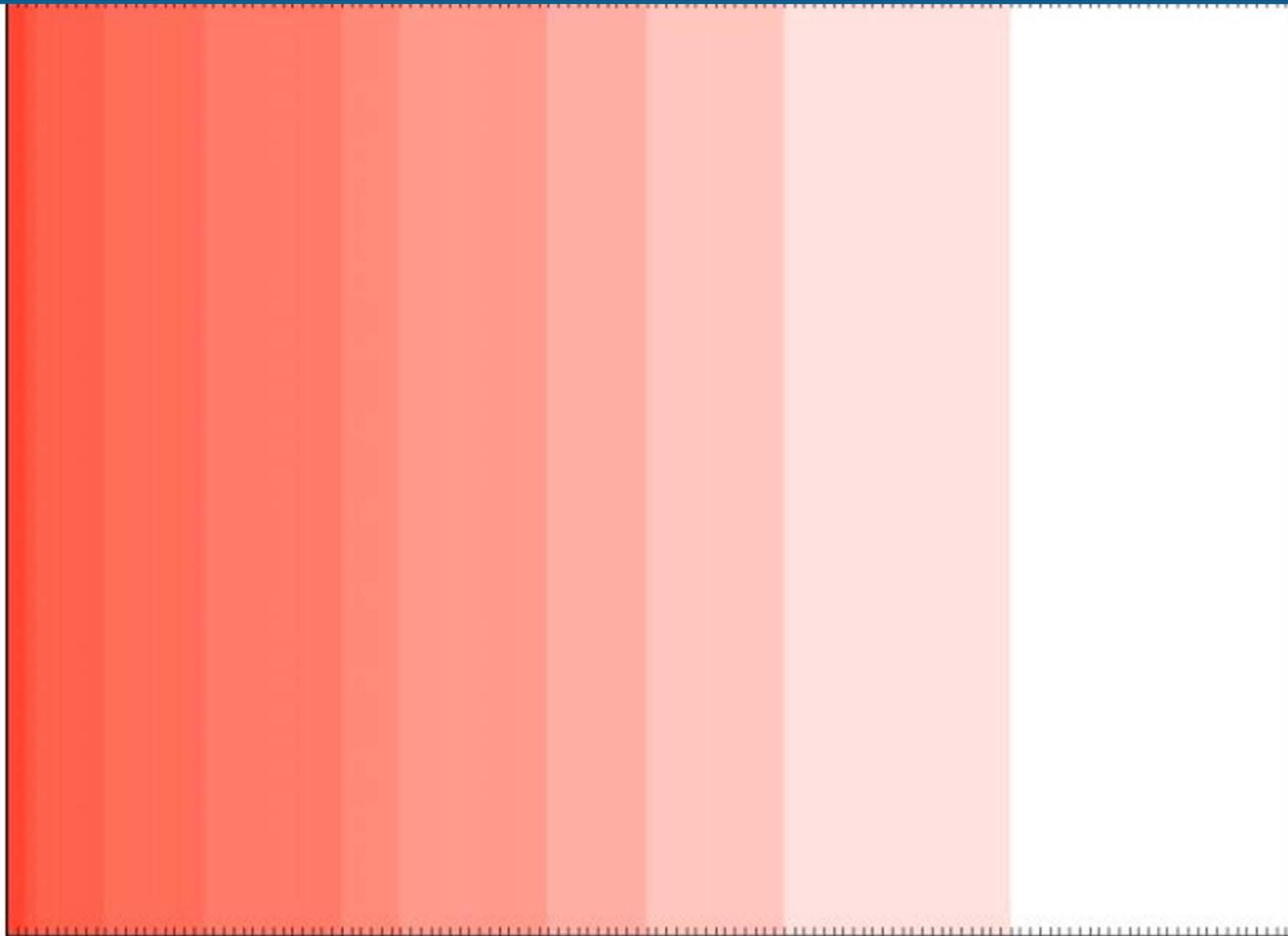


Finalists - CWE-843



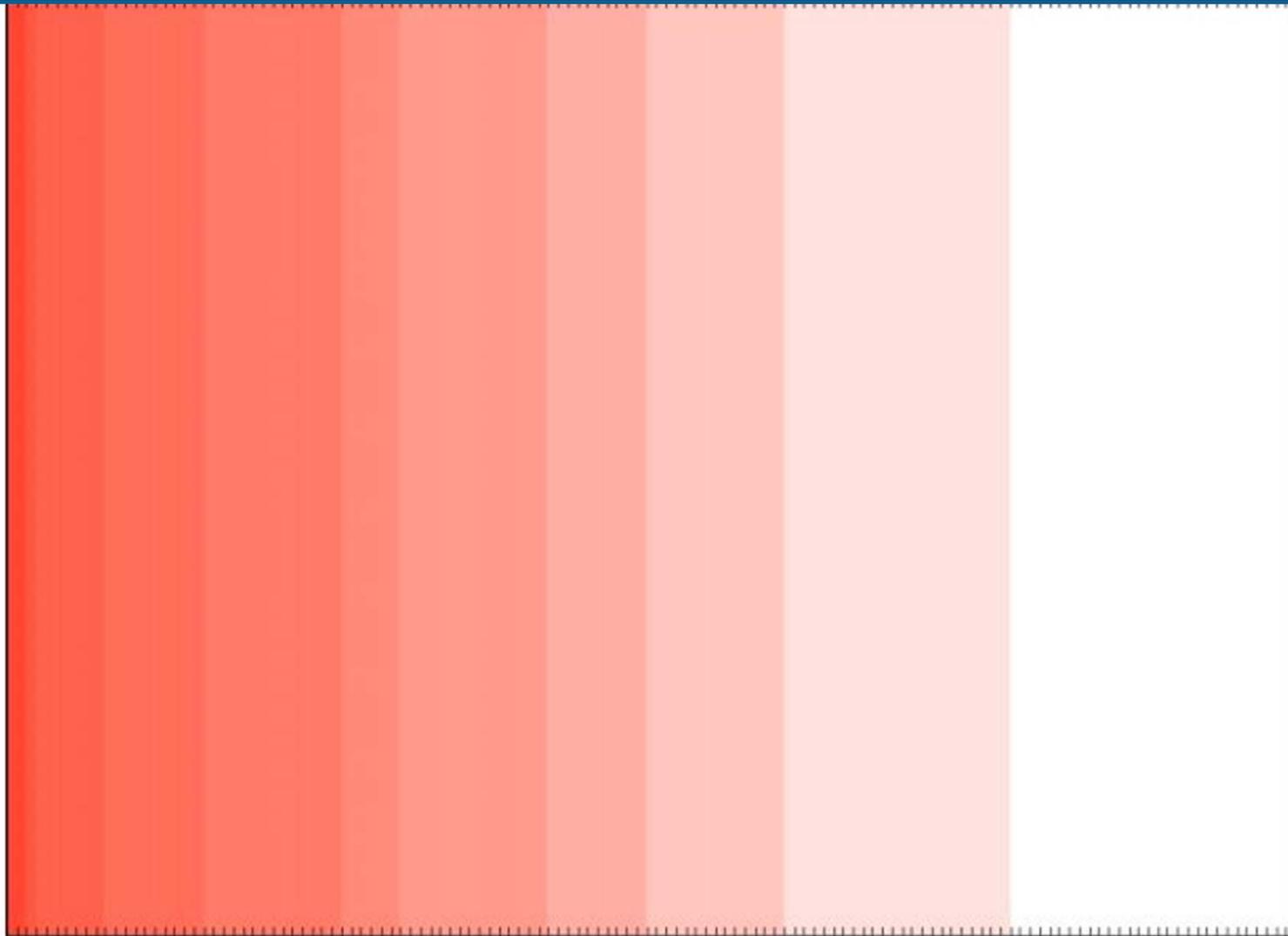


Finalists - CWE-20



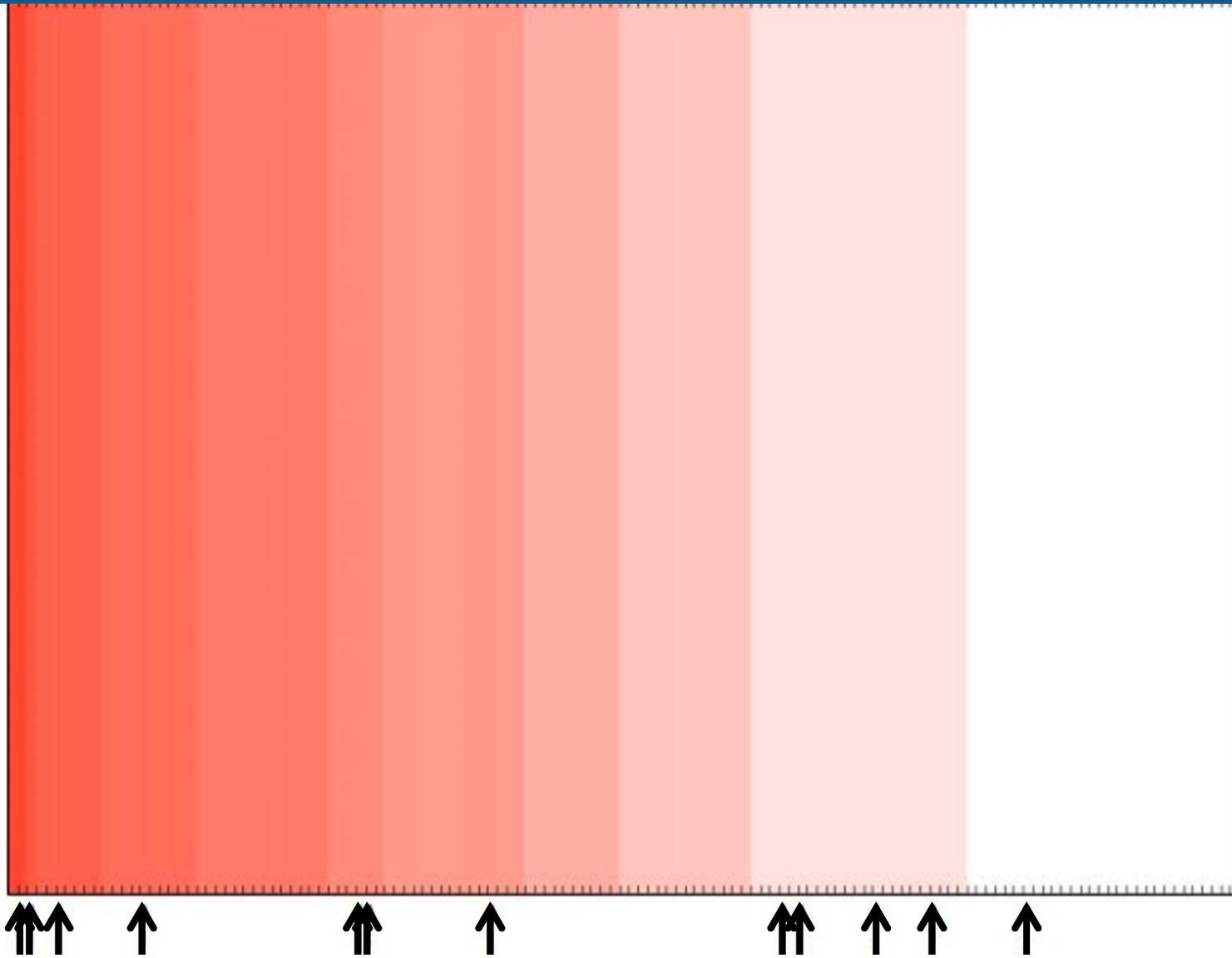


Finalists - CWE-129



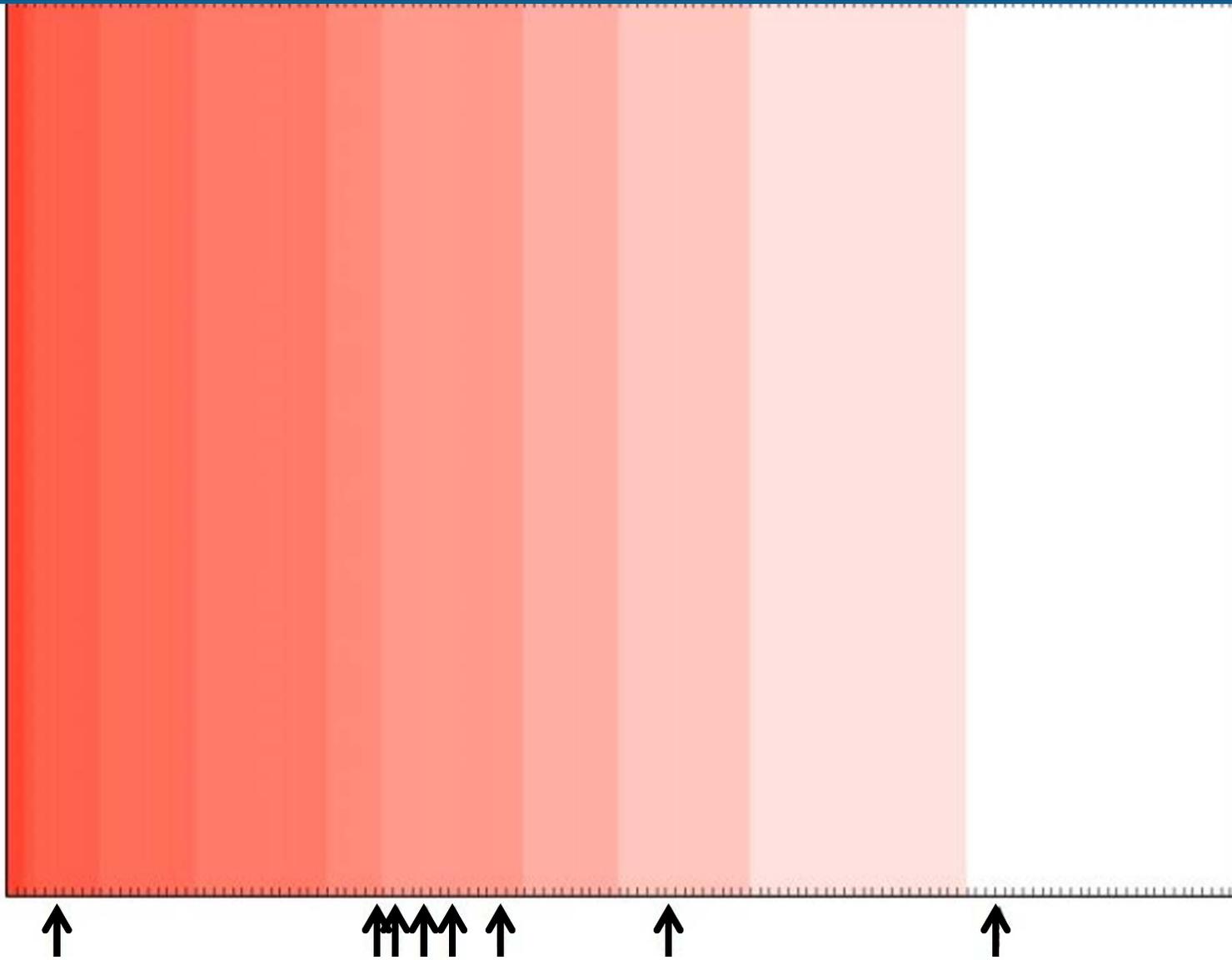


Finalists - CWE-190



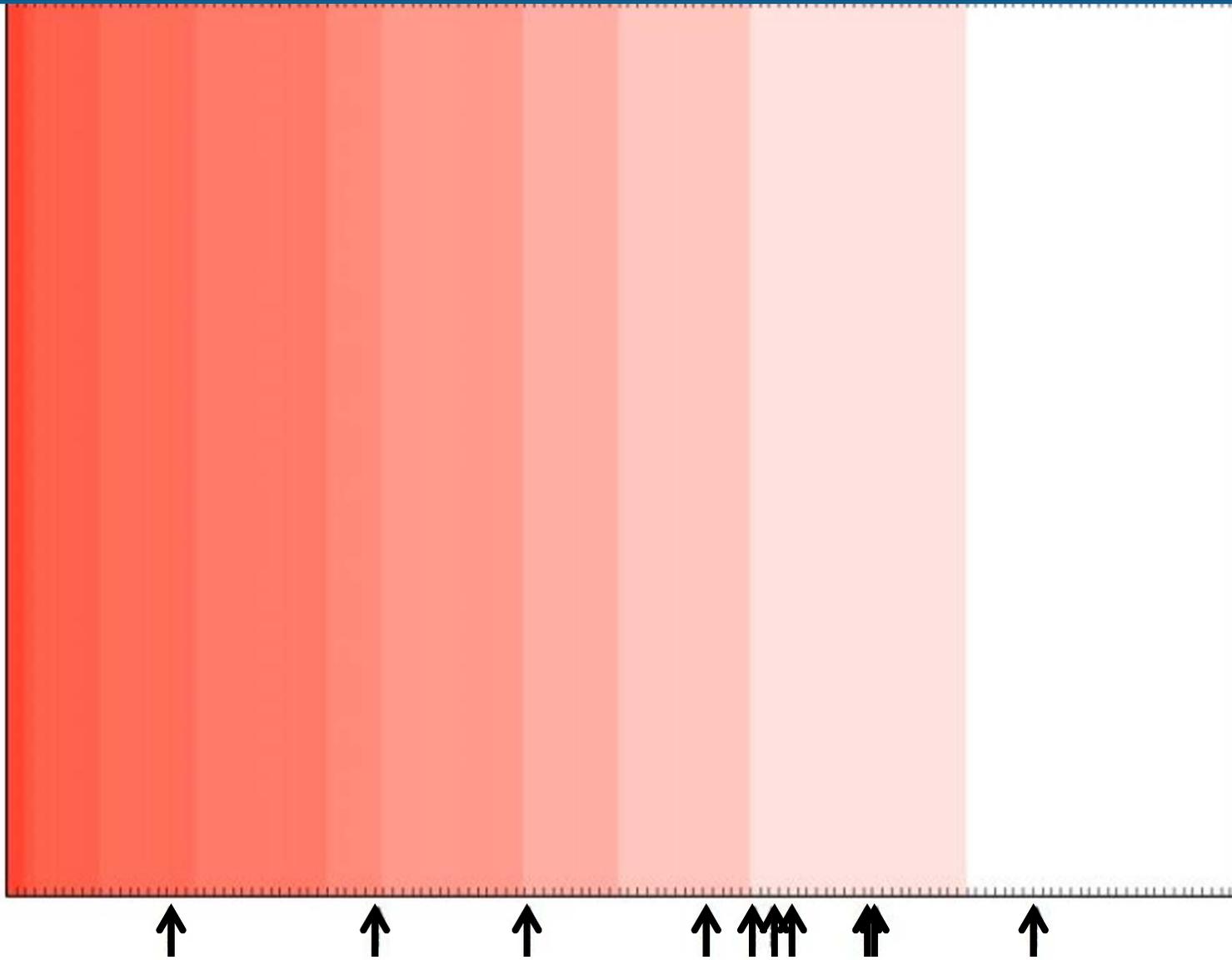


Finalists - CWE-416



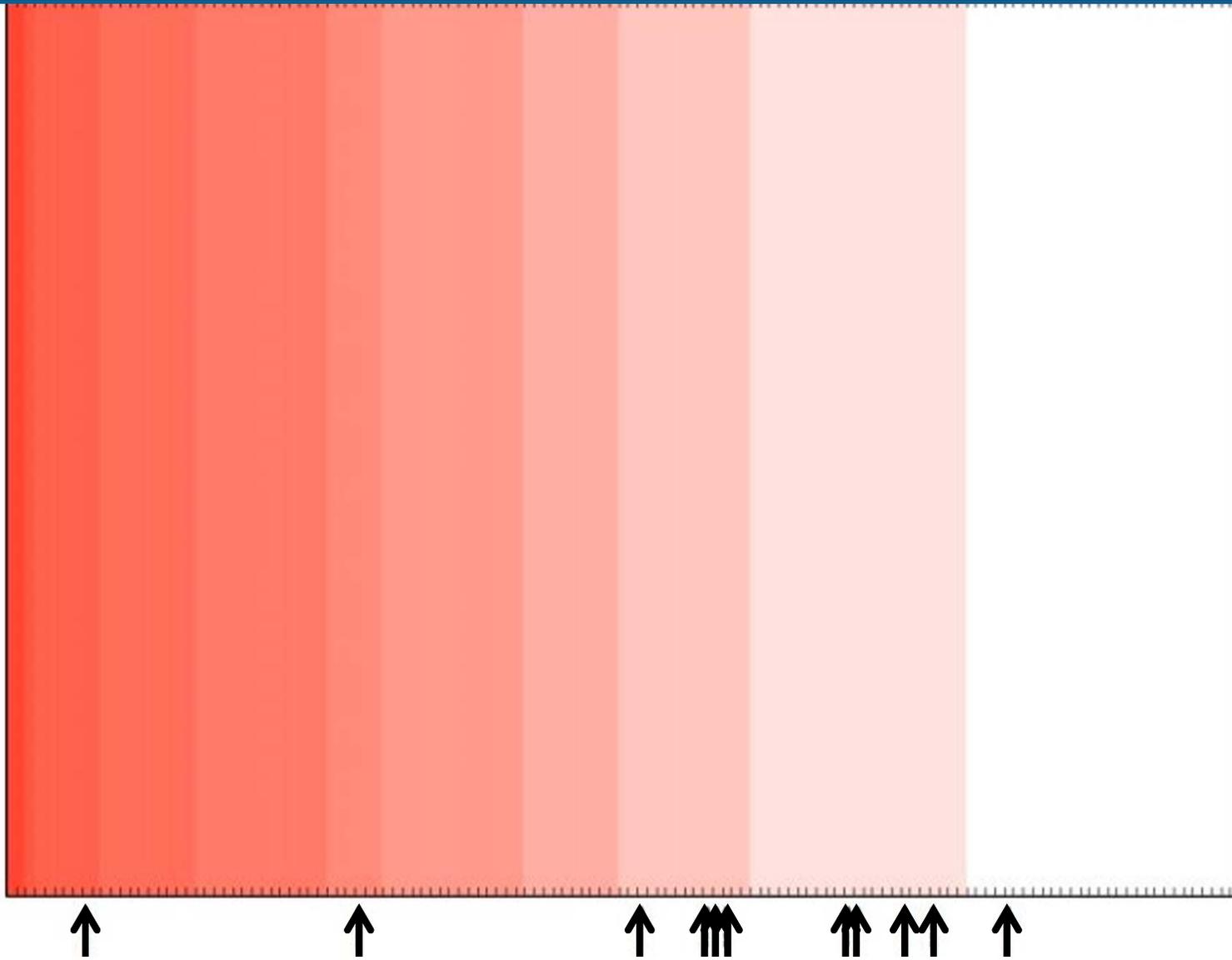


Finalists - CWE-193



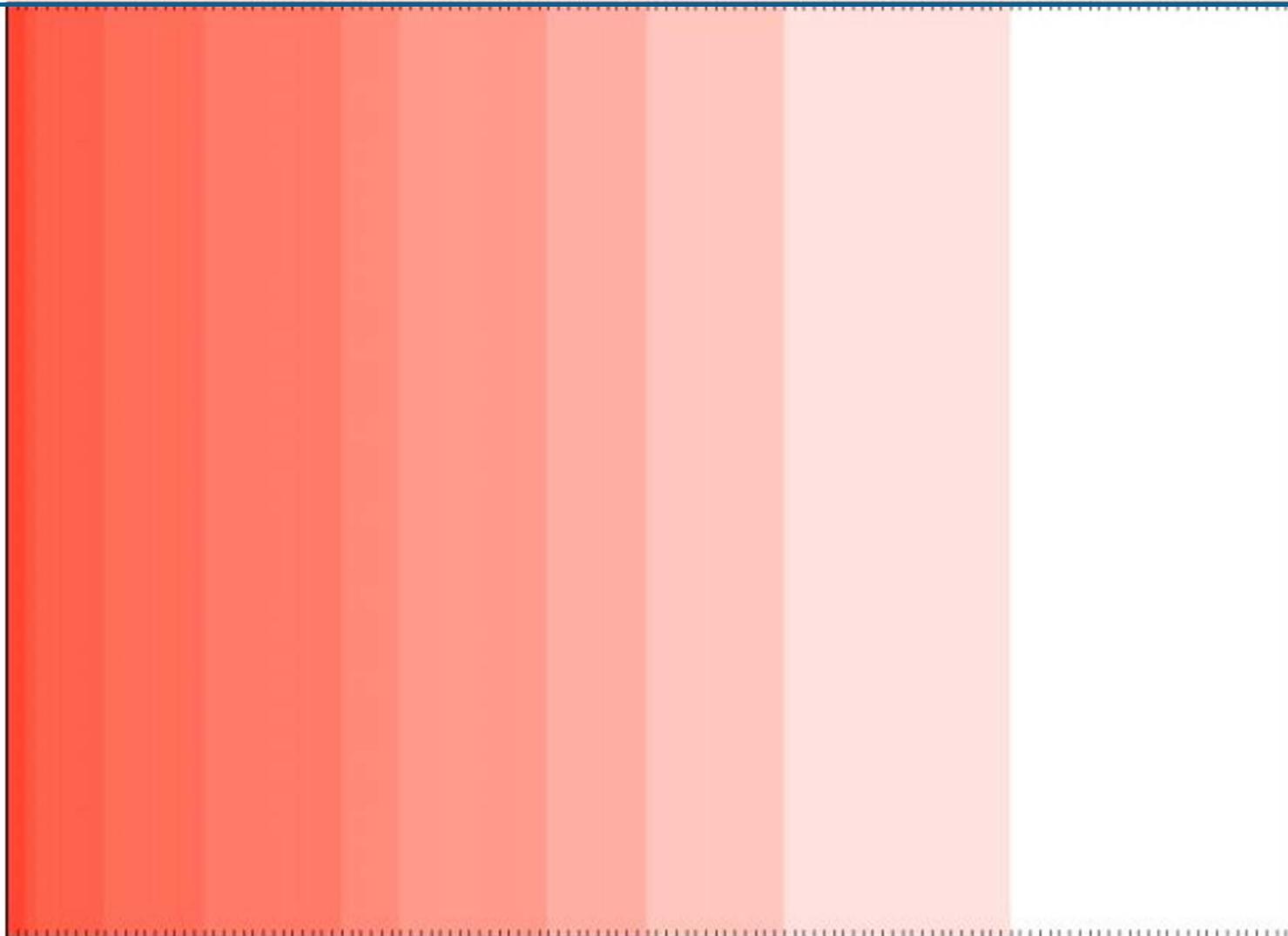


Finalists - CWE-131



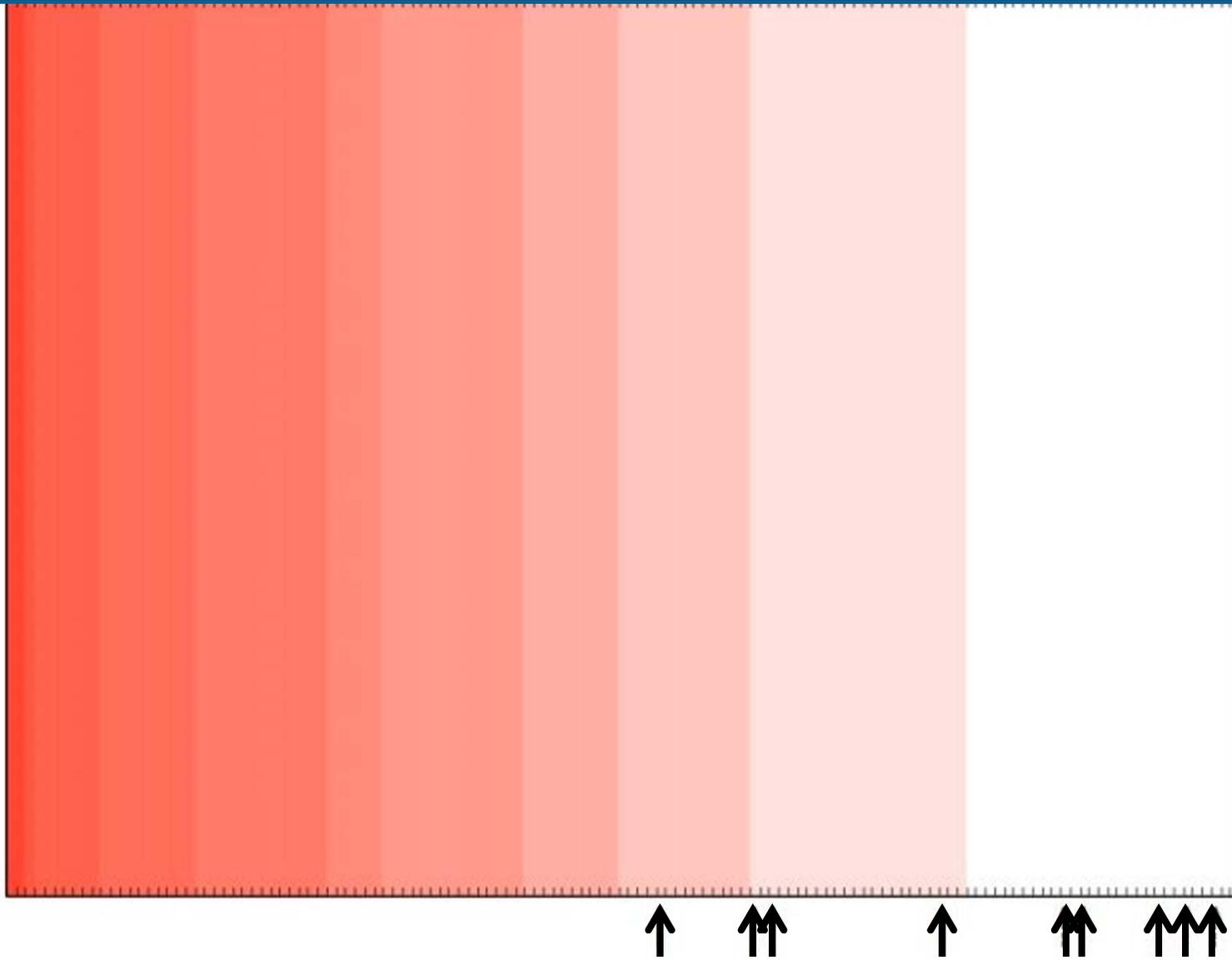


AllTeams - IPC





AllTeams - Tokenized



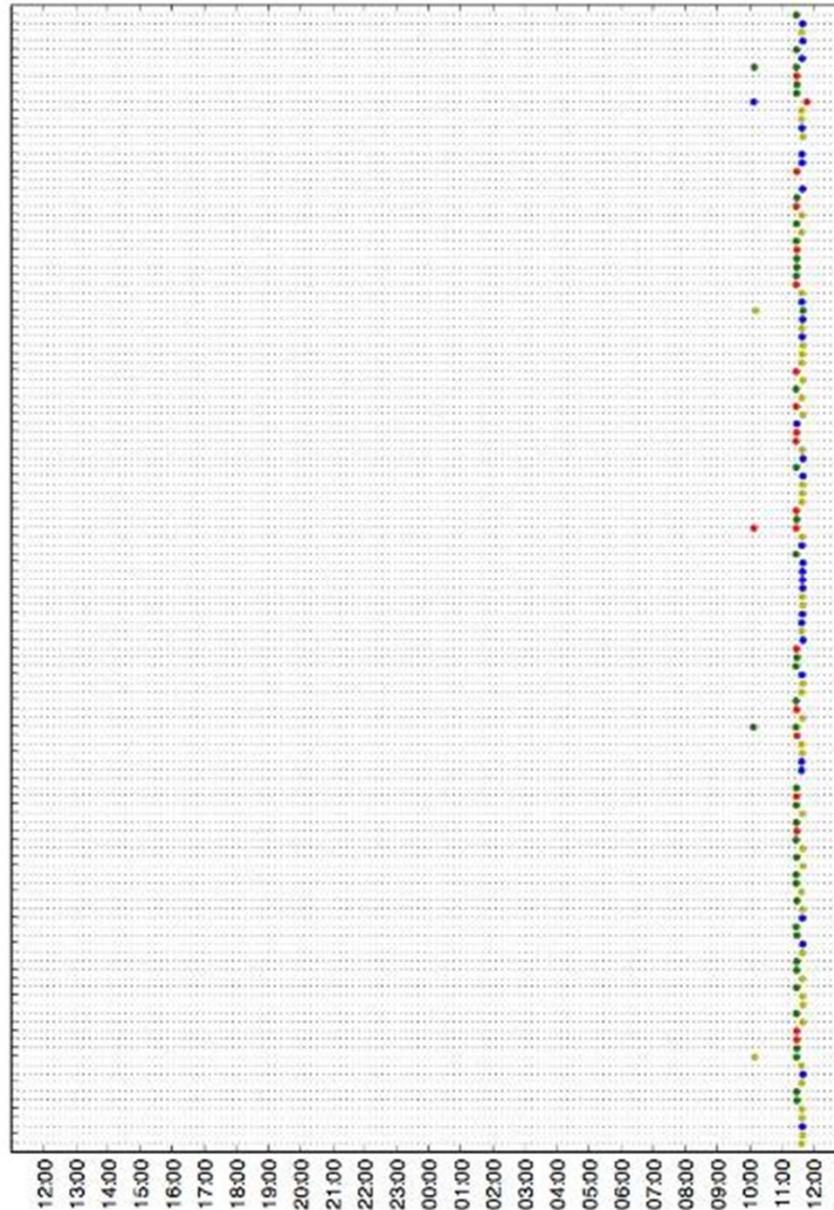


DARPA BAA 14-05



---

“...the speed and scale of automation”



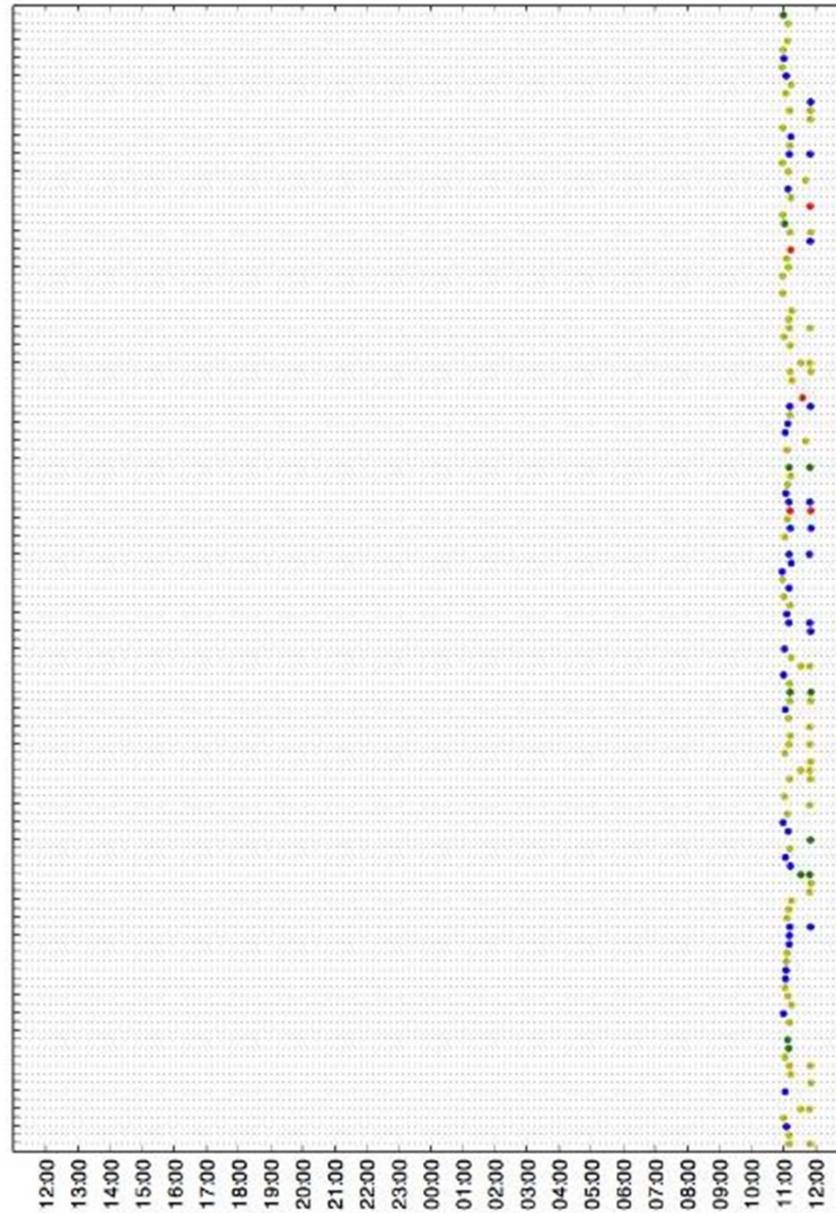
## Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



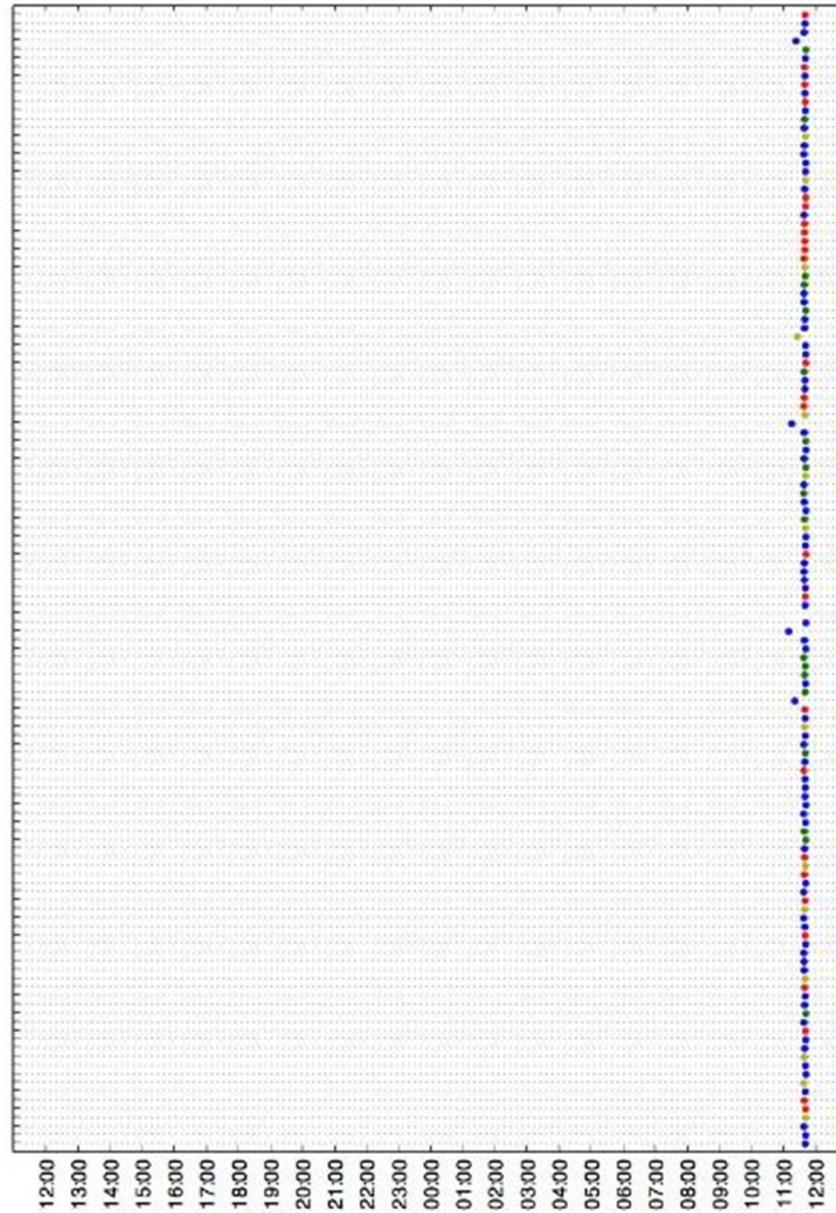
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



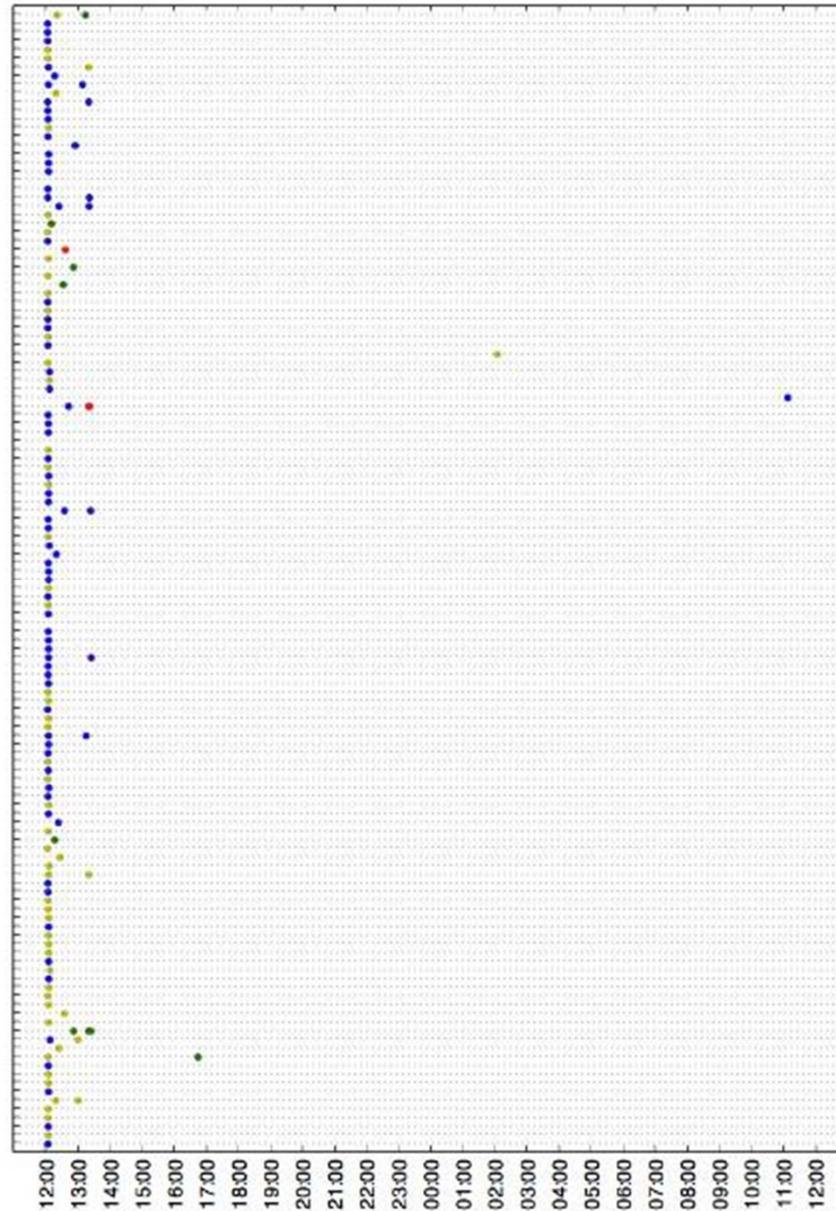
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



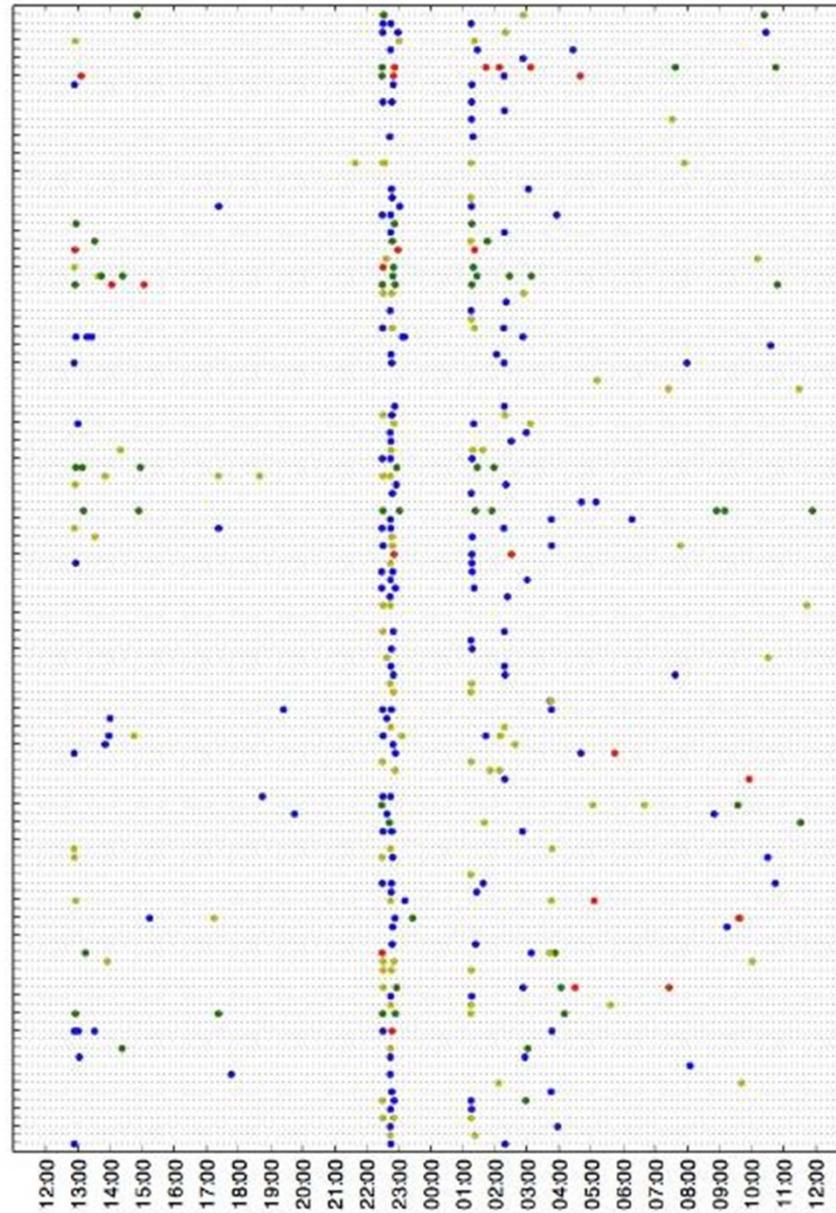
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



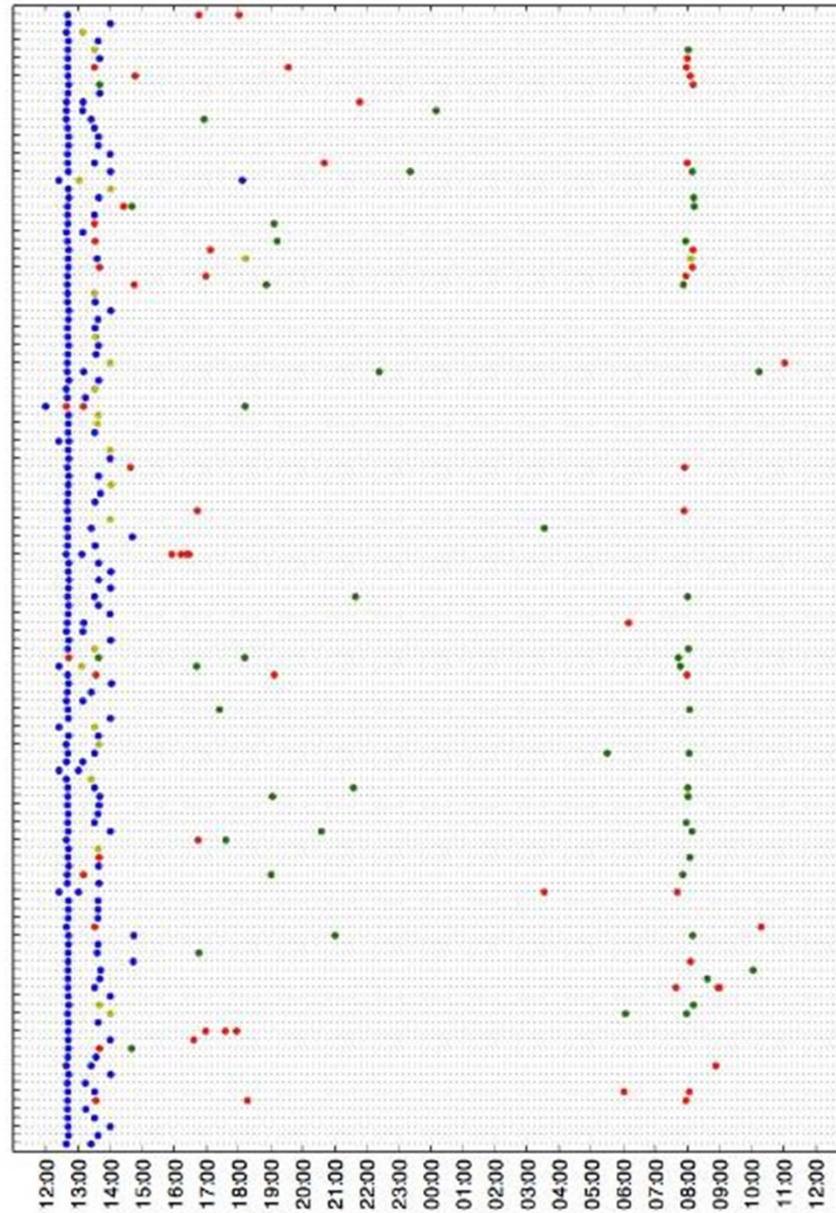
## Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



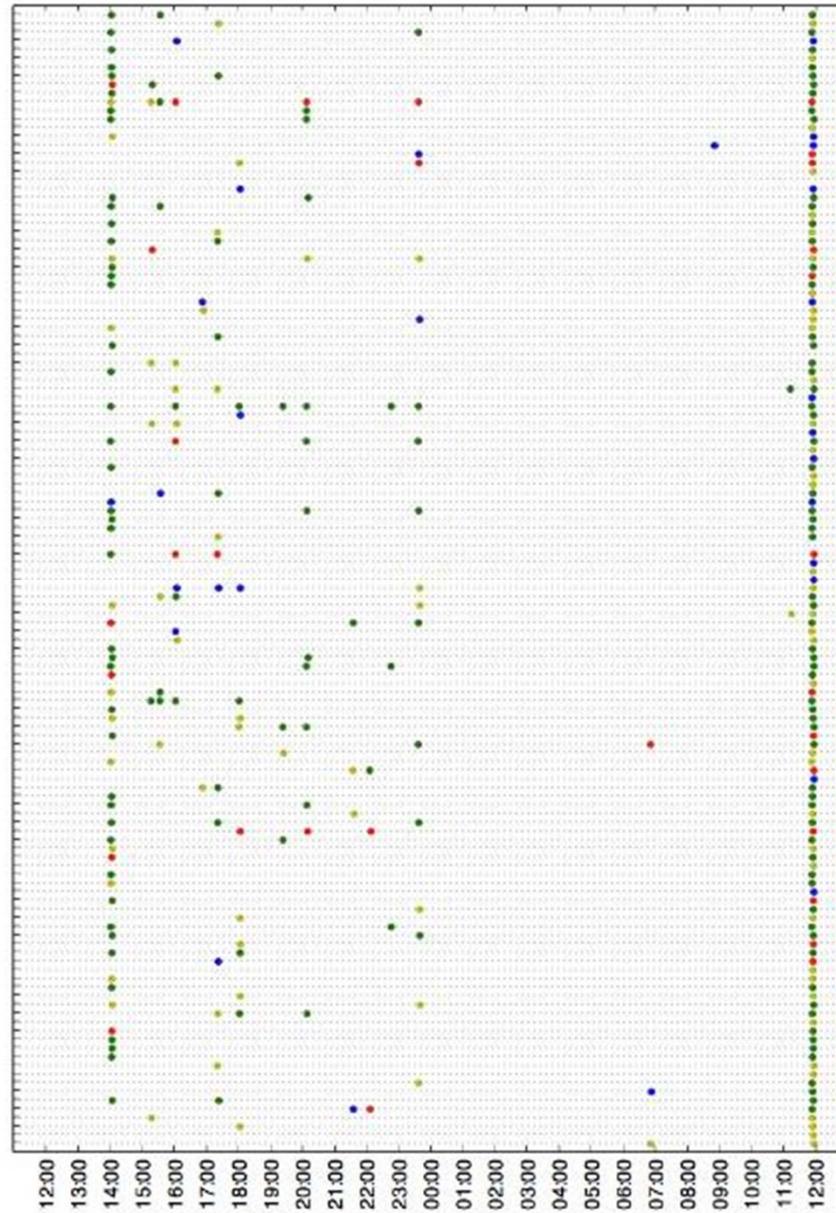
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



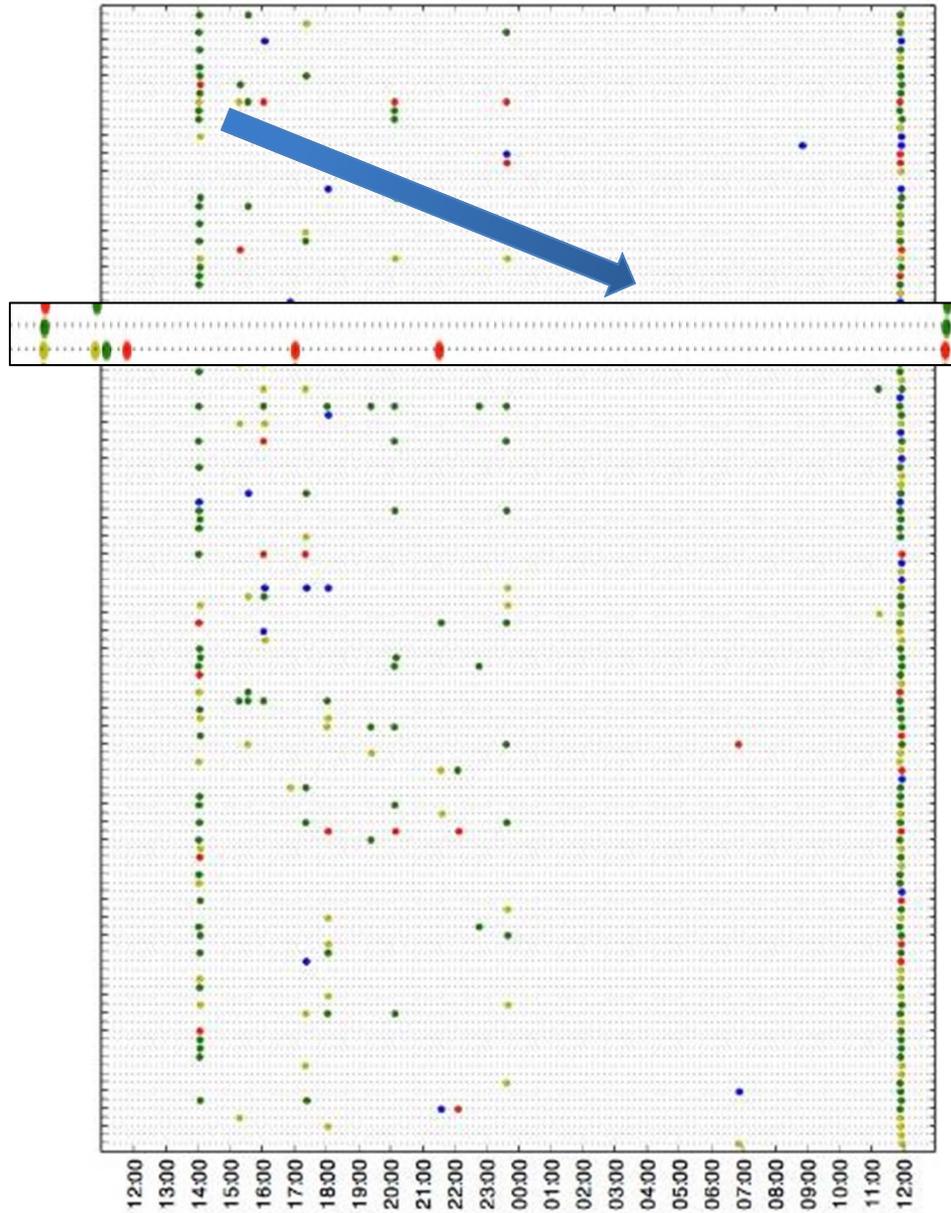
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



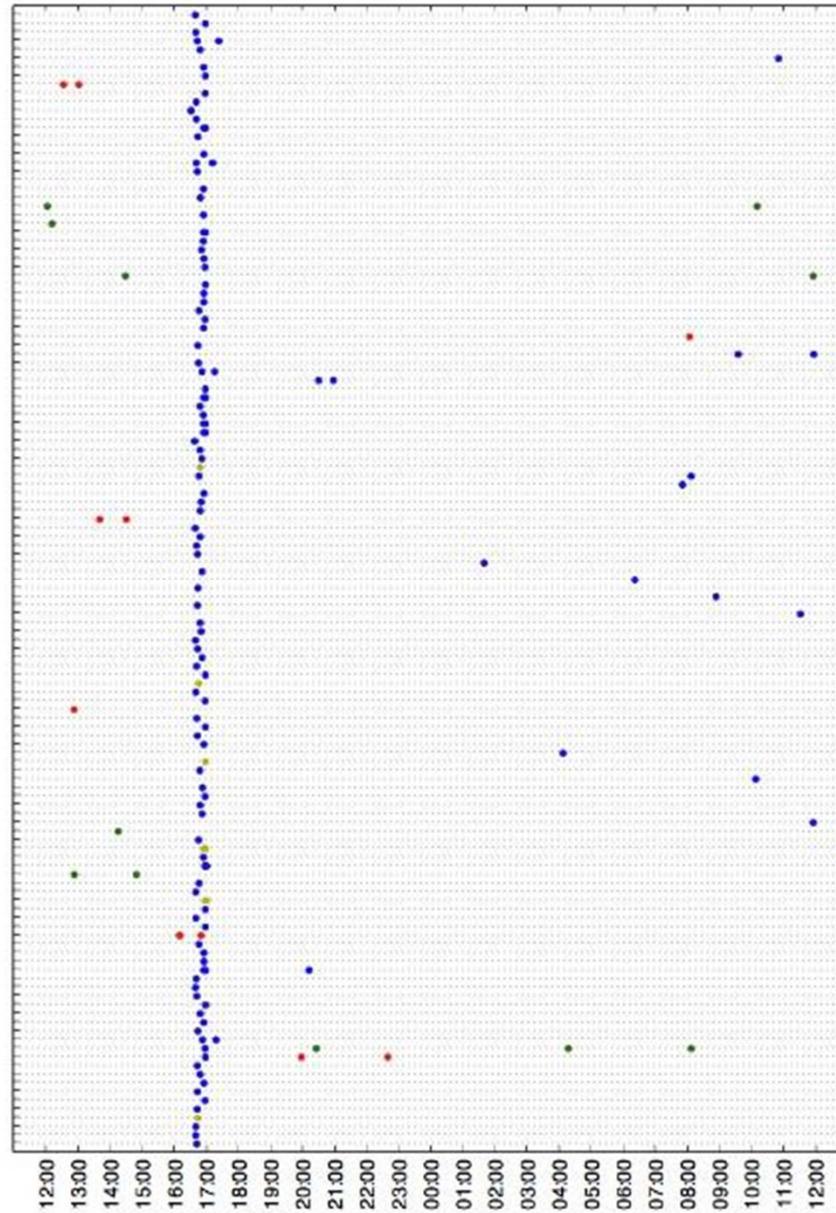
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



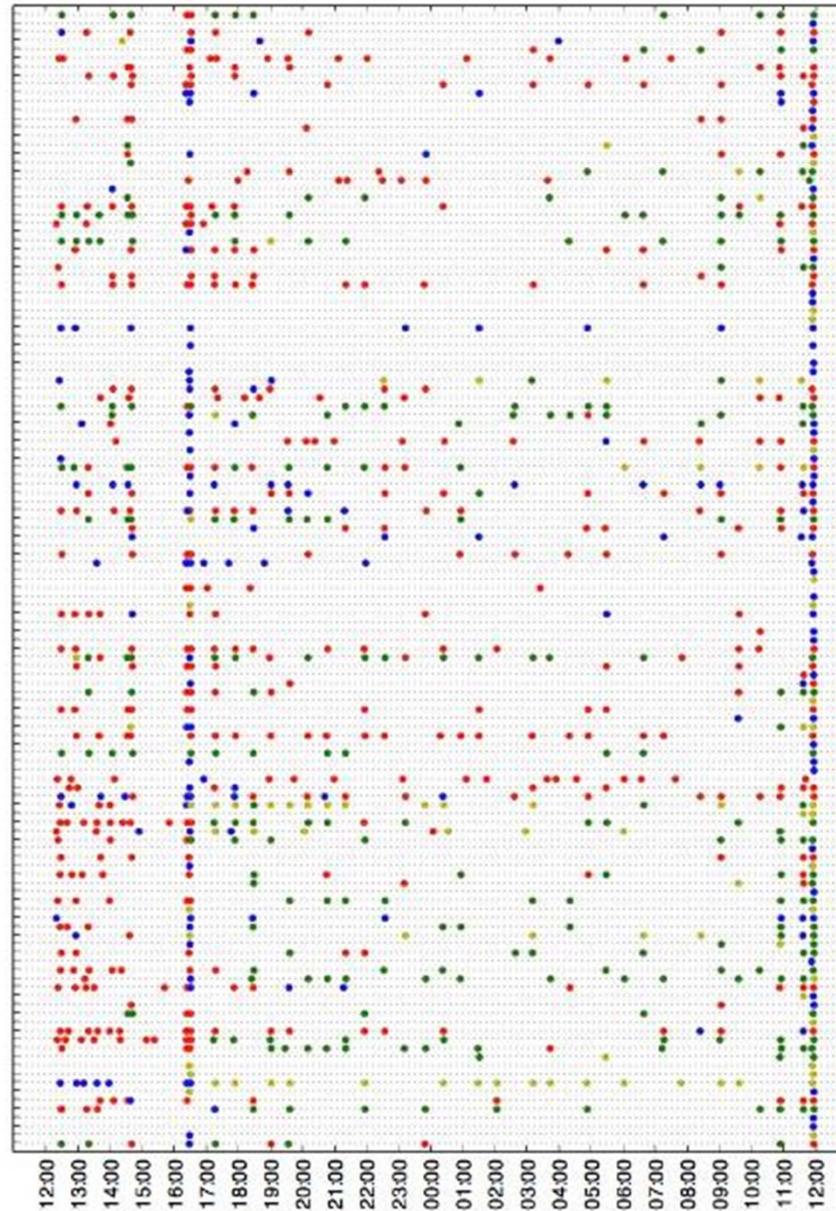
### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



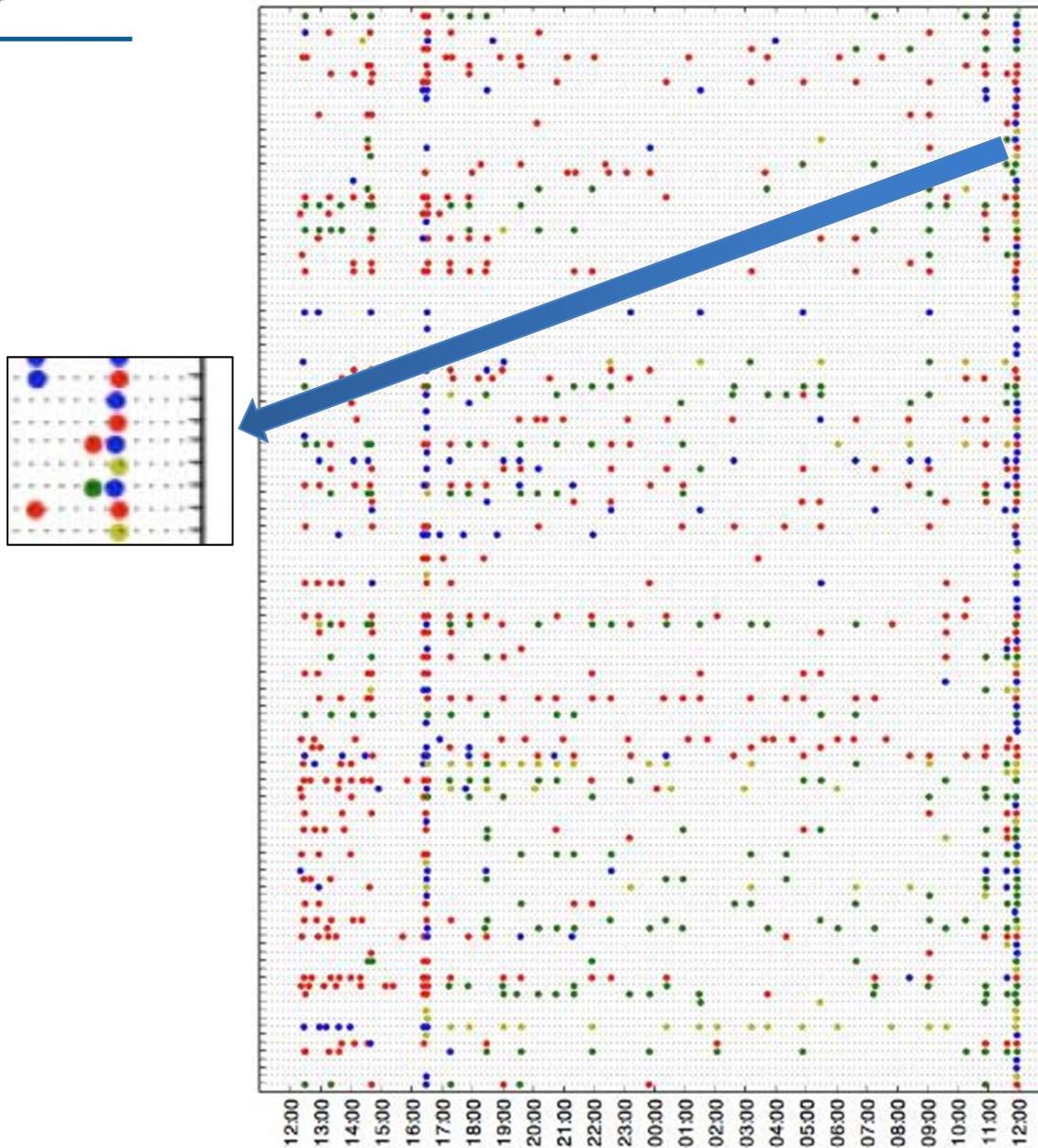
## Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission



### Legend

- Successful POV & Successful Patch
- Successful POV Only
- Successful Patch Only
- Unsuccessful Submission





## The DARPA Cyber Grand Challenge

- Seeks to make software safety the expert domain of machines
- Develops Fully Automated Opponents:
  - Reverse engineer unknown software
  - Locate weaknesses
  - Heal weaknesses without sacrificing
    - Functionality
    - Correctness
    - Performance
- Adopts the Capture The Flag competition format
  - Capture Flags: prove weakness
  - Defend Flags: remove weakness

The three funded-track teams heading to the CGC finals are:

**CodeJitsu (Berkeley, Calif.):** A team affiliated with the University of California, Berkeley

**ForAllSecure (Pittsburgh, Pa.):** A startup founded by a team of computer security researchers from Carnegie Mellon University

**TECHx (Charlottesville, Va.):** Software analysis experts from GrammaTech, Inc., a developer of software assurance tools and advanced cybersecurity solutions, and the University of Virginia

The four open-track teams are:

**CSDS (Moscow, Idaho):** A professor and post-doctoral researcher from the University of Idaho

**DeepRed (Arlington, Va.):** A team of engineers from the Raytheon Company

**disekt (Athens, Ga.):** Four people, working out of a technology incubator, who participate in CTF competitions around the world

**Shellphish (Santa Barbara, Calif.):** A group of computer science graduate students at the University of California, Santa Barbara



# CGC Qualification Event



[https://cgc.darpa.mil/CGC\\_Master\\_Schedule\\_15\\_Apr\\_15.pdf](https://cgc.darpa.mil/CGC_Master_Schedule_15_Apr_15.pdf)



# CGC Areas of Excellence



	Areas of Excellence (AoE)	CGC Qualification Event (CQE)	CGC Final Event (CFE)
1	<b>Autonomous Analysis:</b> The automated comprehension of computer software (e.g., CBs) provided through a Competition Framework.	✓	✓
2	<b>Autonomous Patching:</b> The automatic patching of security flaws in CBs provided through a Competition Framework.	✓	✓
3	<b>Autonomous Vulnerability Scanning:</b> The ability to construct input which when transmitted over a network provides proof of the existence of flaws in CBs operated by competitors. These inputs shall be regarded as Proofs of Vulnerability.	✓	✓
4	<b>Autonomous Service Resiliency:</b> The ability to maintain the availability and intended function of CBs provided through a Competition Framework.	✓	✓
5	<b>Autonomous Network Defense:</b> The ability to discover and mitigate security flaws in CBs from the vantage point of a network security device.		✓



## Major differences



### CQE

POV = Crash with security implications

Competitors compete in isolation

All binaries are compiled using CGC tools – well controlled

Greater ability to debug (A111)

### CFE

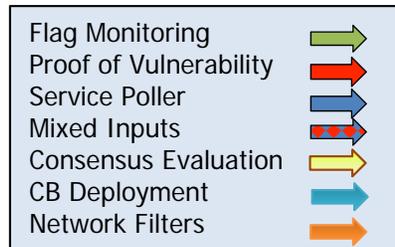
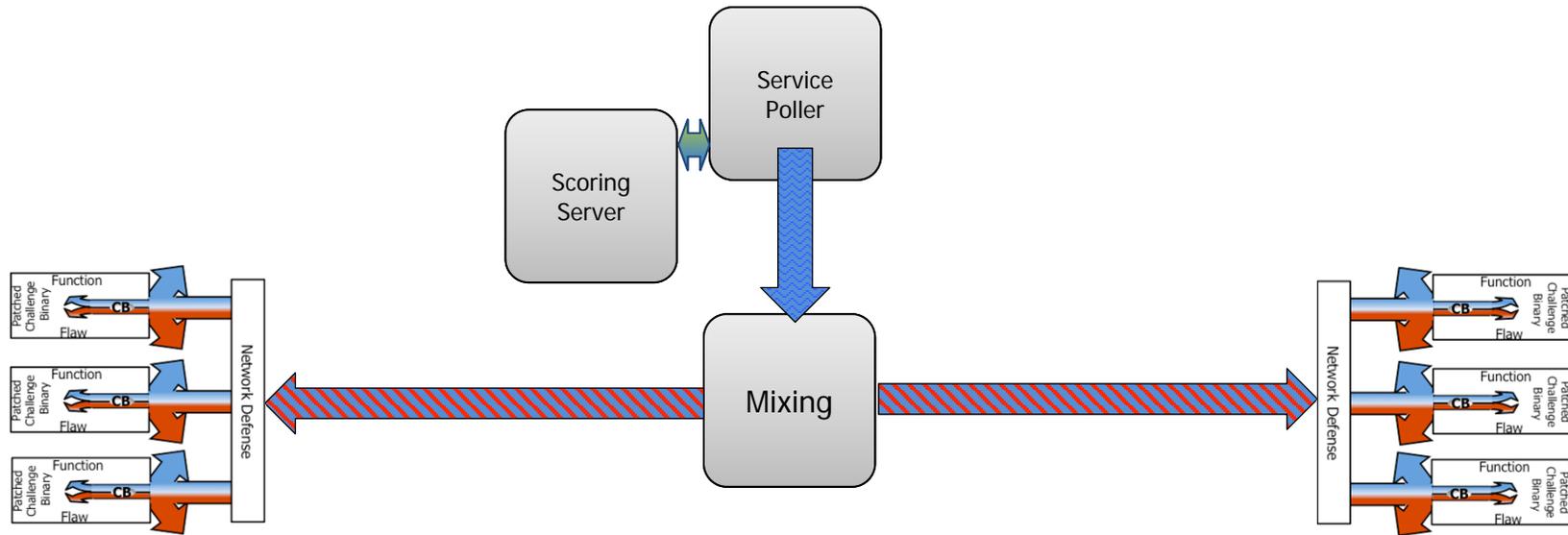
POV =

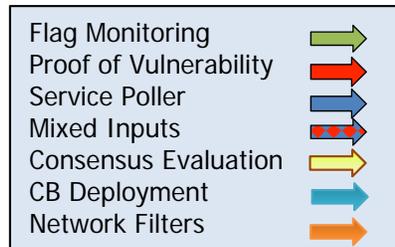
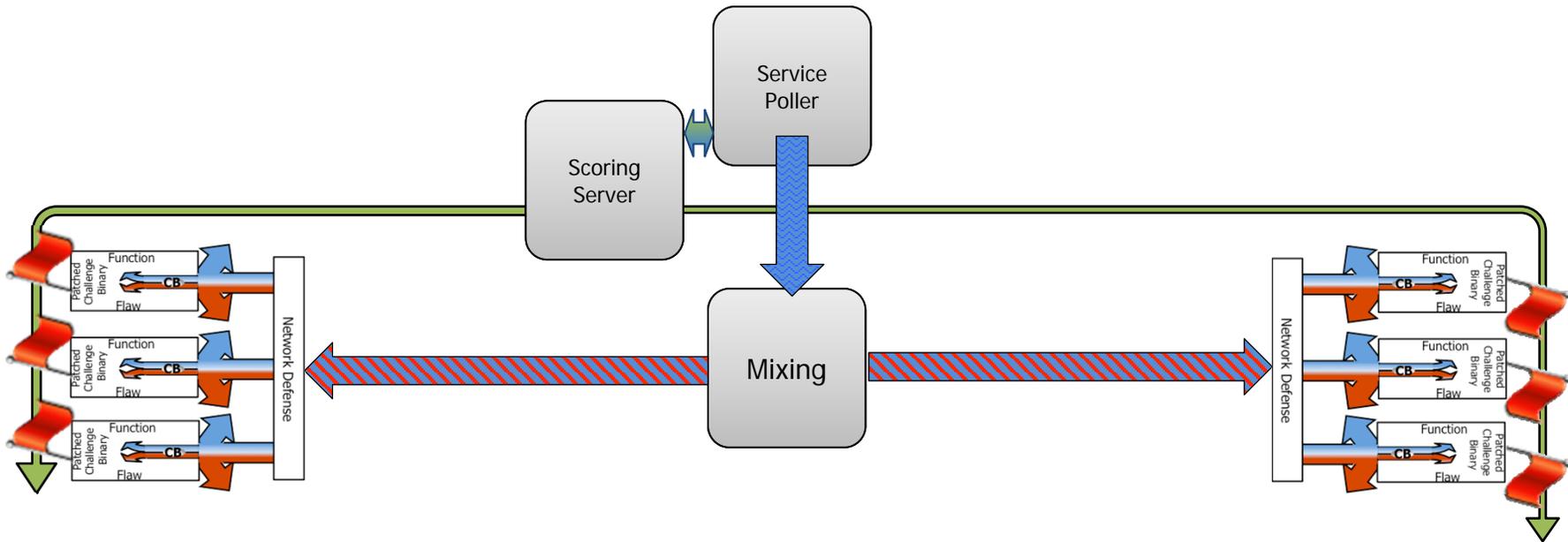
- Registers
- Memory

Head-to-head competition

Competitor-generated replacements

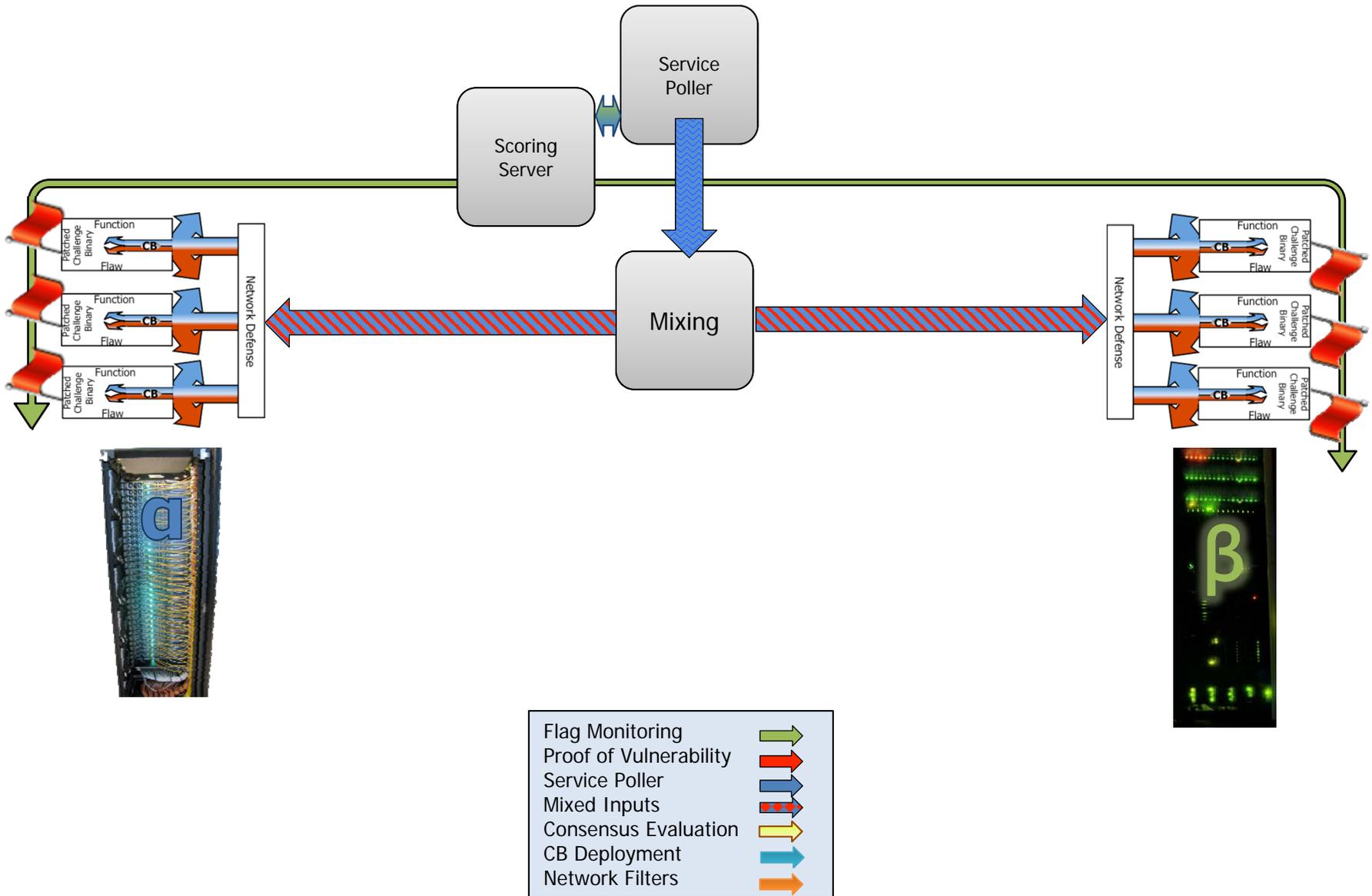
Competitors have the ability to program a network security appliance





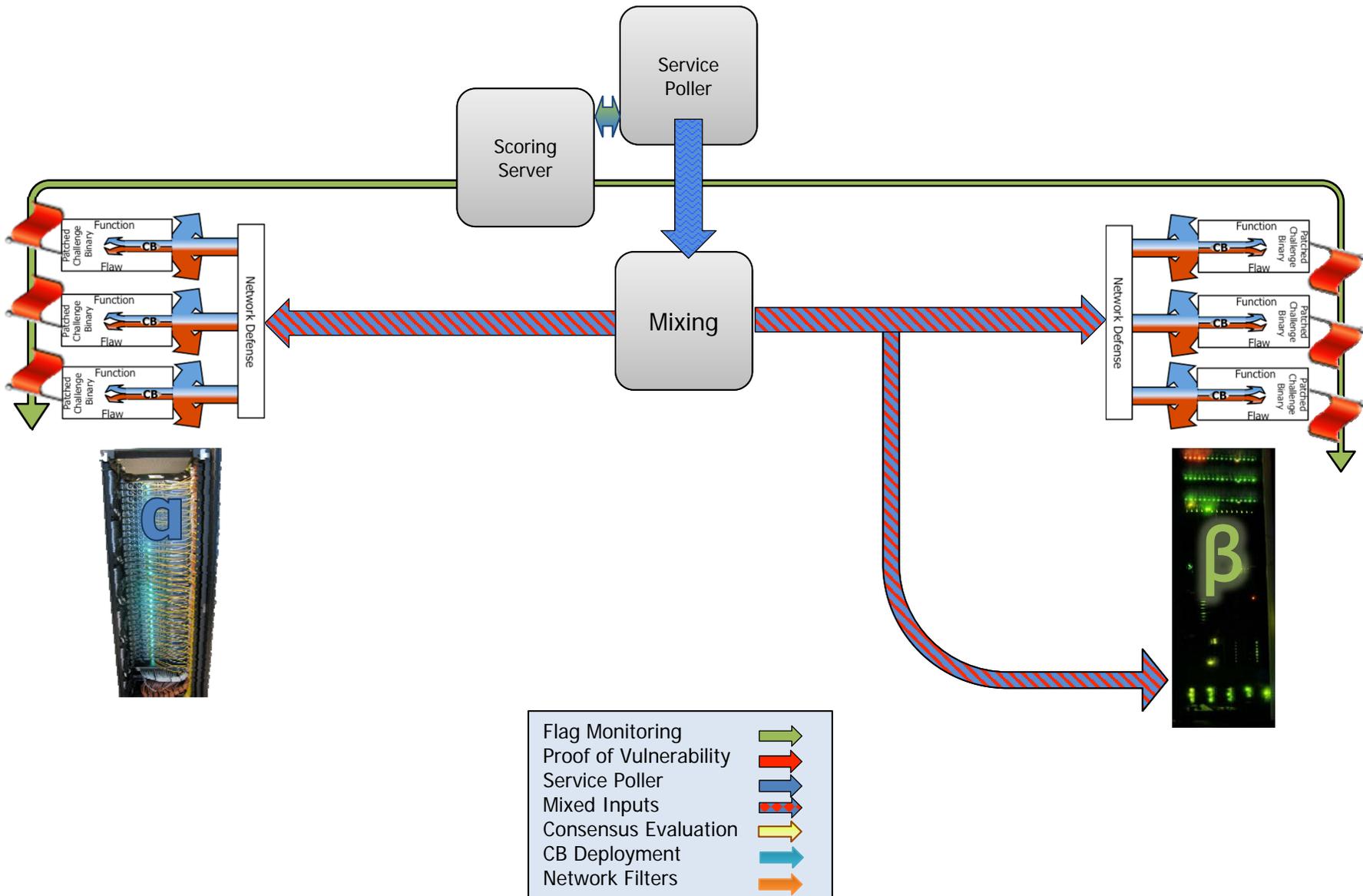


# Consensus Evaluation





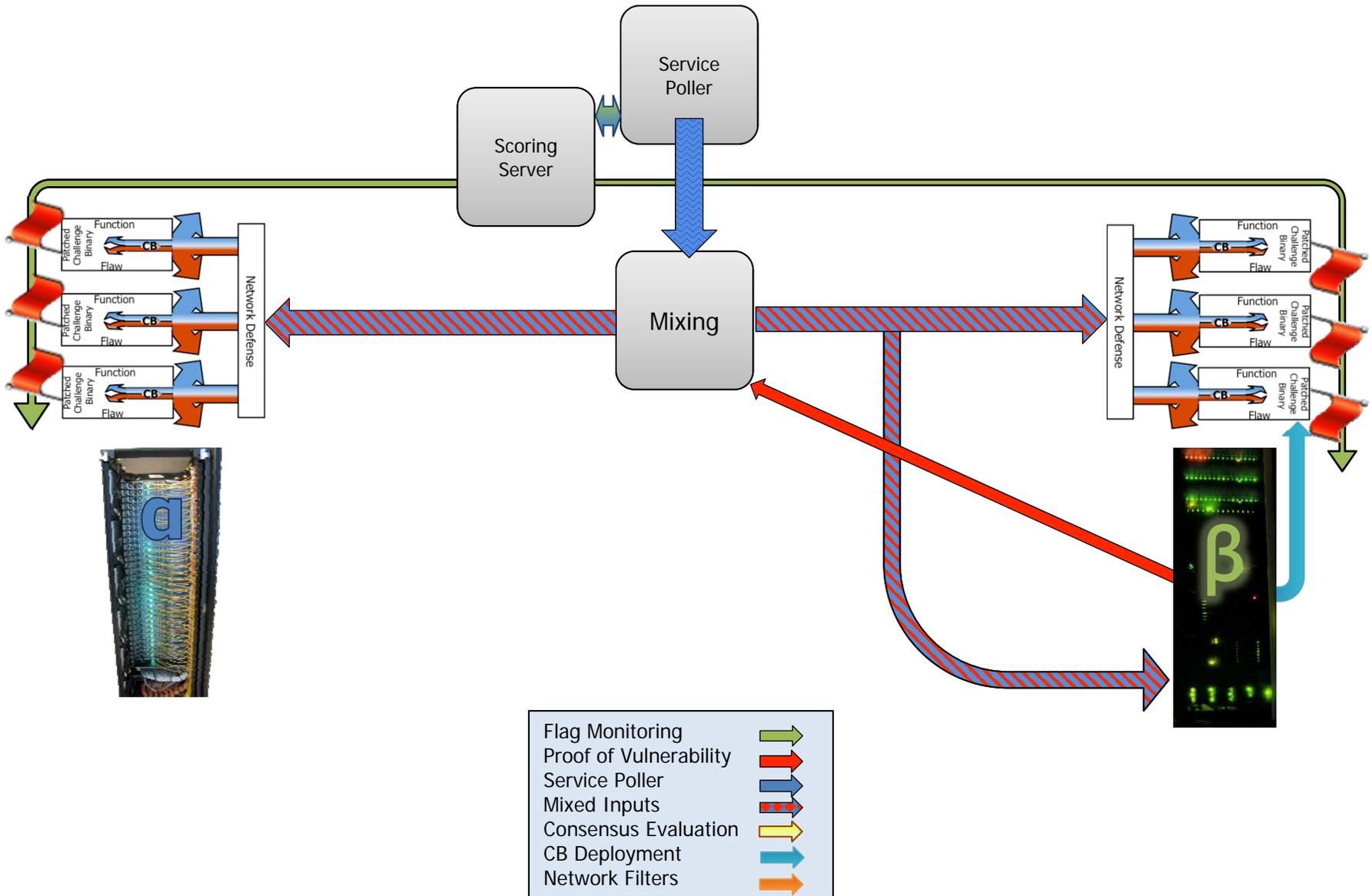
# Consensus Evaluation





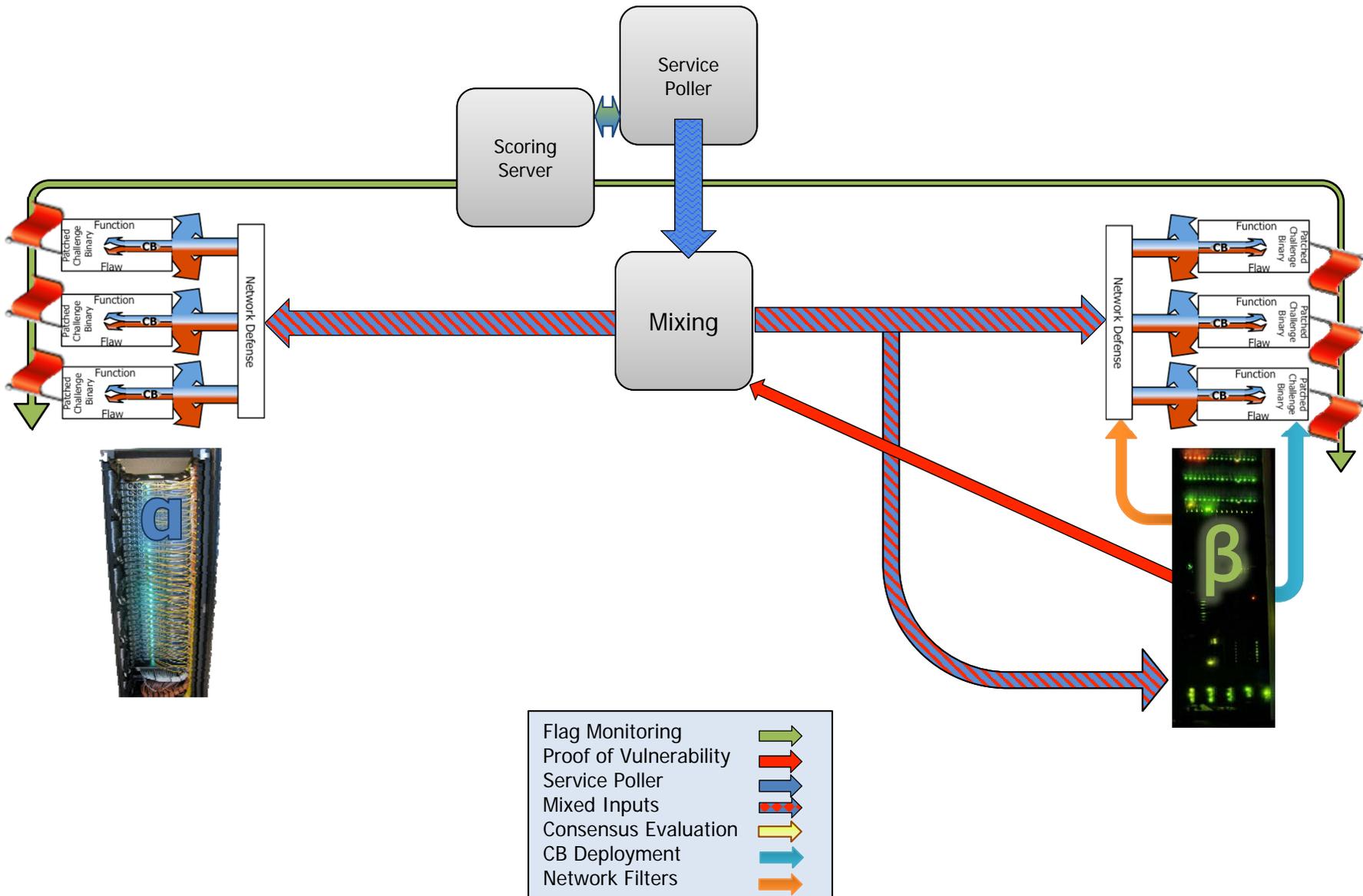


# Consensus Evaluation





# Consensus Evaluation





# Additional security layers often create vulnerabilities...

## Current vulnerability watch list:

Vulnerability Title	Fix Avail?	Date Added
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Local Privilege Escalation Vulnerability	No	8/25/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Denial of Service Vulnerability	Yes	8/24/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Buffer Overflow Vulnerability	No	8/20/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Sanitization Bypass Weakness	No	8/18/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Security Bypass Vulnerability	No	8/17/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Multiple Security Vulnerabilities	Yes	8/16/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/16/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Use-After-Free Memory Corruption Vulnerability	No	8/12/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Remote Code Execution Vulnerability	No	8/10/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Multiple Buffer Overflow Vulnerabilities	No	8/10/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Stack Buffer Overflow Vulnerability	Yes	8/10/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Security-Bypass Vulnerability	No	8/10/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Multiple Security Vulnerabilities	No	8/10/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Buffer Overflow Vulnerability	No	7/29/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Remote Privilege Escalation Vulnerability	No	7/28/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Cross Site Request Forgery Vulnerability	No	7/26/2010
XXXXXXXXXXXXX XXXXXXXXXXXXXXXX Multiple Denial Of Service Vulnerabilities	No	7/22/2010



6 of the vulnerabilities are in security software

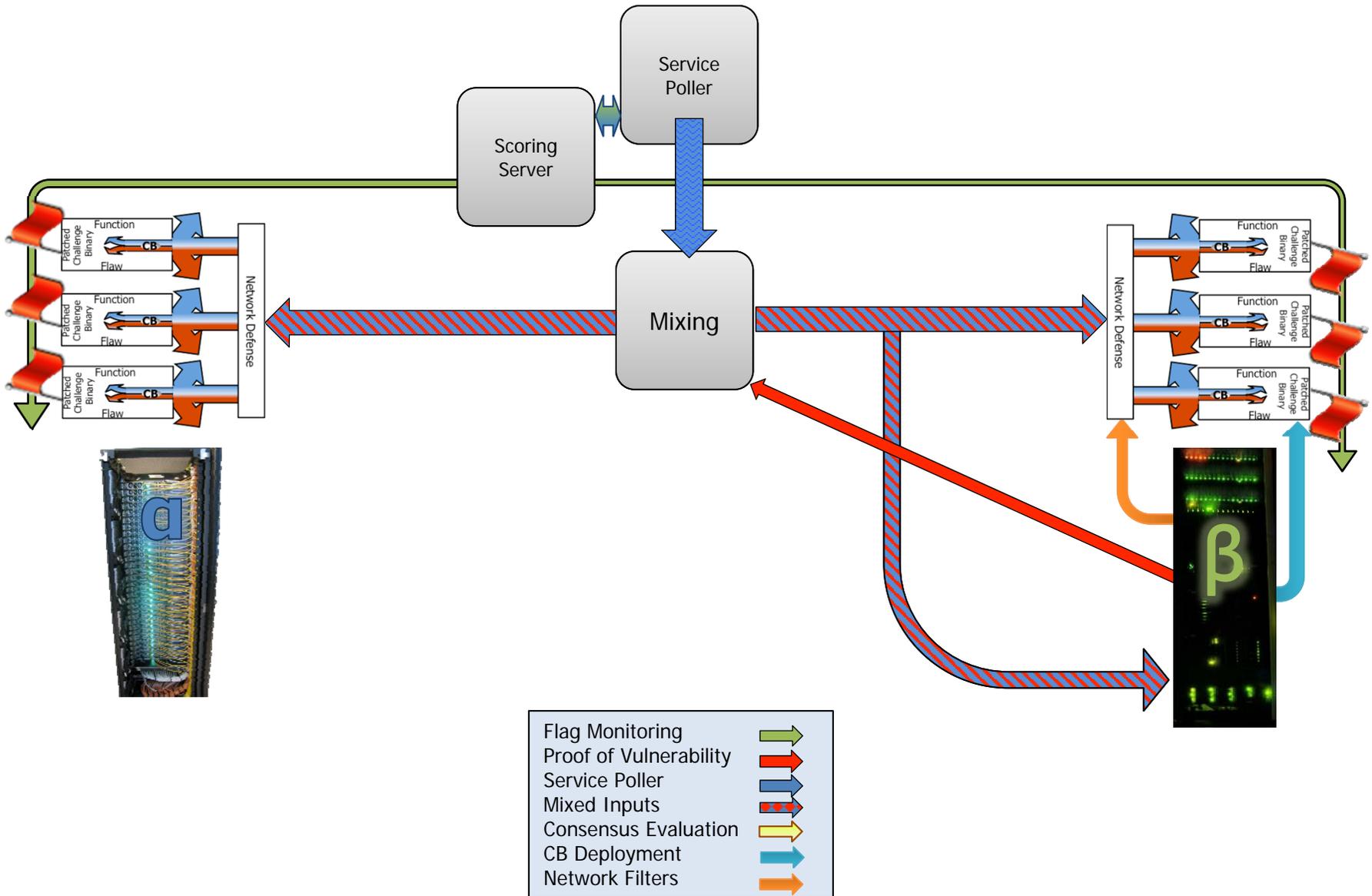


Color Code Key:

Vendor Replied – Fix in development	Awaiting Vendor Reply/Confirmation	Awaiting CC/S/A use validation
-------------------------------------	------------------------------------	--------------------------------

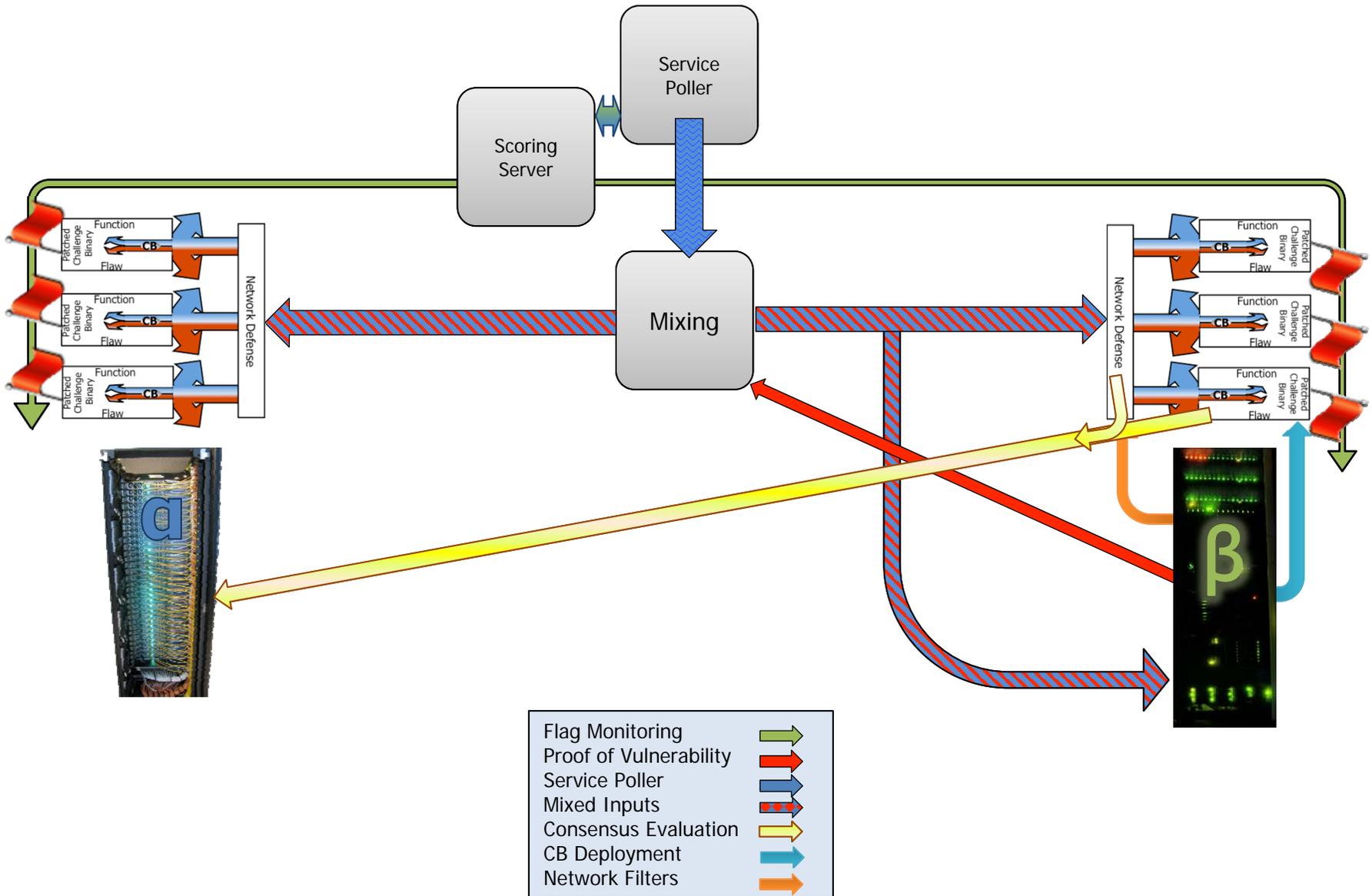


# Consensus Evaluation



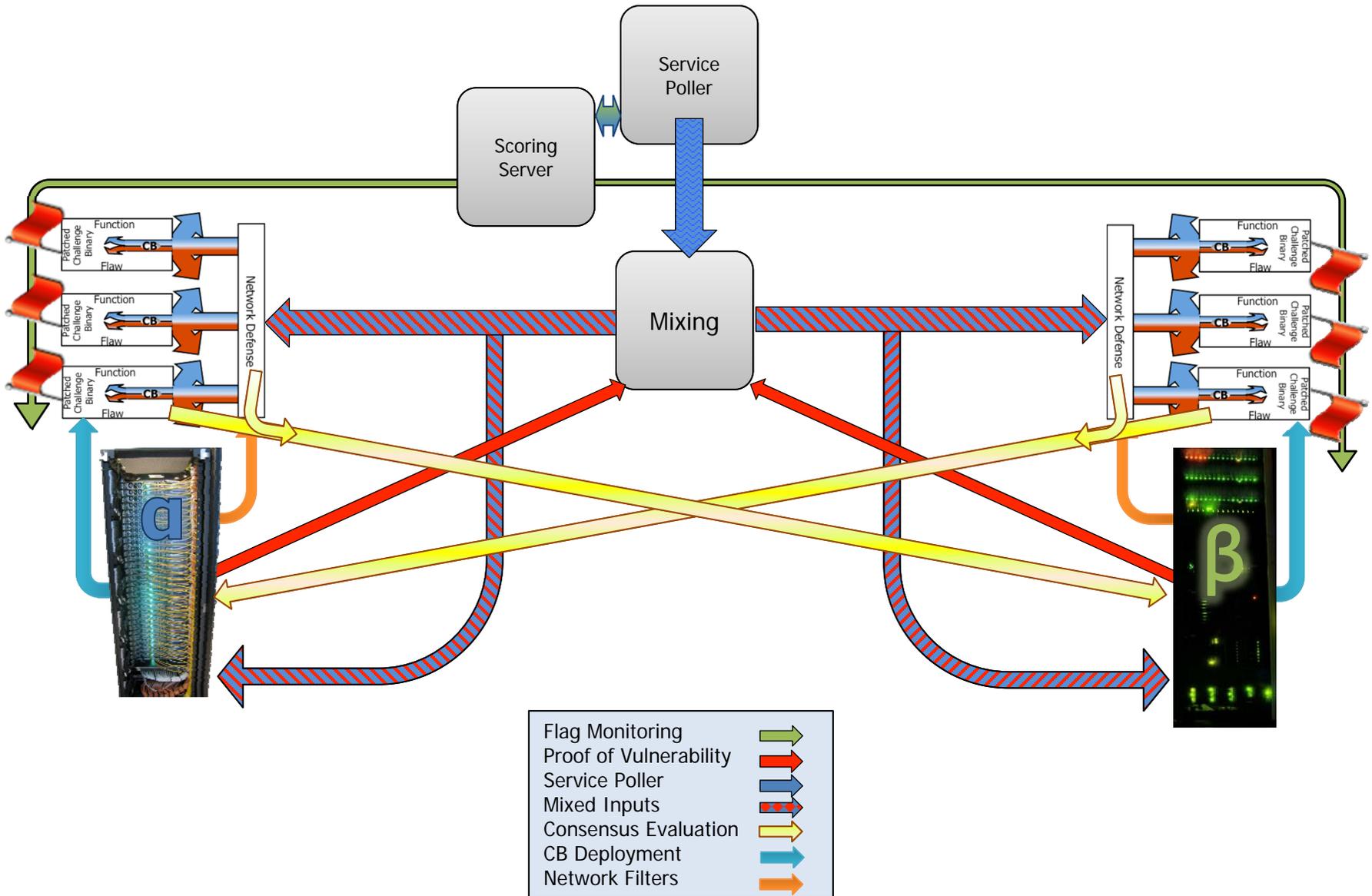


# Consensus Evaluation





# Consensus Evaluation





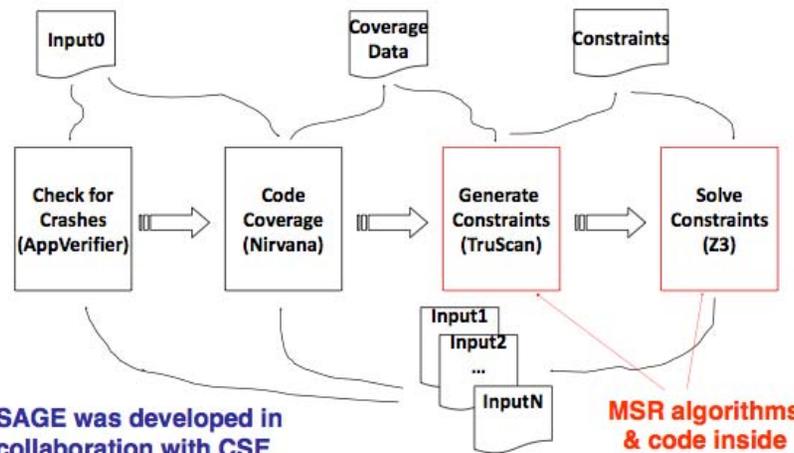
# At Microsoft, a Precursor



Microsoft Research  
TechFest2011  
the & in R&D

## SAGE: Whitebox Fuzzing for Security Testing

**Basic idea:** 1.Run the program with first inputs,  
2.gather constraints on inputs at conditional statements,  
3.use a constraint solver to generate new test inputs,  
4.repeat - possibly forever!



SAGE was developed in collaboration with CSE

**Impact:** since 2007

- 200+ machine years (in largest fuzzing lab in the world)
- 1 Billion+ constraints (largest SMT solver usage ever!)
- 100s of apps, 100s of bugs (missed by everything else...)
- Ex: **1/3** of all Win7 WEX security bugs found by SAGE →
- Bug fixes shipped quietly (no MSRCs) to 1 Billion+ PCs
- Millions of dollars saved (for Microsoft and the world)
- SAGE is now used daily in Windows, Office, etc.

The SAGE team:

MSR: E. Bounimova, P. Godefroid, D. Molnar  
CSE: M. Levin, Ch. Marsh, L. Fang, S. de Jong,...  
+ thanks to all the SAGE users!

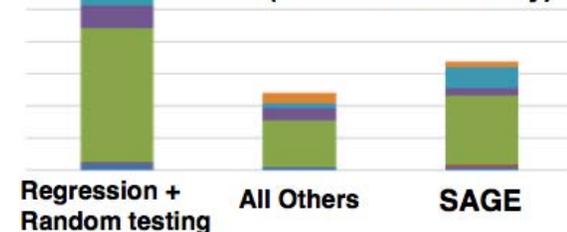
Windows: N. Bartmon, E. Douglas, D. Duran, I. Sheldon  
Office: T. Gallagher, E. Jarvi, O. Timofte

SAGE is the first whitebox fuzzer

Research Challenges:

- How to recover from **imprecision**? PLDI'05, PLDI'11
- How to **scale** to billions of x86 instructions? NDSS'08
- How to check **many properties** together? EMSOFT'08
- How to leverage **grammar** specifications? PLDI'08
- How to deal with **path explosion**? POPL'07, TACAS'08
- How to reason **precisely** about pointers? ISSTA'09
- How to deal with **floating-point** instr.? ISSTA'10
- How to deal with input-dependent **loops**? ISSTA'11
- + research on **constraint solvers**

How bugs were found (Win7 WEX Security)



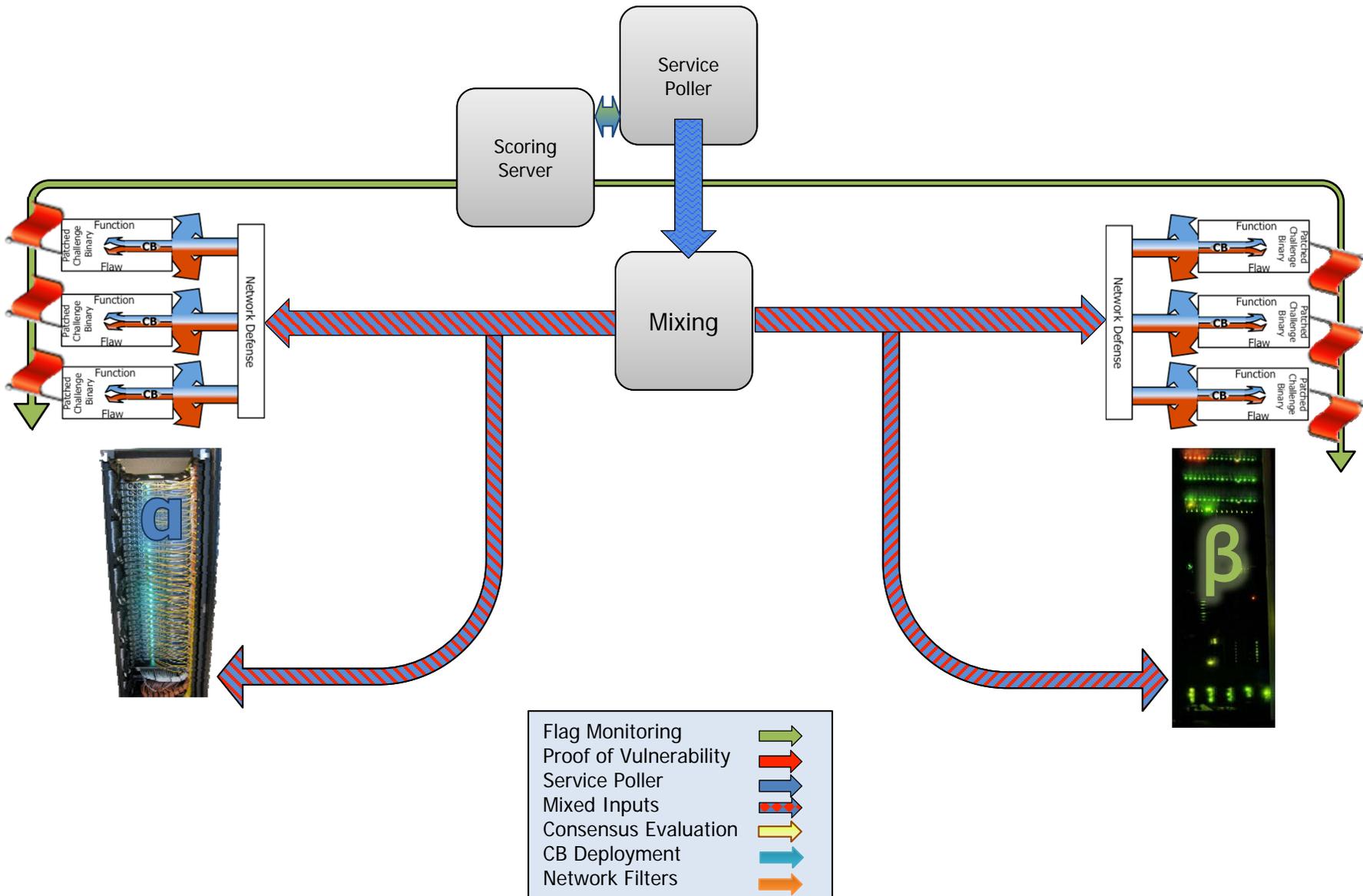
Machine Reasoning accounts for 33% of flaws automatically removed from DoD Windows systems.

Microsoft Research

Microsoft

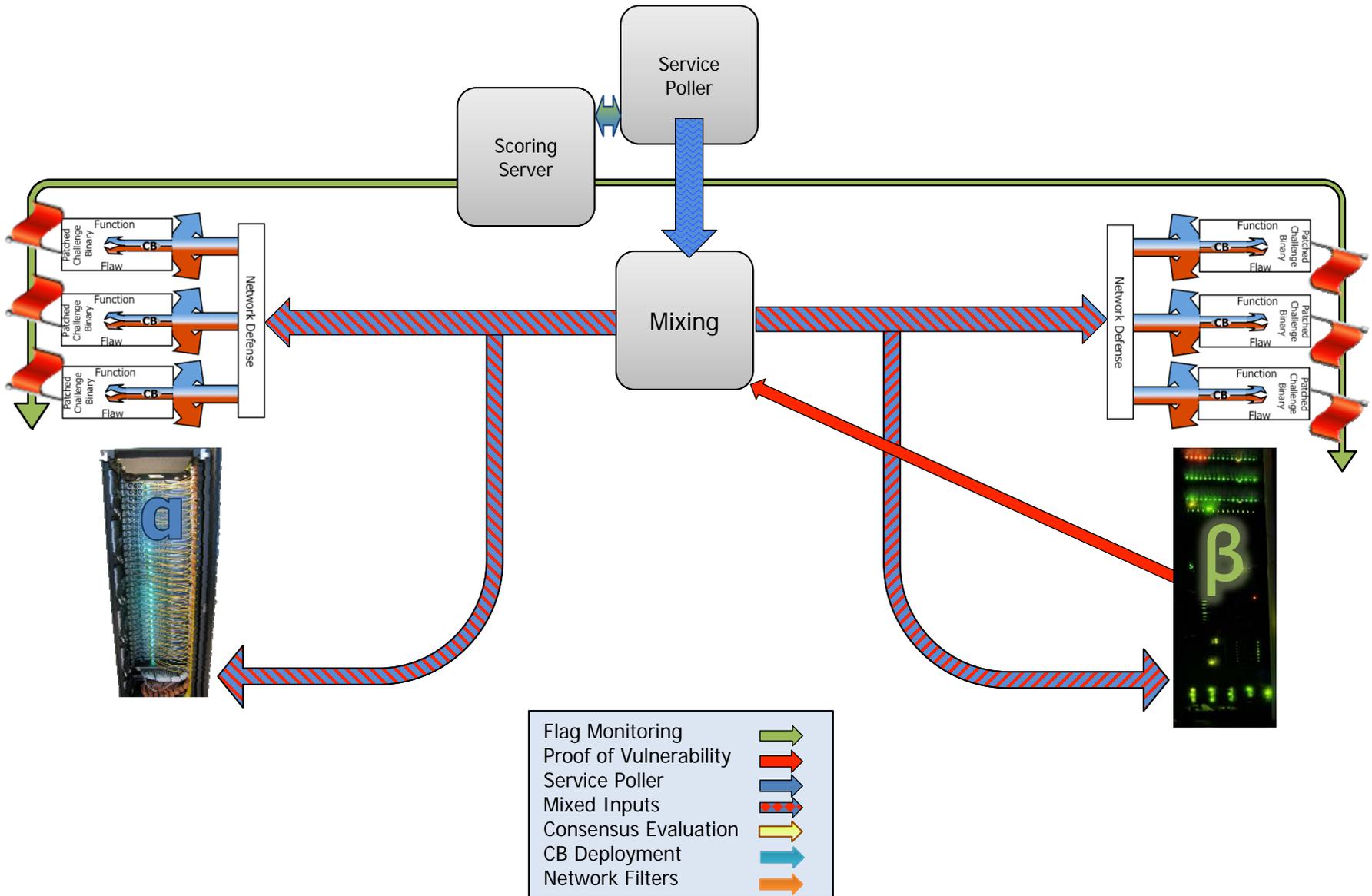


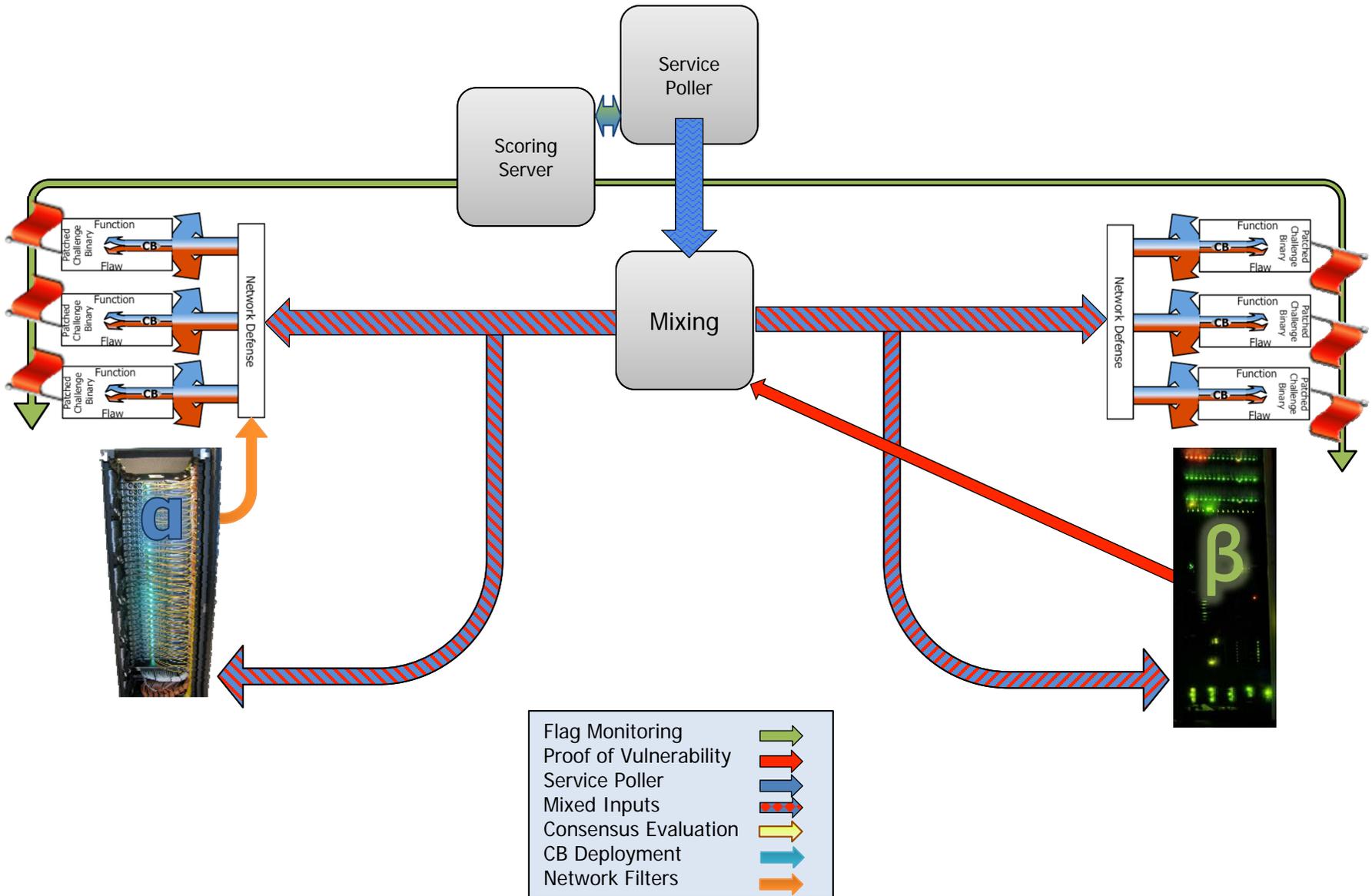
# Consensus Evaluation





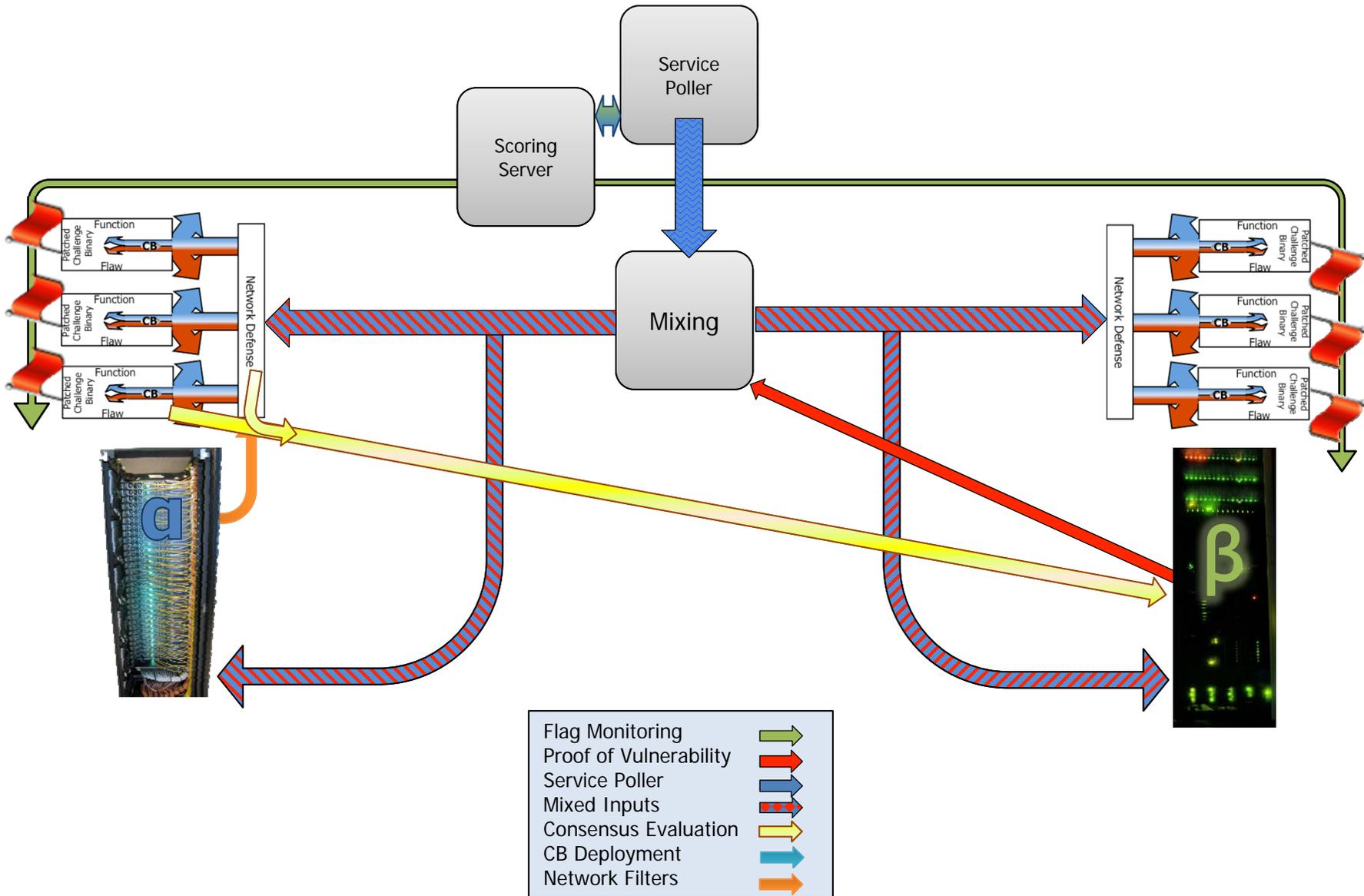
# Consensus Evaluation







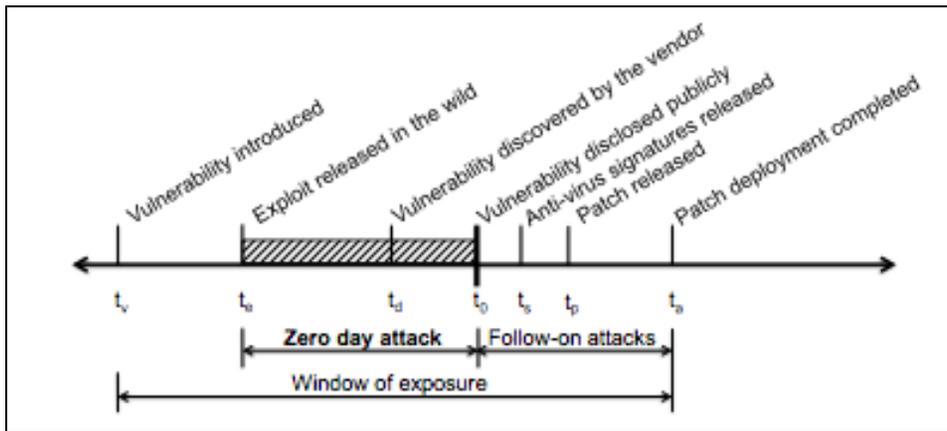
# Consensus Evaluation





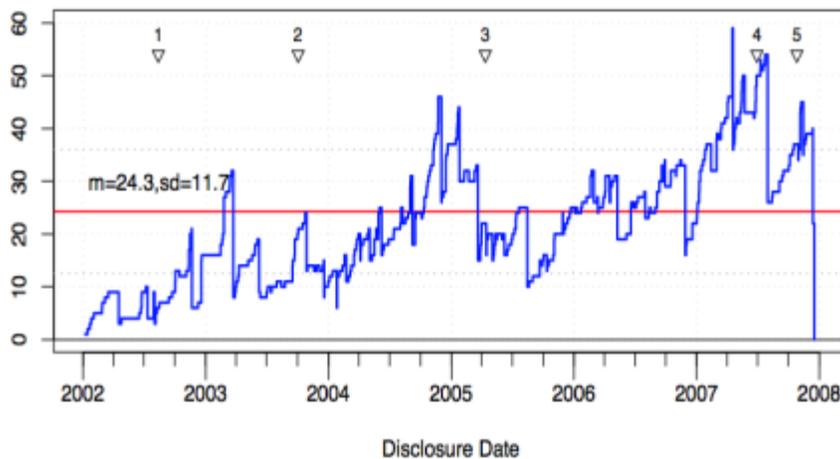


# Defensive Adaptation Speed



\*

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

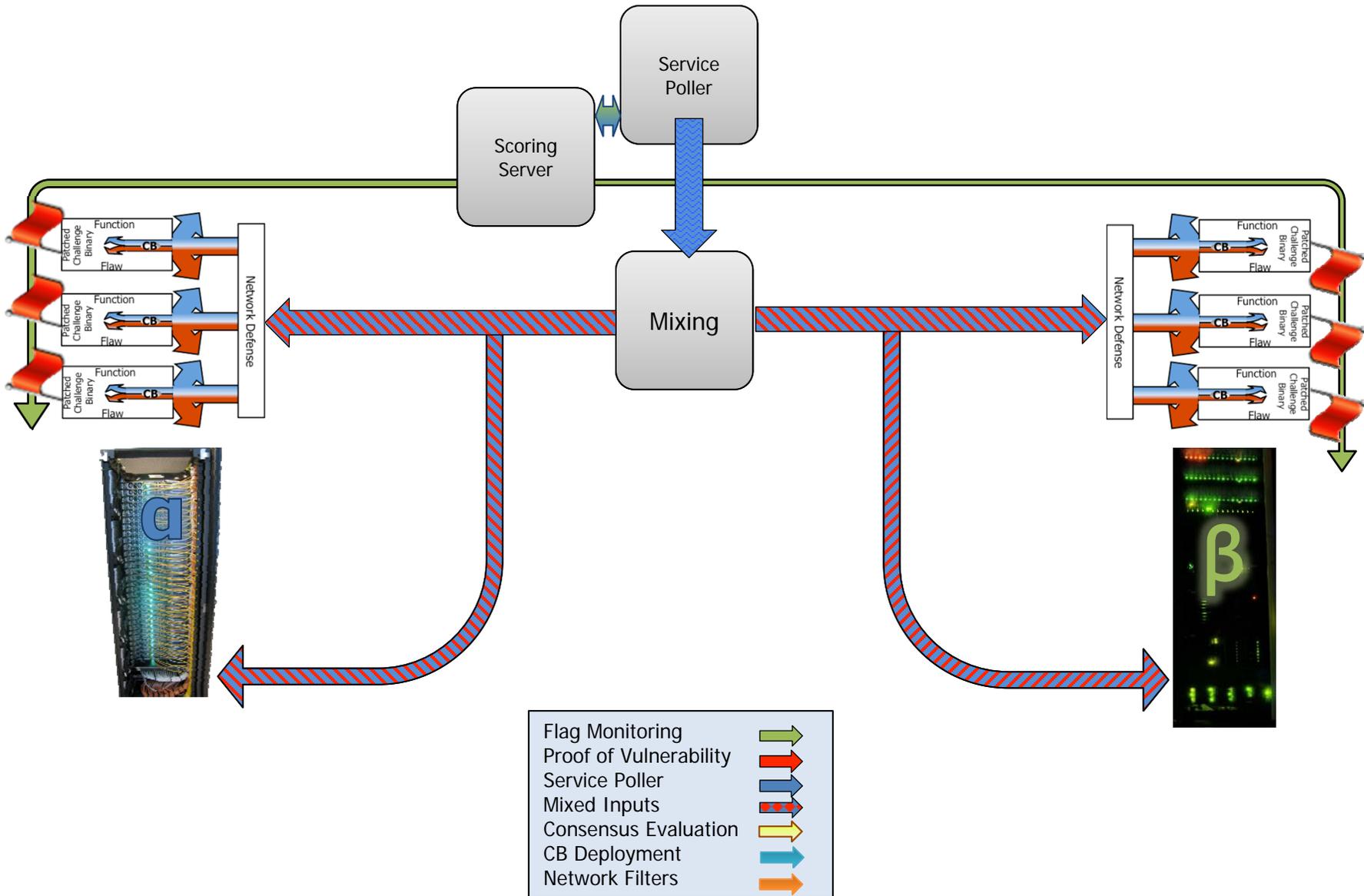


\*\*

...and takes 24 days to patch.

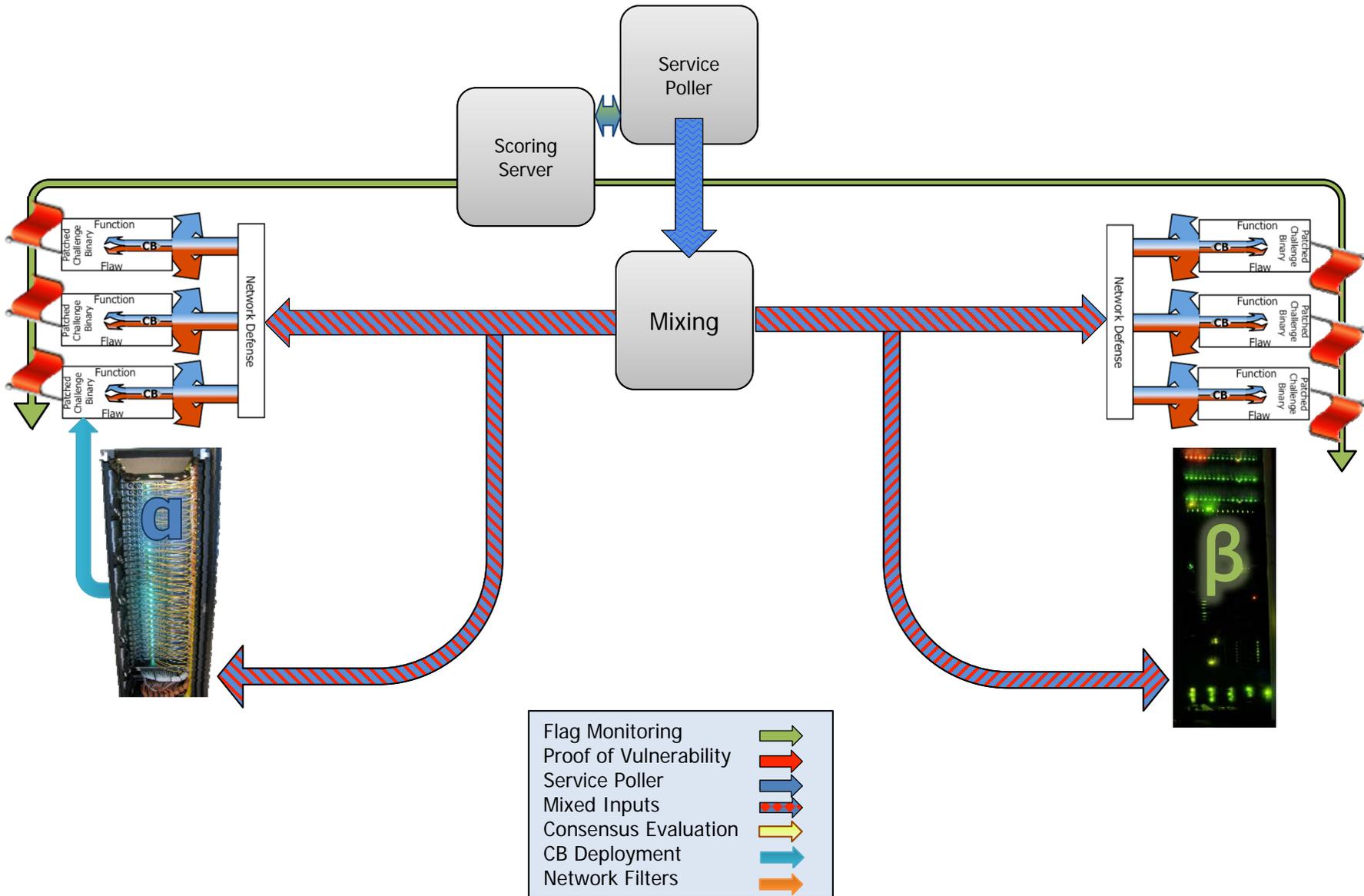


# Consensus Evaluation



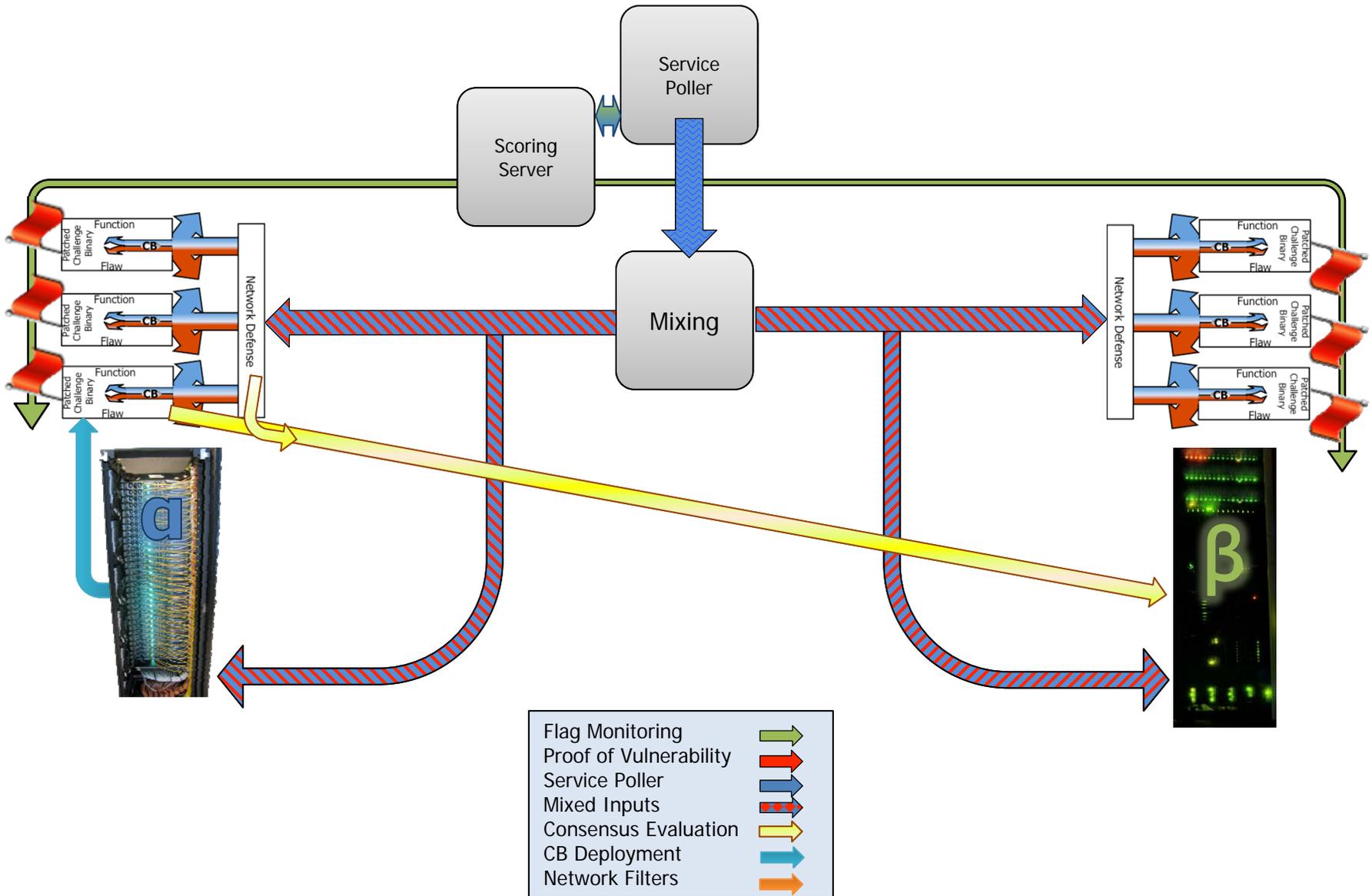


# Consensus Evaluation



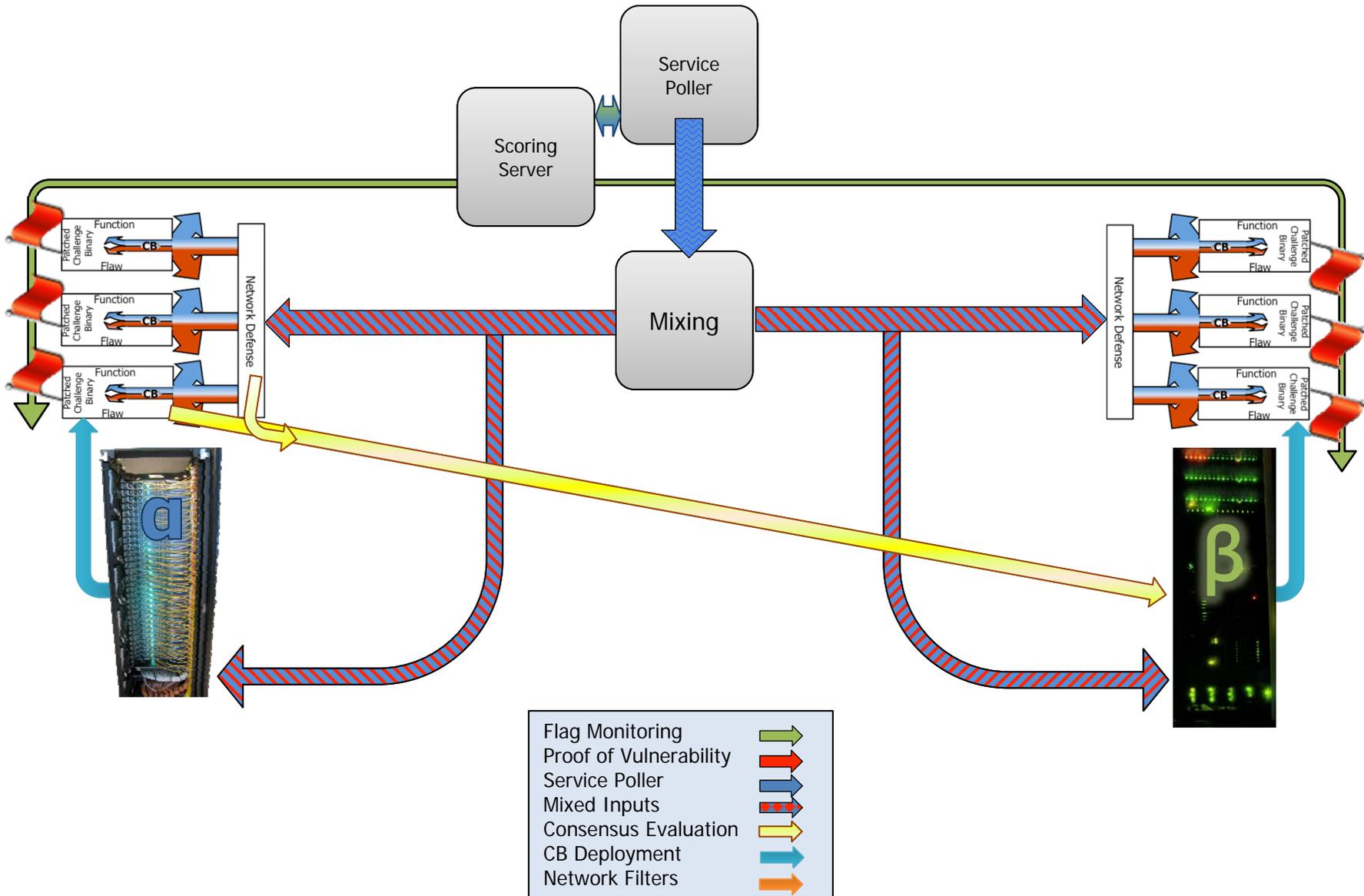


# Consensus Evaluation



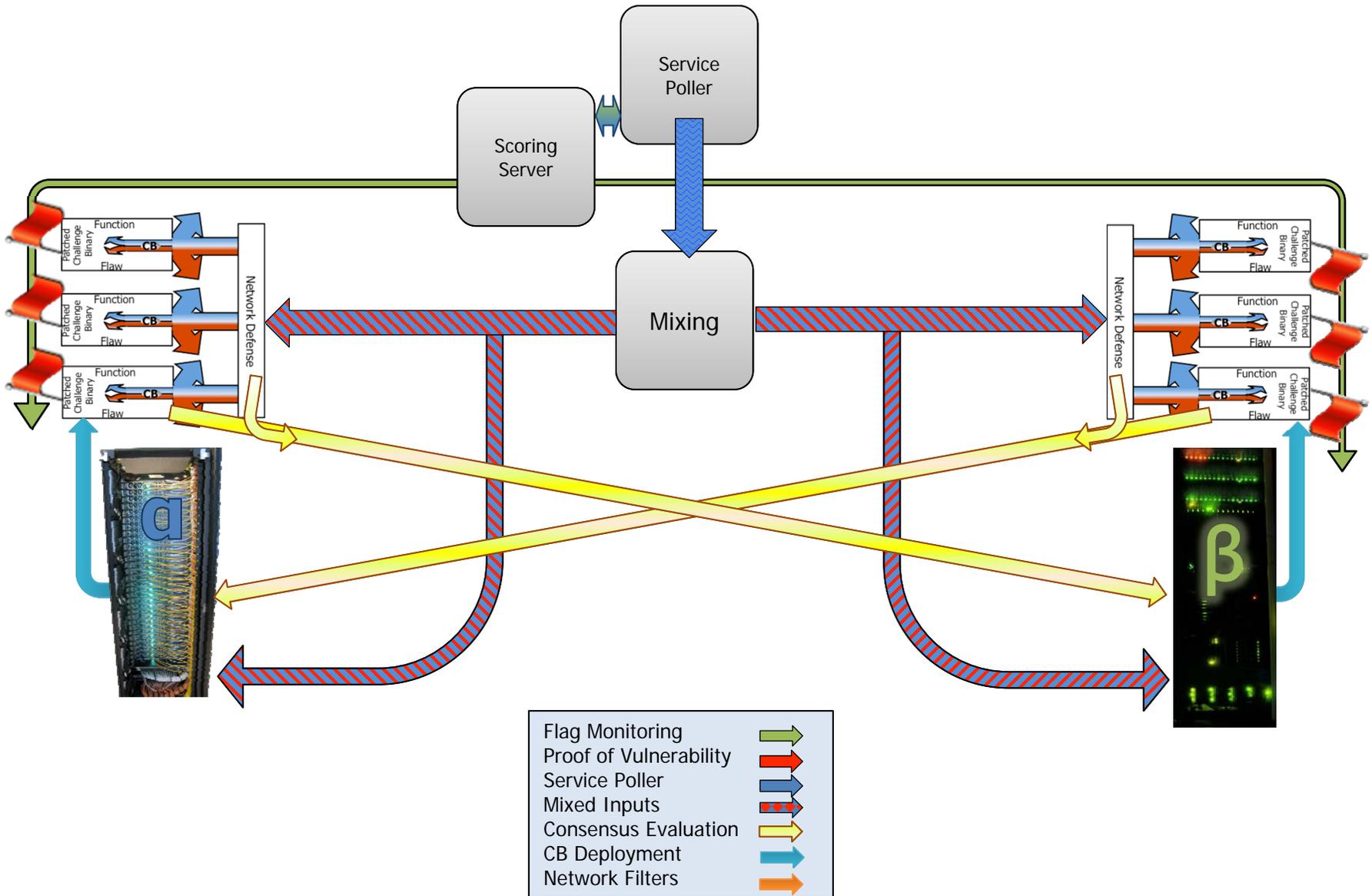


# Consensus Evaluation



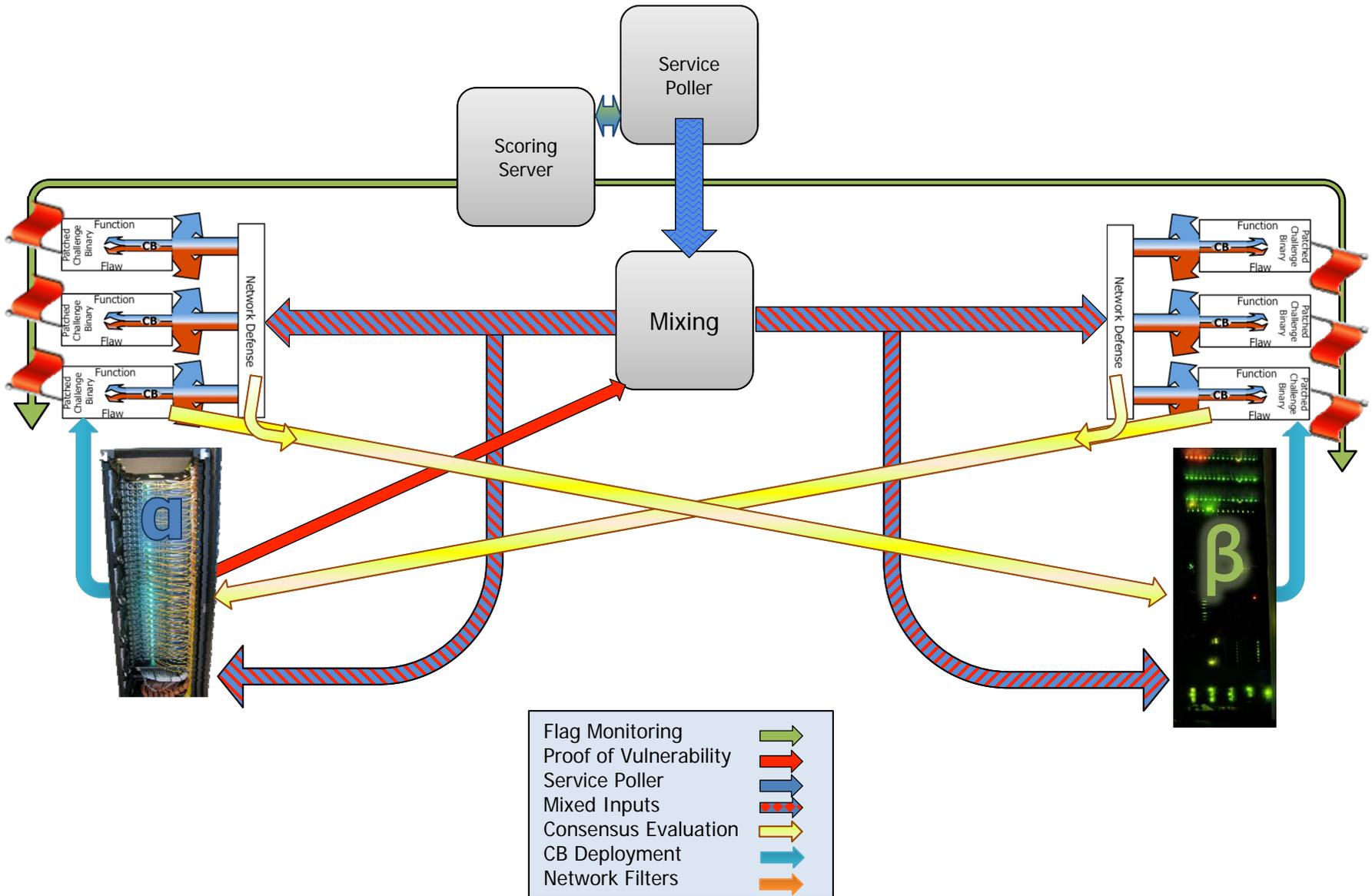


# Consensus Evaluation



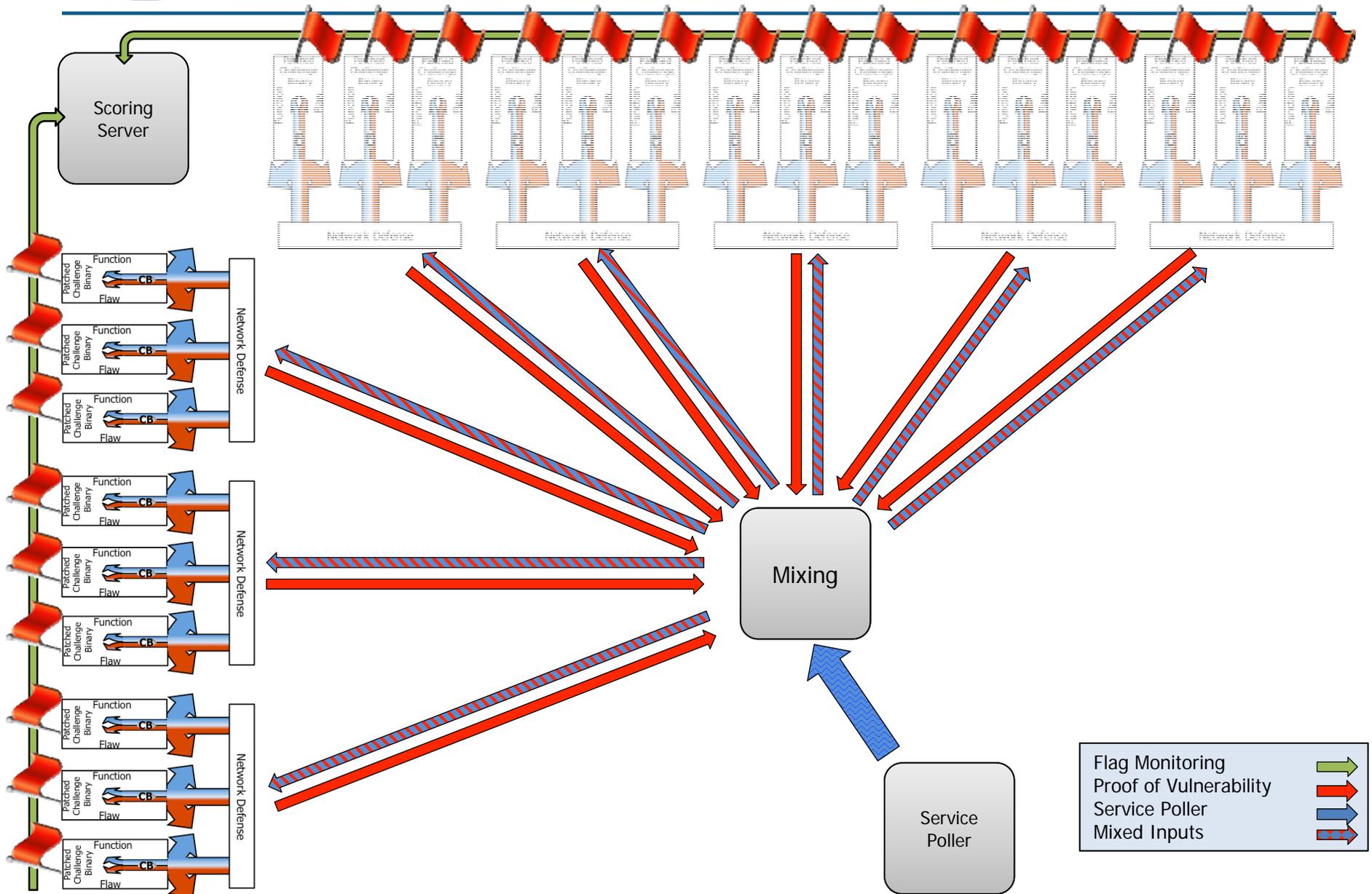


# Consensus Evaluation





# Scheduled Final Event: Multi-Team Real Time Tournament





# Final Event: August 4<sup>th</sup>, 2016: DEF CON





Thank You



Our competitors from around the world

The Cyber Grand Challenge infrastructure team

This talk:

- Dr. Lok Yan @AFRL
- Michael Zhivich, MIT/LL



For more information:

[www.darpa.mil/cybergrandchallenge](http://www.darpa.mil/cybergrandchallenge)

---