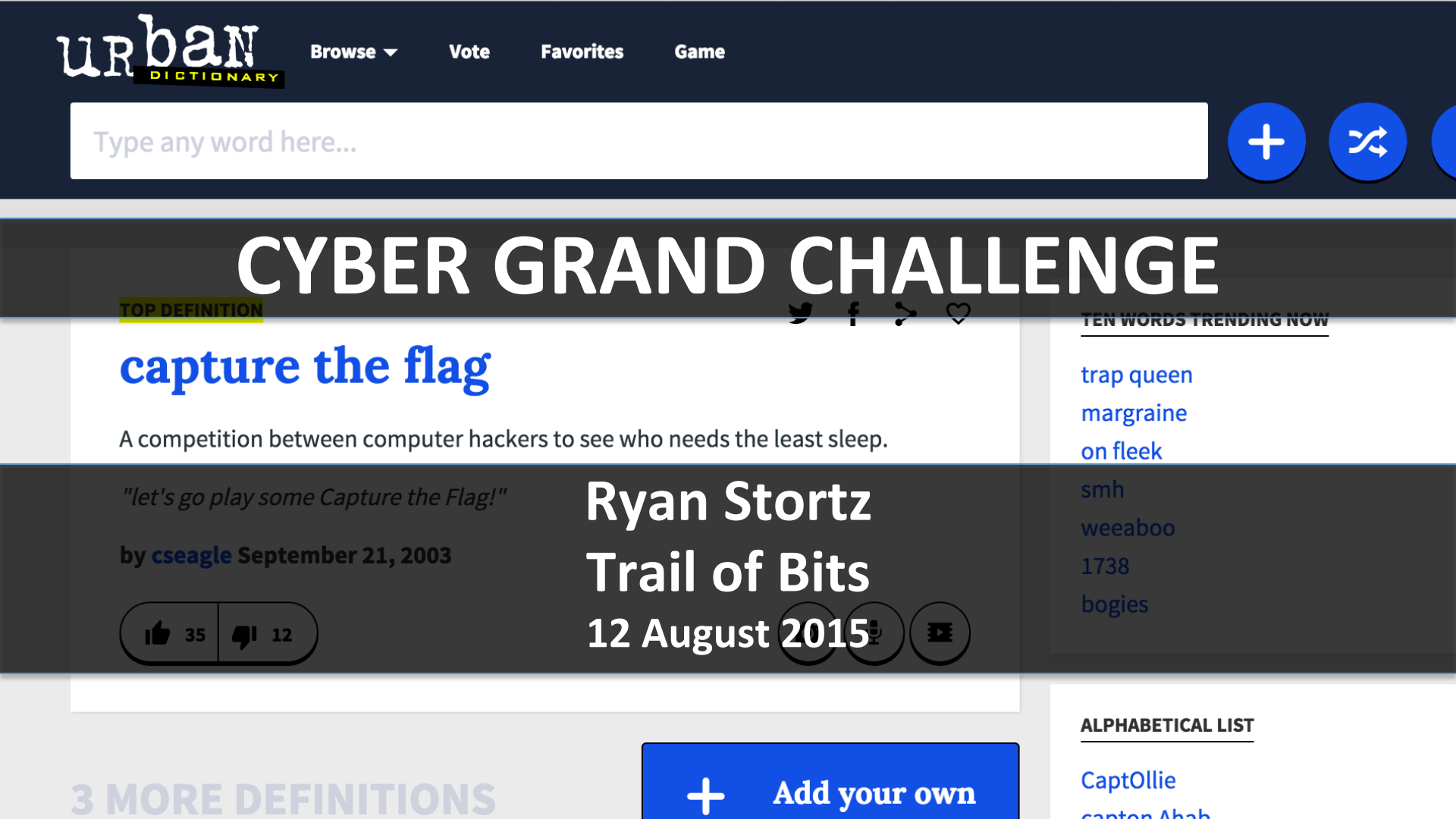

urban DICTIONARY
Browse
Vote
Favorites
Game
Type any word here...
CYBER GRAND CHALLENGE
TOP DEFINITION
capture the flag
A competition between computer hackers to see who needs the least sleep.
"let's go play some Capture the Flag!"
by cseagle September 21, 2003
35
12
Ryan Stortz
Trail of Bits
12 August 2015
TEN WORDS TRENDING NOW
trap queen
margraine
on fleek
smh
weeaboo
1738
bogies
3 MORE DEFINITIONS
Add your own
ALPHABETICAL LIST
CaptOllie

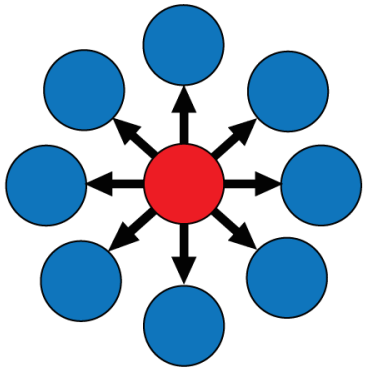
CAPTURE THE FLAG

Capture the Flag

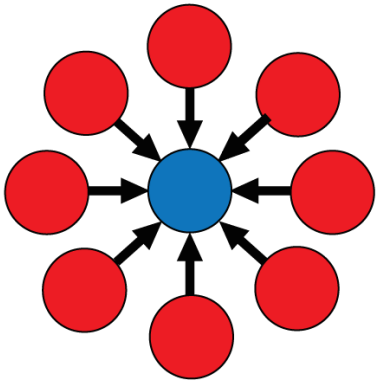
- Capture the Flag is a competition that tests a variety of computer security-related skills.

- Challenge areas include:
  - Cryptanalysis
  - Forensics
  - Network Analysis
  - Network and Application Exploitation
  - Programming
  - Reverse Engineering
  - Trivia

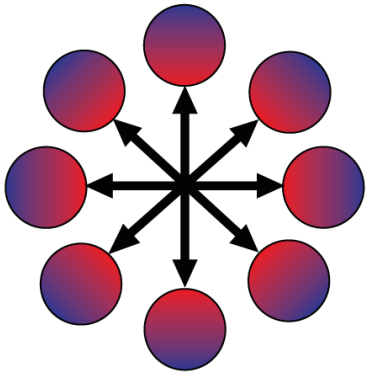
CTF Competition Format



Blue Team

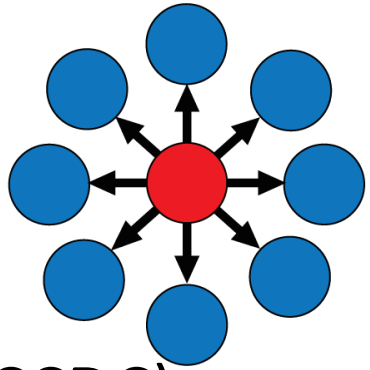


Red Team



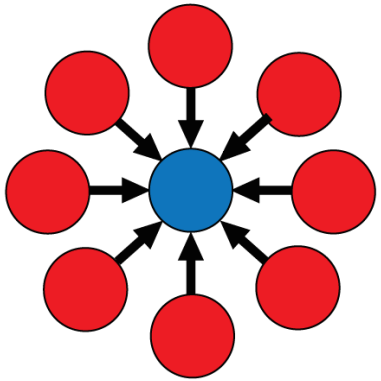
Full Spectrum

Blue Team



- Focused on IT Security
  - Collegiate Cyber Defense Competition (CCDC)
  - CyberPatriot
- Largely IT security competitions
  - Sometimes described as “Patch and Pray”
- Embraces the term “Cyber”

Red Team



- Scoreboard-driven competitions
  - Ghost in the Shellcode
  - PlaidCTF
  - CSAW CTF
- By far the most popular
  - International scores are tracked @ www.ctftime.org

Ghost in the Shellcode 2015

Team Scores

#	Team Name	Score
1	PPP	5451
2	Samurai	4751
3	KAIST GoN	4001
4	TracerTea	3701
5	gallopsled	3601
6	0ops	3601
7	More Smoked Leet Chicken	3551
8	penthackon	3351
9	AcaiBerry	3301
10	Dragon Sector	3301
11	blue-lotus	3051
12	BalalaikaCr3w	2751
13	0x8F	2601
14	shellphish	2501
15	DatNoobs	2401
16	KITCTF	2301
17	Eindbazen	2201
18	0x0	2101
19	Hardc0de	2051
20	StratumAuhuur	1951

Crypto

Knockers
150 Points - Unlocked

Nikoli
250 Points - Unlocked

MTGO
200 Points - Unlocked

vig128
128 Points - Unlocked (*)

Forensics

UNOPENED
100 Points - Locked

Rubicon
400 Points - Unlocked

cloudfs
200 Points - Unlocked

Portal2
400 Points - Unlocked

The Alpha Molecule
200 Points - Unlocked

Programming

Edgy
300 Points - Unlocked

UNOPENED

Trivia

Biggest
1 Points - Unlocked

Reverse Engineering

huffy
300 Points - Unlocked

Web

aart
200 Points - Unlocked

Choose your Pwn Adventure 3

Unbearable Revenge
200 Points - Unlocked

Until the Cows Come Home
100 Points - Unlocked

Overachiever
200 Points - Unlocked

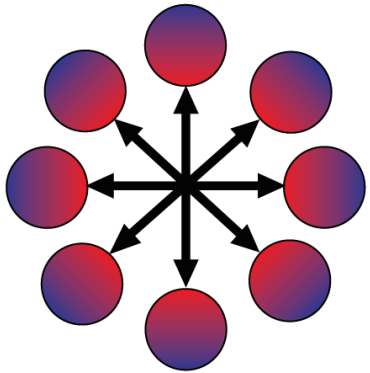
Egg Hunter
250 Points - Unlocked

Pirate's Treasure
500 Points - Unlocked

Fire and Ice
300 Points - Unlocked

Blocky's Revenge

Full Spectrum



- Attack-Defense
  - DEFCON CTF
  - RuCTFe
- Most dynamic of the three formats
  - Basically unrestricted cyber war in a single room
- Teams host and patch custom vulnerable services, while attacking the same services run by other teams.

Why does this matter?

- Many people believe CTF is the best crucible for tool & strategy development
  - Specifically Full Spectrum competitions

- HatesIrony made custom tools for:
  - Reverse Engineering
  - Packet Analysis
  - Automated Network Exploitation
  - Key/Flag Submission
  - Runtime Application Defenses
  - Network De-anonymizationTools

- ...and we threw them out each year as the game evolved

HATES IRONY

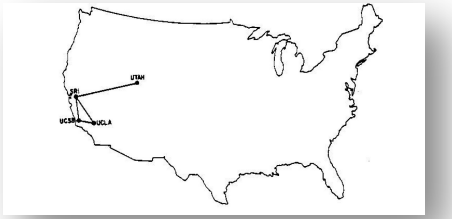
https://binary.ninja/


THE CYBER GRAND CHALLENGE

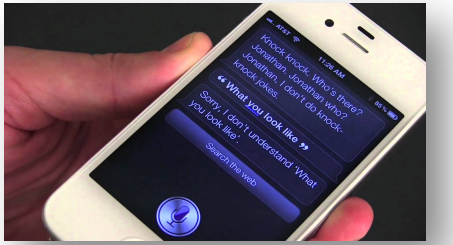
Defense Advanced Research Projects Agency

Agency overview

Formed	1958
Headquarters	Arlington, Virginia, U.S.
Employees	240
Annual budget	US$2.8 billion
Agency executive	Arati Prabhakar, Director
Parent agency	U.S. Department of Defense
Website	www.darpa.mil



ARPANET
1969



Siri
2003

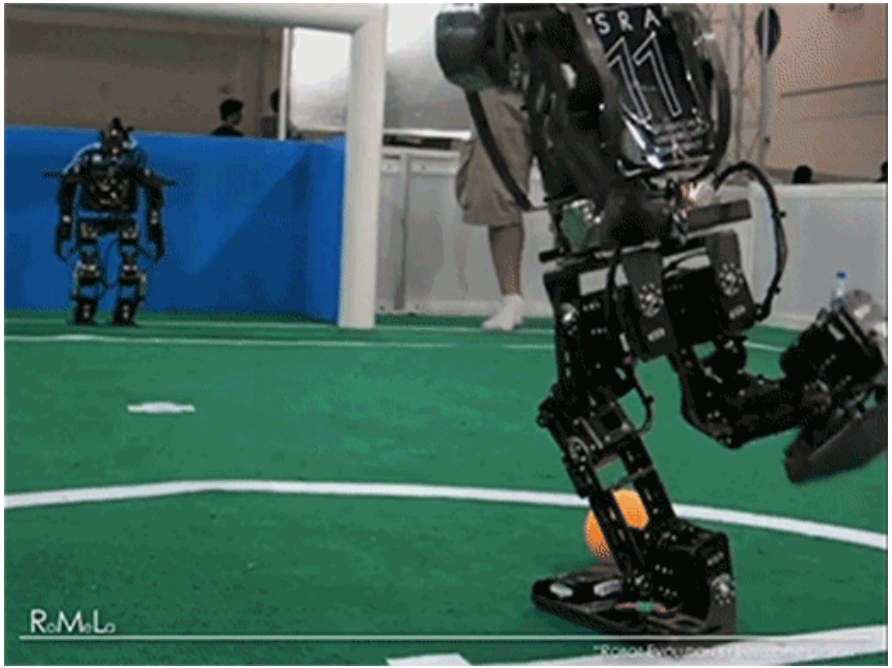
Grand Challenge
2004

CTF Nerds + DARPA

Cyber Grand Challenge

“Cyber Grand Challenge (CGC) is a contest to build high-performance computers capable of playing in a Capture-the-Flag style cyber-security competition.”

Humans > Robots



* Yes, even at software

Competition Setup

- Teams build “Cyber Reasoning Systems”
  - Fully autonomous systems
- Given a bundle of Challenge Binaries (CBs)
  - CB := “Pwnables”
- Teams create Proof of Vulnerabilities (PoV)
  - PoVs are inputs that trigger vulnerable code paths (via a crash)
- CRSs will compete against each other

DECREE

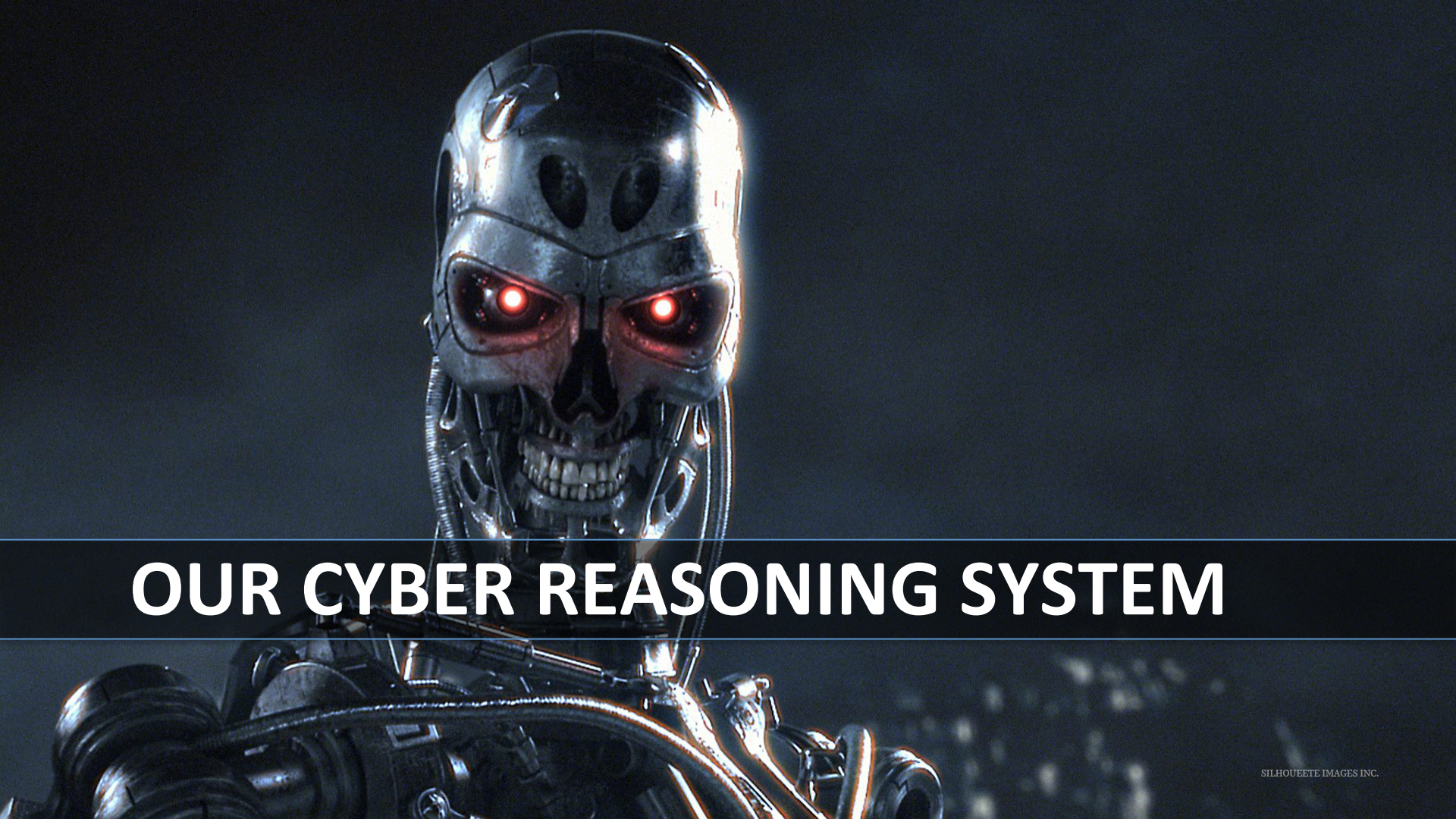
- OS: DECREE
  - DARPA Experimental Cyber Research Evaluation Environment
  - Similar to Linux x86
  - Executables are statically-linked ELFs
  - Implemented as a new binfmt
- Removes complexity
- Very clever implementation

#	Syscall	Linux Eq.
1	_terminate	exit
2	transmit	send
3	receive	recv
4	fdwait	select
5	allocate	mmap
6	deallocate	munmap
7	random	n/a

```c
//CADET's first C program

#include <libcgc.h>
#include "libc.h"

#define HI "\nWelcome to Palindrome Finder\n\n"
#define ASK "\tPlease enter a possible palindrome: "
#define YES "\t\tYes, that's a palindrome!\n\n"
#define NO "\t\tNope, that's not a palindrome\n\n"
#define EASTEREGG "\n\nEASTER EGG!\n\n"

int check();

int main(void) {
    int r;

    if (transmit_all(1, HI, sizeof(HI)-1) != 0) {
        _terminate(0);
    }

    while(1){
        if (transmit_all(1, ASK, sizeof(ASK)-1) != 0) {
            _terminate(0);
        }
        r = check();
        if (r == -1){
            break;
        }
        else if (r == 0){
            if (transmit_all(1, NO, sizeof(NO)-1) != 0) {
                _terminate(0);
            }
        }
        else{
            if (transmit_all(1, YES, sizeof(YES)-1) != 0) {
                _terminate(0);
            }
        }
    }
    return 0;
}
```

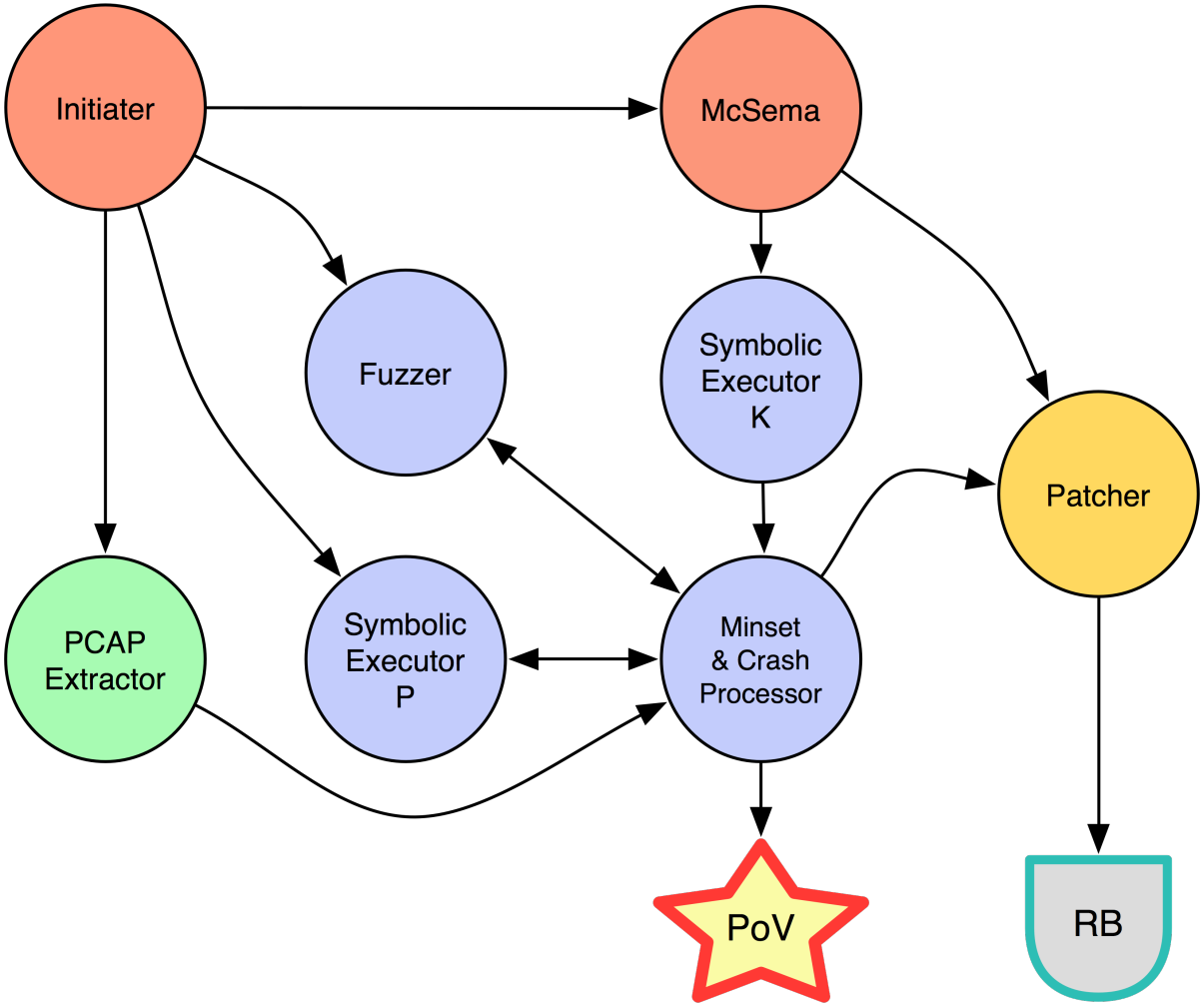
```c
int check(){
    int len = -1;
    int i;
    int pal = 1;
    char string[64];
    for (i = 0; i < sizeof(string); i++)
        string[i] = '\0';
#ifdef PATCHED
    if (receive_delim(0, string, sizeof(string), '\n') != 0)
        return -1;
#else
    if (receive_delim(0, string, 128, '\n') != 0)
        return -1;
#endif
    for(i = 0; string[i] != '\0'; i++){
        len++;
    }
    int steps = len;
    if(len % 2 == 1){
        steps--;
    }
    for(i = 0; i <= steps/2; i++){
        if(string[i] != string[len-1-i]){
            pal = 0;
        }
    }
    if(string[0] == '^'){
        if (transmit_all(1, EASTEREGG, sizeof(EASTEREGG)-1) != 0) {
            _terminate(0);
        }
    }
    return pal;
}
```

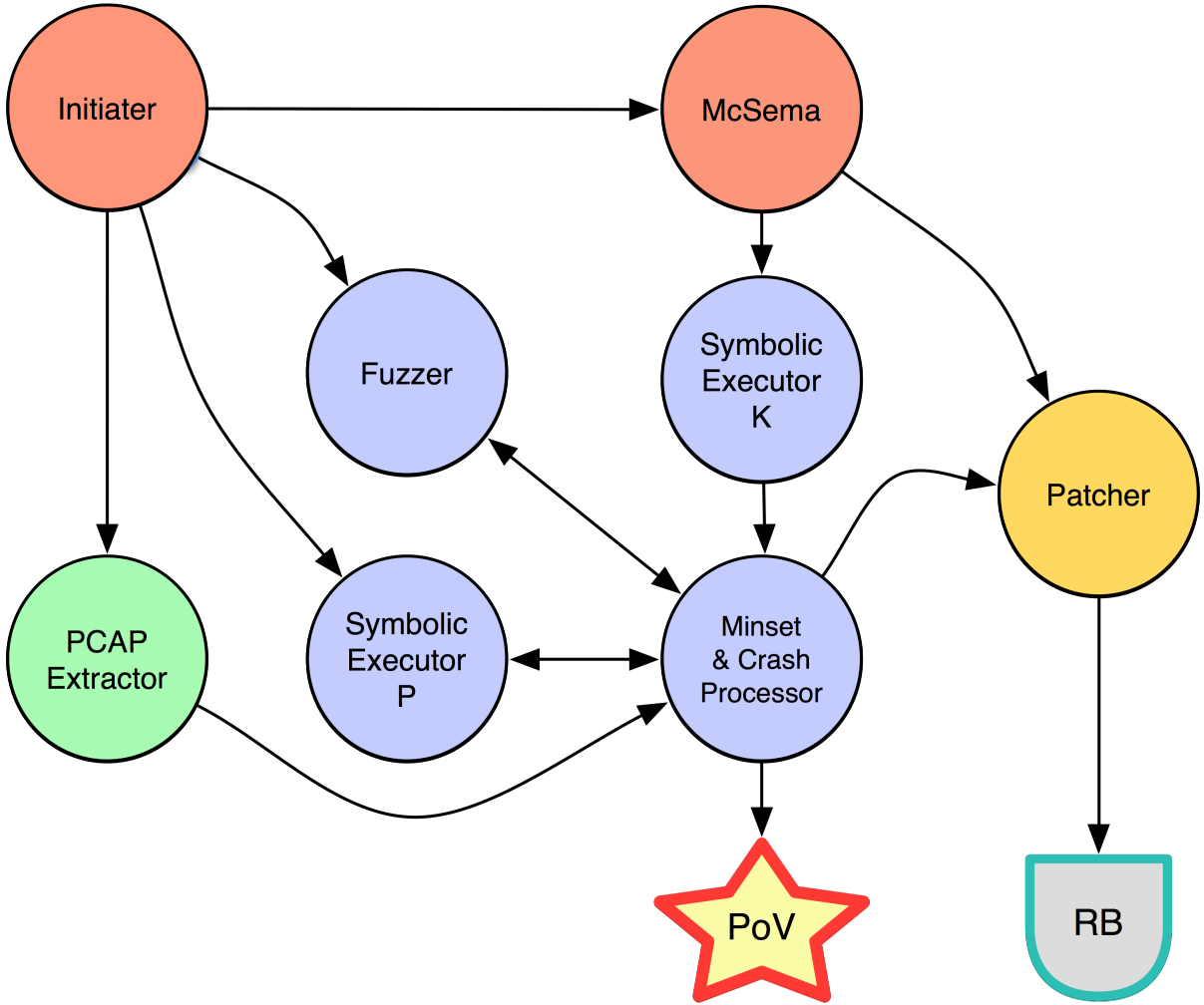
Example Challenge: CADET_0001



OUR CYBER REASONING SYSTEM

SILHOUEETE IMAGES INC.


Initiater
McSema
Fuzzer
Symbolic
Executor
K
Patcher
PCAP
Extractor
Symbolic
Executor
P
Minset
& Crash
Processor
PoV
RB


Initiater
McSema
Fuzzer
Symbolic Executor K
Patcher
PCAP Extractor
Symbolic Executor P
Minset & Crash Processor
PoV
RB

Bug Finding

- Fuzzing
- Symbolic Execution
- Concolic Execution
  - Concrete + Symbolic….get it? Ha ha…


FUZZING

Fuzzing

- Fuzzing is a technique where you take valid inputs, corrupt them slightly, feed them through an application, and hope for a crash.

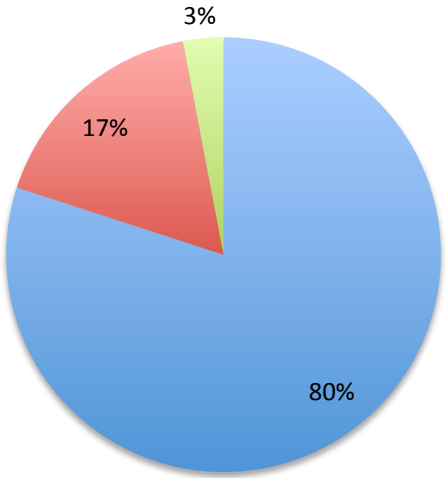
- Extremely effective for many file formats
- Requires good input data to be truly effective

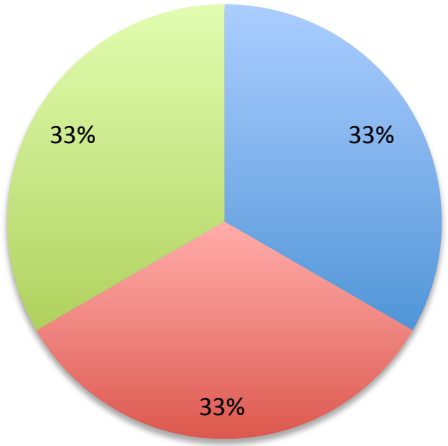
Granary - Binary Translation

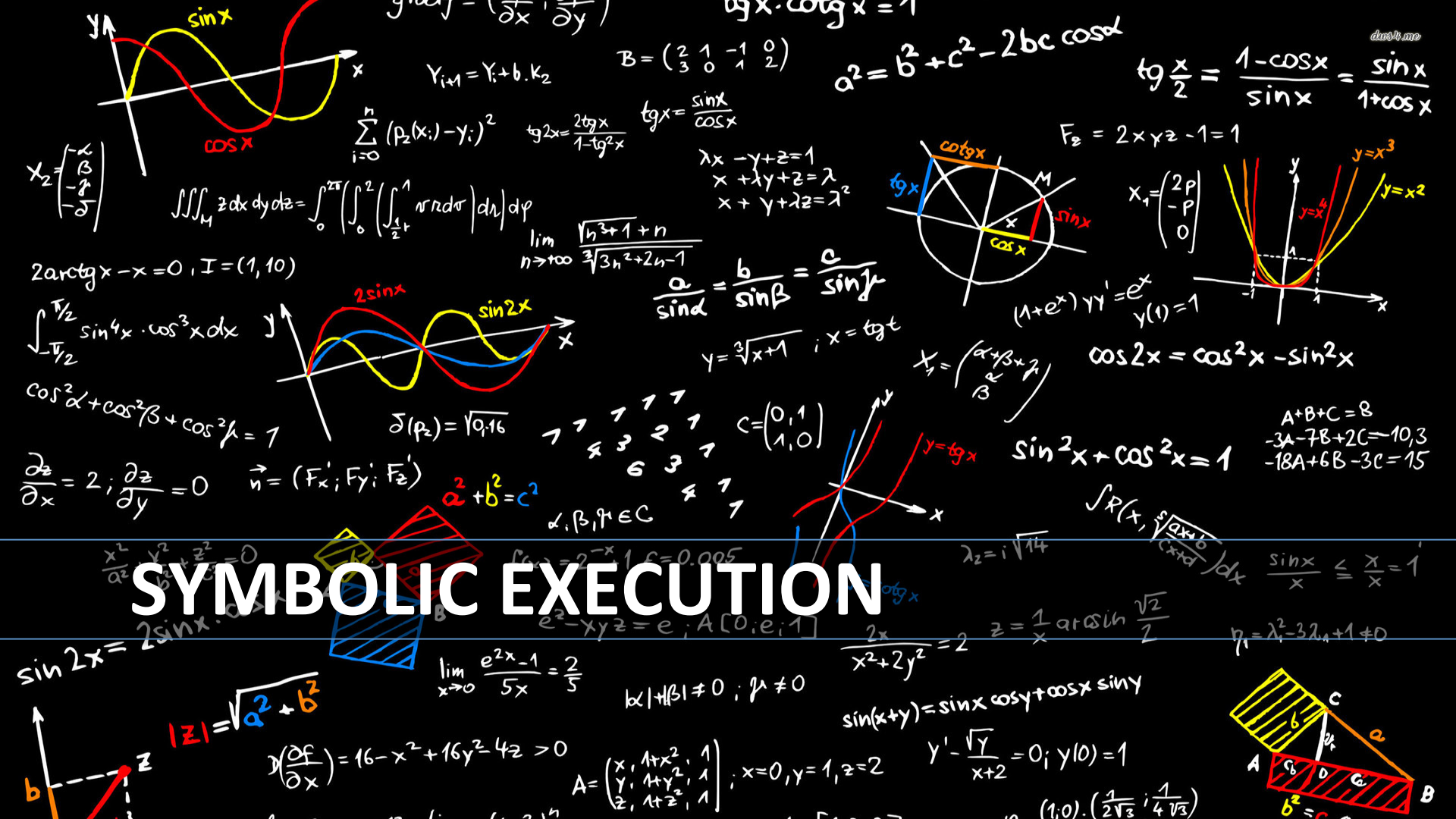
- Persistent Code Cache
- Code Coverage
- No System IO
  - Granary emulated all of it
- One instance = 400,000 test cases/hour
  - We had ~10,000 instances for the qualifier

Minset

- Collects the “Minimum Set” of inputs to cause the maximum amount of path coverage.






SYMBOLIC EXECUTION

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	-
Y	**RV**	
-	-	

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	0
Y	**RV**	
0	-	

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	0
Y	**RV**	
0	-	

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	0
Y	**RV**	
0	-	

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	0
Y	**RV**	
1	-	

Concrete Execution : Symbolic Execution :: **Arithmetic** : Algebra

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
5	12	0
Y	**RV**	
1	1	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	-
Y	**RV**	
-	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	0
Y	**RV**	
0	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	0
Y	**RV**	
0	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	2∧(a==2)
Y	**RV**	
0	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	2∧(a==2)
Y	**RV**	
0	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	2∧(a==2)
Y	**RV**	
1∧(b>10)	-	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	$2 \wedge (a==2)$
Y	**RV**	
$1 \wedge (b>10) \vee b \wedge (b \leq 10)$	-	

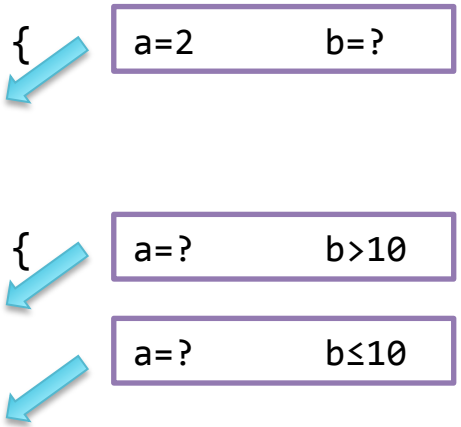
Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```

A	B	X
?	?	$2 \wedge (a==2)$
Y	**RV**	
$1 \wedge (b>10) \vee b \wedge (b \leq 10)$	$2 \wedge (a==2) + 1 \wedge (b>10) \vee b \wedge (b \leq 10)$	

Concrete Execution : **Symbolic Execution** :: Arithmetic : **Algebra**

```
 1 int ex(int a, int b)
 2 {
 3     int x = 0, y = 0;
 4
 5     if (a == 2) {
 6         x = 2;
 7     }
 8
 9     if (b > 10) {
10         y = 1;
11     } else {
12         y = b;
13     }
14
15     return x + y;
16 }
```



#	A	B
1	==2	>10
2	==2	≤10
3	!=2	>10
4	!=2	≤10

Input generation with SMT-LIB

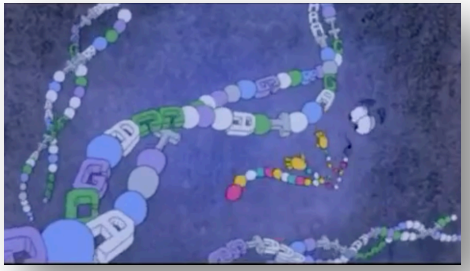

PATCHING

Automated Patching

- We convert CBs to LLVM IR with McSema
- We identify all taint input sources
- Locate all tainted loads and stores
  - In loops only (as an optimization)
- Added checks to verify they're safe

How Replacement Binaries are made

Challenge Binary



Patch

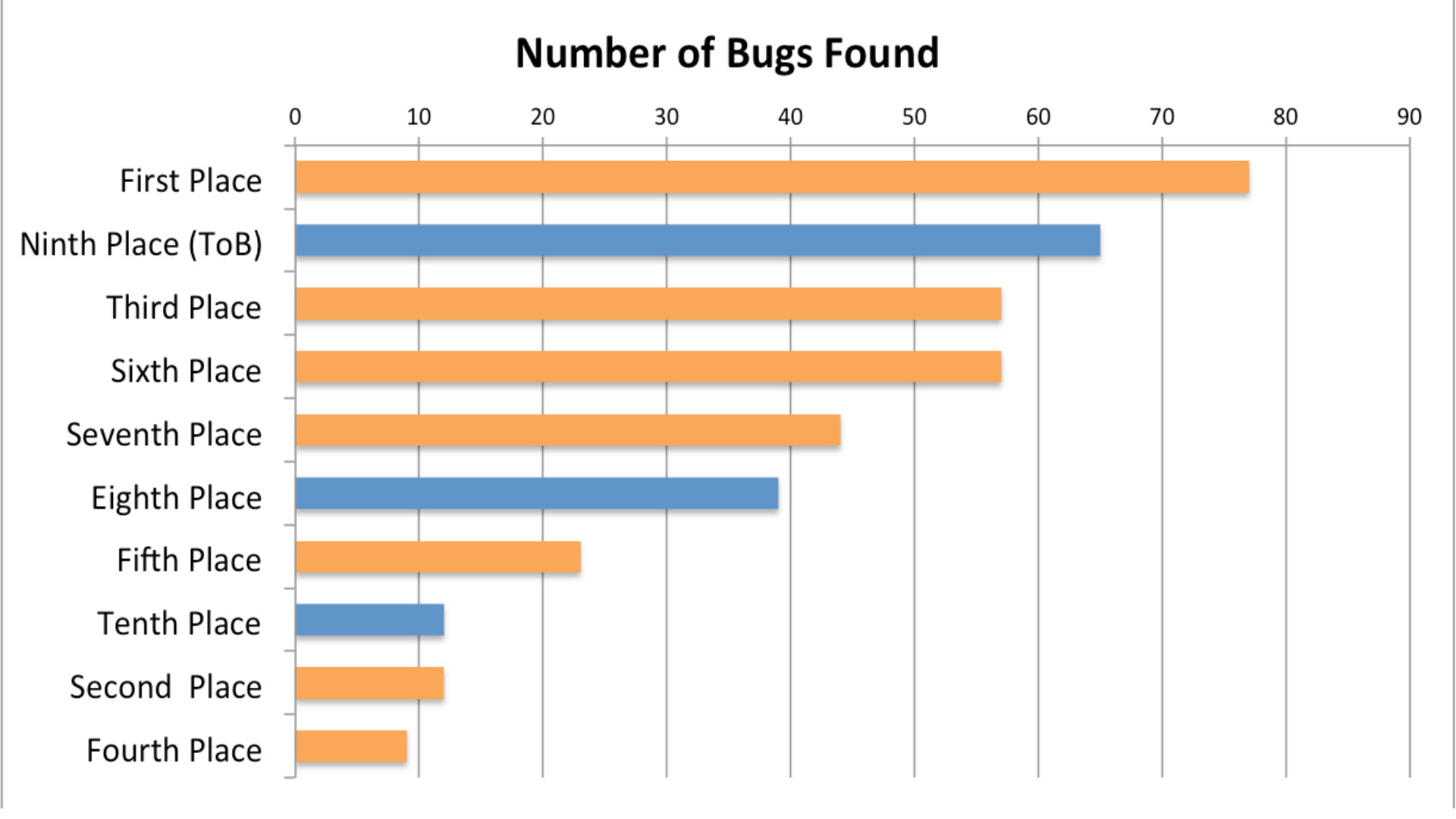
McSema

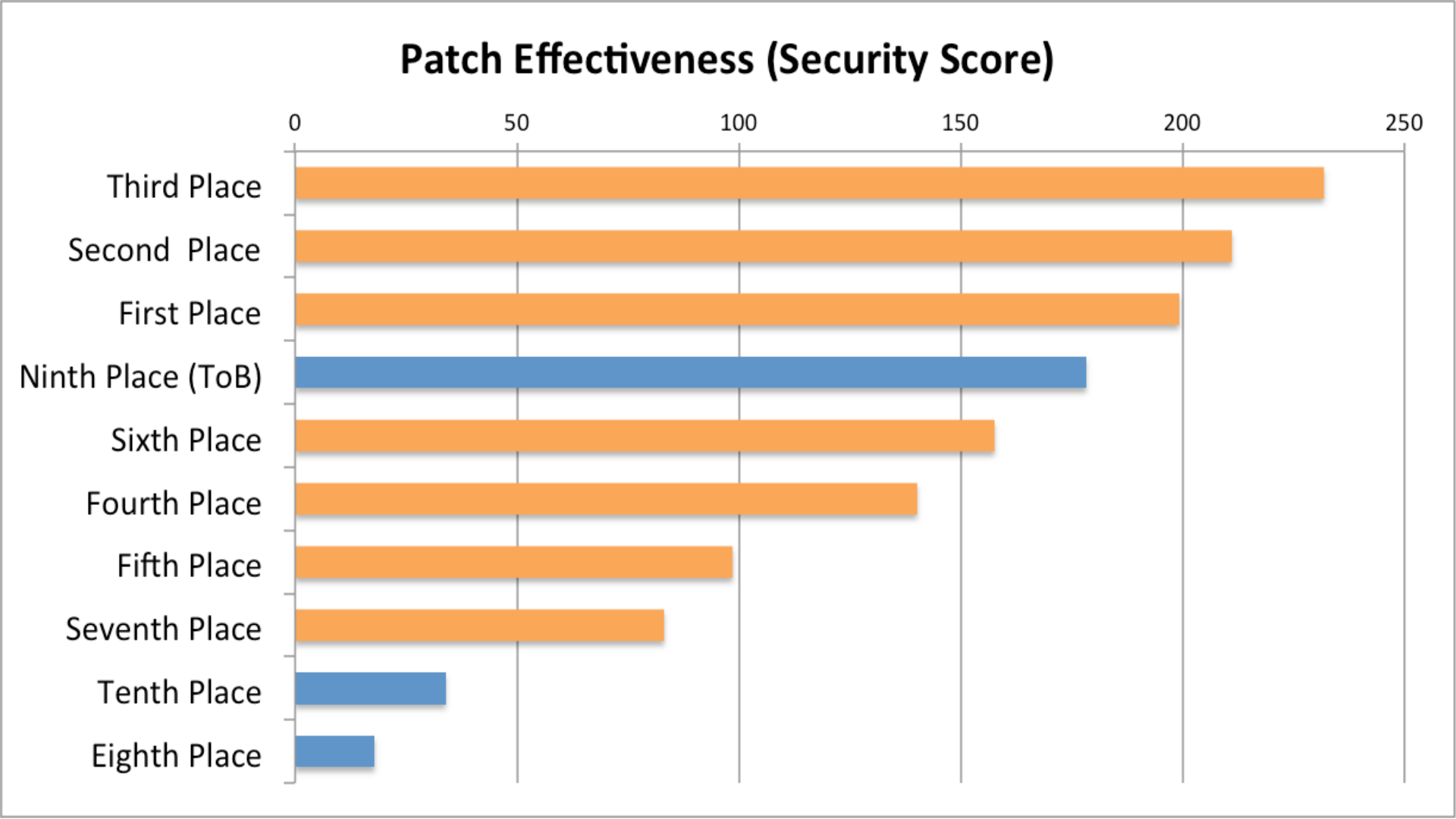
Replacement Binary

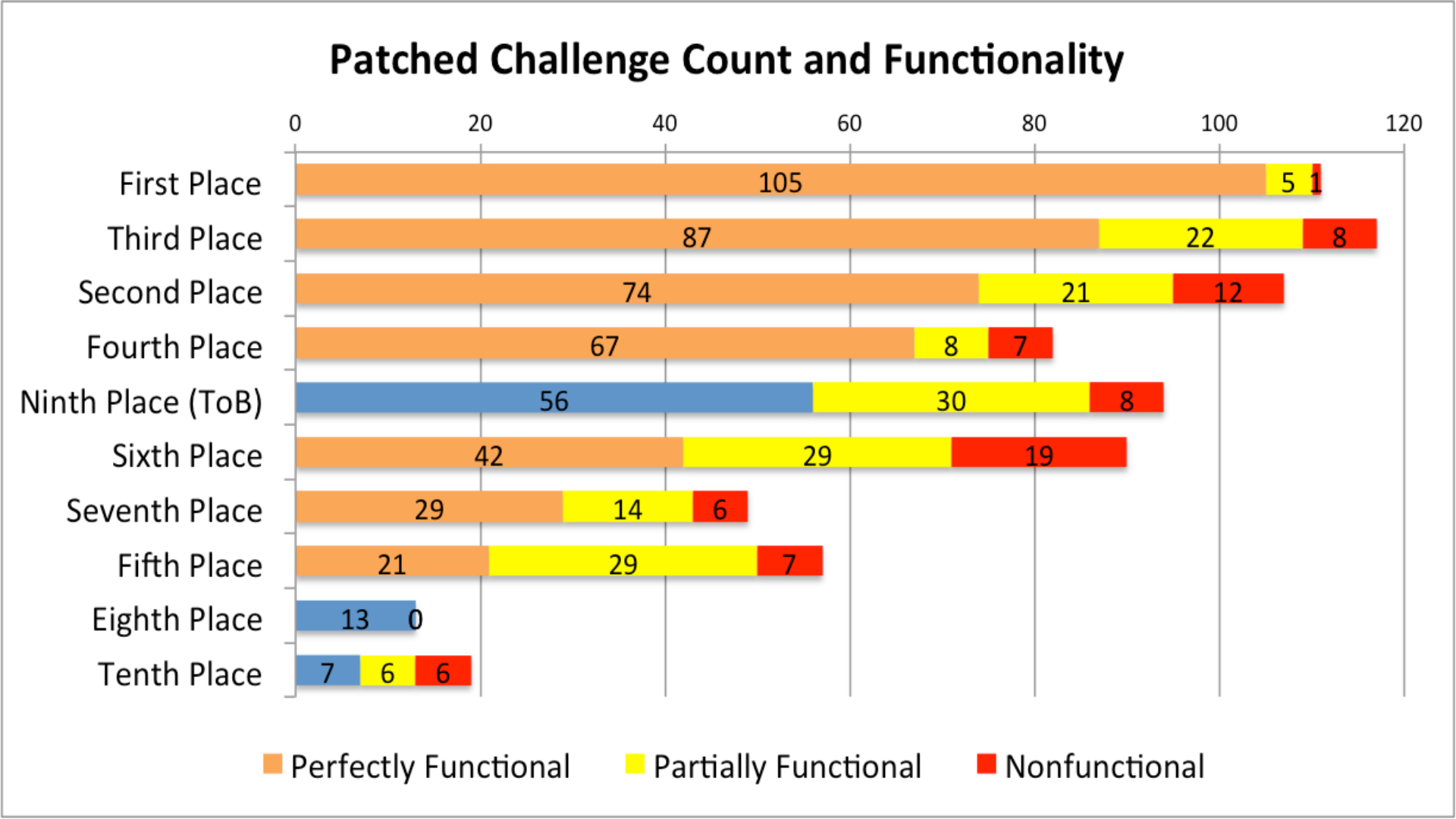
LLVM IR

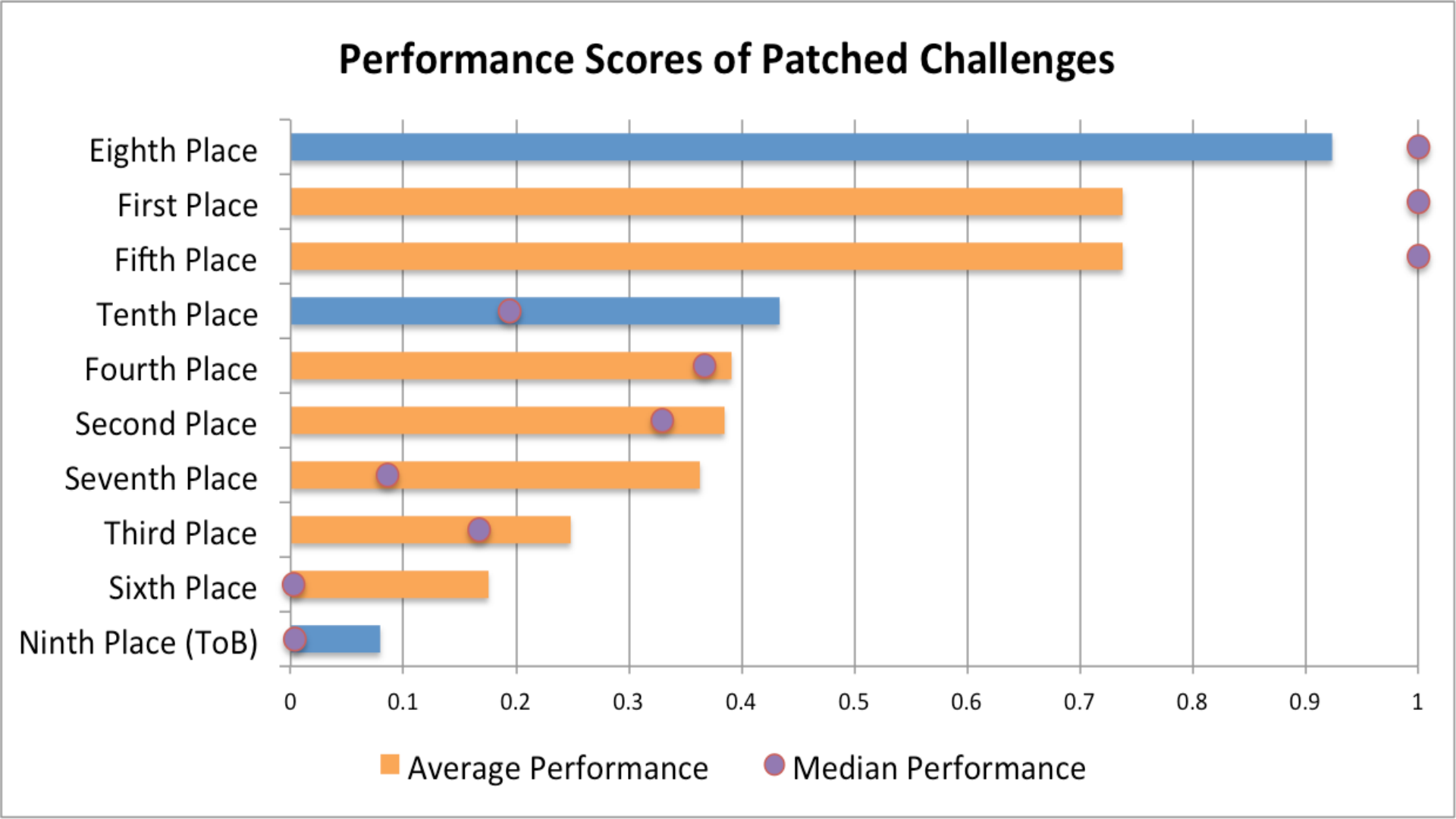
DEMO

QUESTIONS?


Number of Bugs Found
0
10
20
30
40
50
60
70
80
90
First Place
Ninth Place (ToB)
Third Place
Sixth Place
Seventh Place
Eighth Place
Fifth Place
Tenth Place
Second Place
Fourth Place


Patch Effectiveness (Security Score)
0
50
100
150
200
250
Third Place
Second Place
First Place
Ninth Place (ToB)
Sixth Place
Fourth Place
Fifth Place
Seventh Place
Tenth Place
Eighth Place


Patched Challenge Count and Functionality
0
20
40
60
80
100
120
First Place
105
5
1
Third Place
87
22
8
Second Place
74
21
12
Fourth Place
67
8
7
Ninth Place (ToB)
56
30
8
Sixth Place
42
29
19
Seventh Place
29
14
6
Fifth Place
21
29
7
Eighth Place
13
0
Tenth Place
7
6
6
Perfectly Functional
Partially Functional
Nonfunctional


Performance Scores of Patched Challenges
Eighth Place
First Place
Fifth Place
Tenth Place
Fourth Place
Second Place
Seventh Place
Third Place
Sixth Place
Ninth Place (ToB)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Average Performance
Median Performance