Automatically Identifying Exploitable Bugs

David Brumley
dbrumley@cmu.edu
Carnegie Mellon University

Joint work with:

Brent Hao

Thanassis Averginos

Sang Kil Cha

Ed Schwartz

Made possible with support in part from:

DARPA

Carnegie Mellon CyLab

NSF

LOCKHEED MARTIN

NORTHROP GRUMMAN

Evil David

Find Bugs

Exploit Systems

iwconfig: setuid wireless config

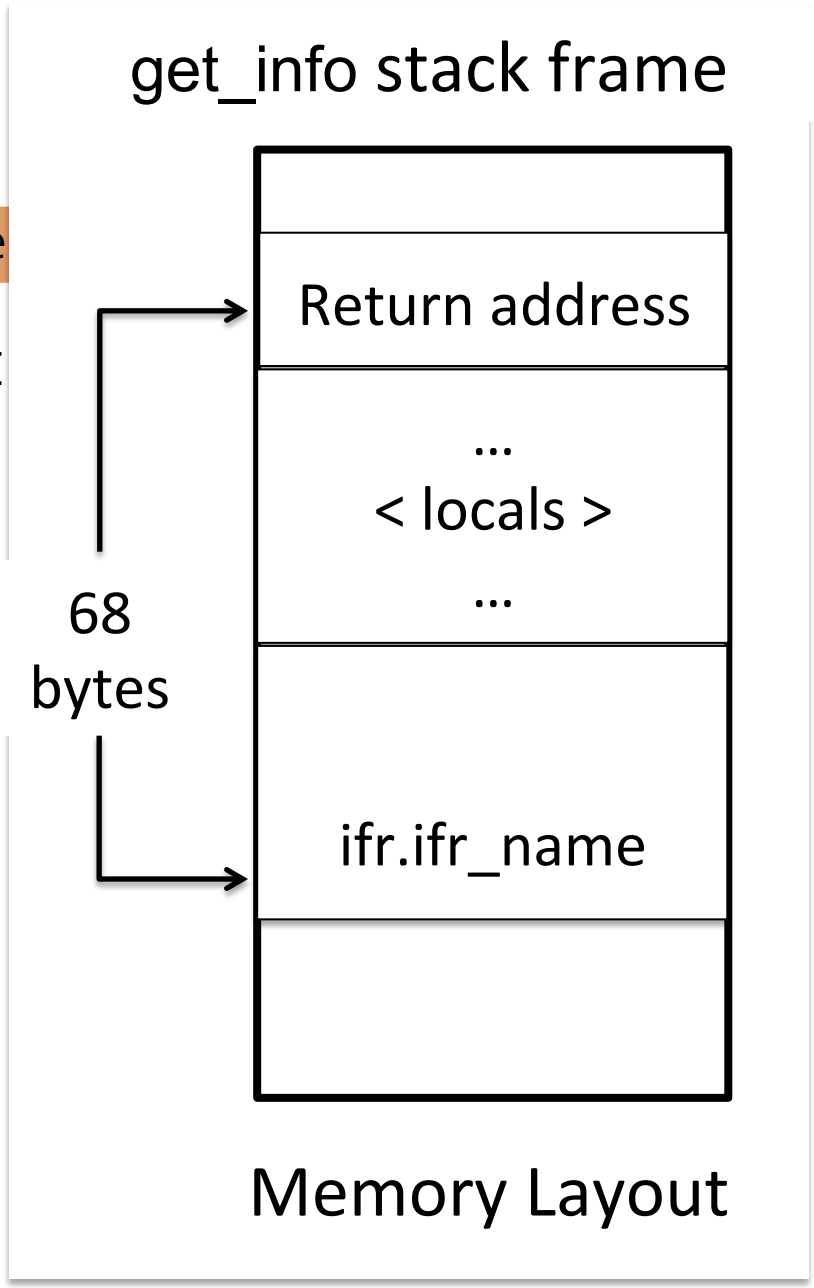
```
1 int get_info(int skfd, char * ifname, …){
2    ...
3    if(iw_get_ext(skfd, ifname, SIOCGIWNAME, &wrq) < 0)
4    {
5       struct ifreq ifr;
6       strcpy(ifr.ifr_name, ifname);
7 }

8 print_info(int skfd, char *ifname,…){
9   ...
10   get_info(skfd, ifname, …);
11 }

12 main(int argc, char *argv[]){
13 ...
14   print_info(skfd, argv[1], NULL, 0);
15 }
```

```
struct ifreq {
  char ifr_name[32]
  …
}
```

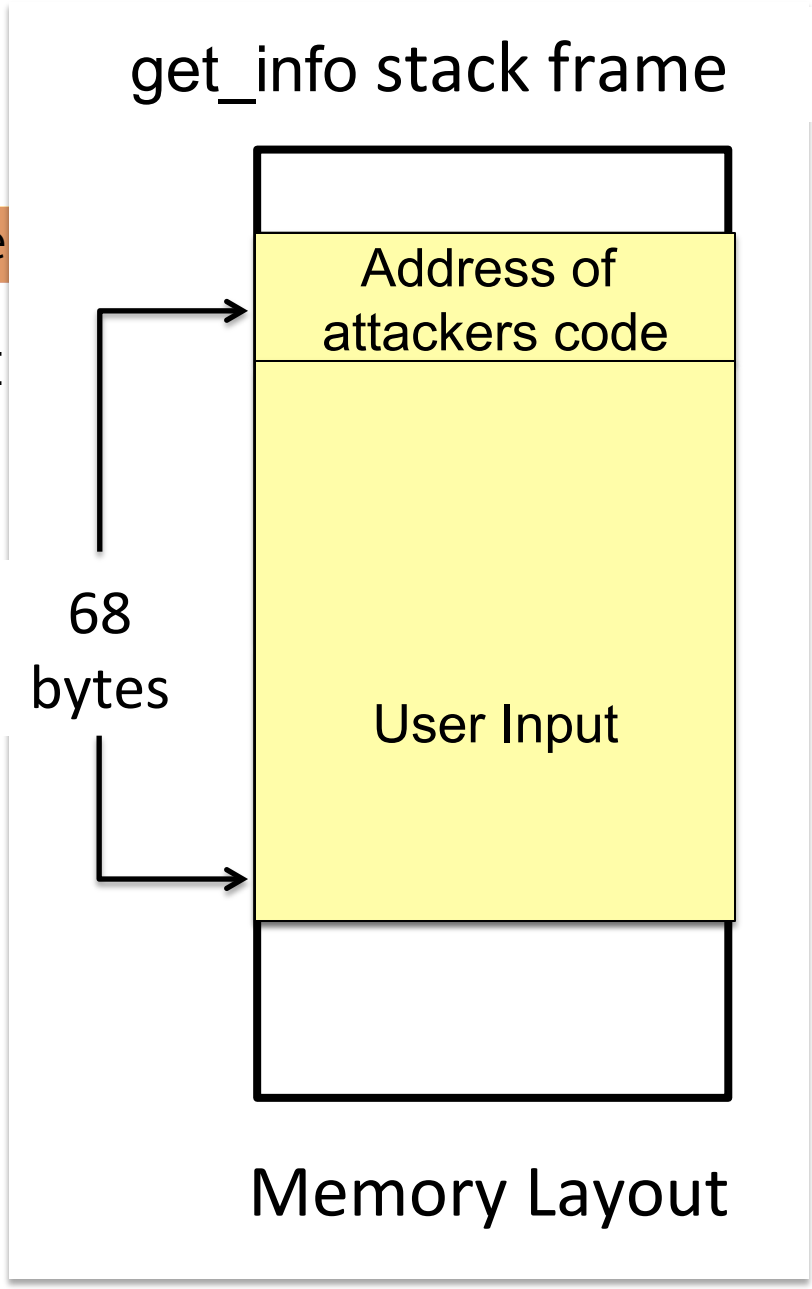
Can you spot the bug?

Is it exploitable?

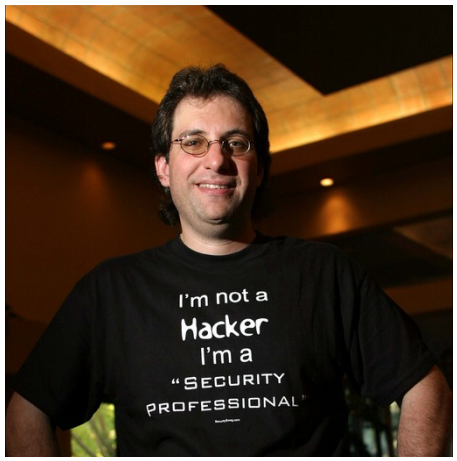
```
1 int get_info(int skfd, char * ifname
2    ...
3    if(iw_get_ext(skfd, ifname, SIOCGI
4    {
5       struct ifreq ifr;
6       strcpy(ifr.ifr_name, ifname);
7 }

8 print_info(int skfd, char *ifname,…)
9   ...
10   get_info(skfd, ifname, …);
11 }

12 main(int argc, char *argv[]){
13 ...
14   print_info(skfd, argv[1], NULL, 0)
15 }
```



```
1 int get_info(int skfd, char * ifname
2    ...
3    if(iw_get_ext(skfd, ifname, SIOCGI
4    {
5       struct ifreq ifr;
6       strcpy(ifr.ifr_name, ifname);
7 }

8 print_info(int skfd, char *ifname,…)
9   ...
10   get_info(skfd, ifname, …);
11 }

12 main(int argc, char *argv[]){
13 ...
14   print_info(skfd, argv[1], NULL, 0)
15 }
```



State-of-the-art in exploit generation



Kevin Mitnick

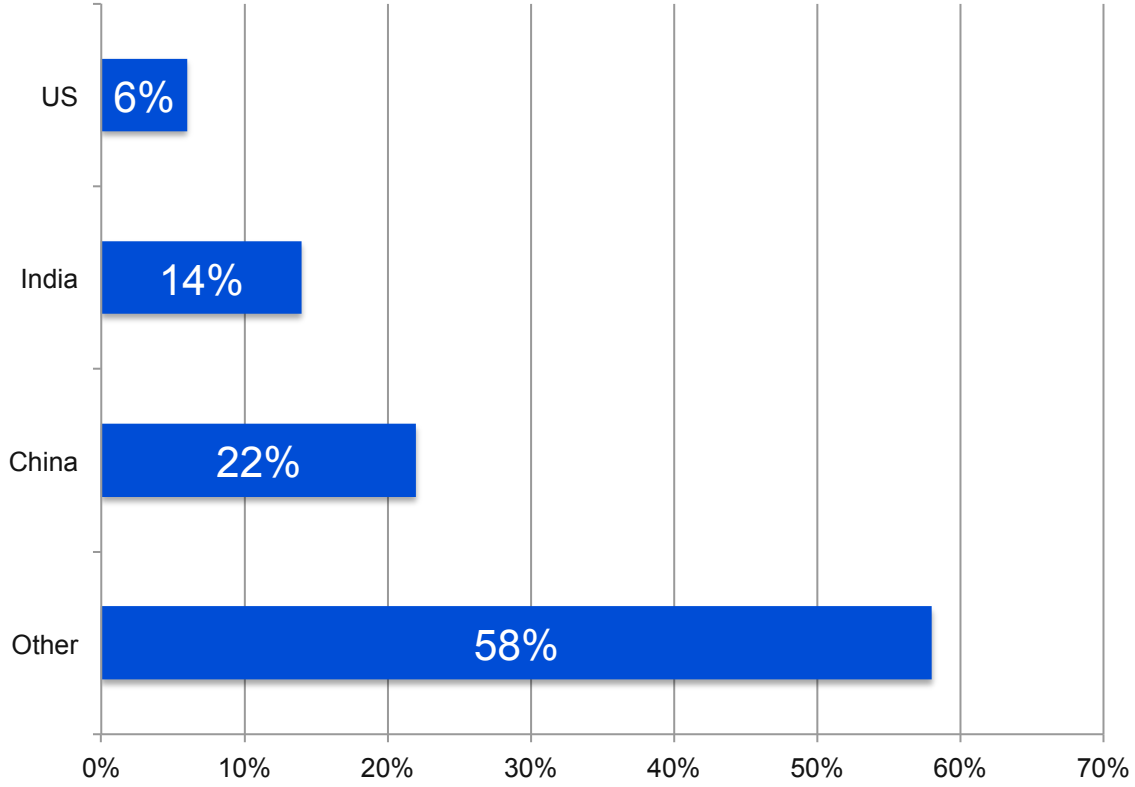
Robert Morris

Peiter Zatko

“Even as the U.S. government strengthens its cadre of cyber-security professionals, it must recognize that long-term trends in human capital do not bode well.”

-- William J. Lynn (D-SECDEF)

Offense



Percentage of World Population

Automatic Exploit Generation

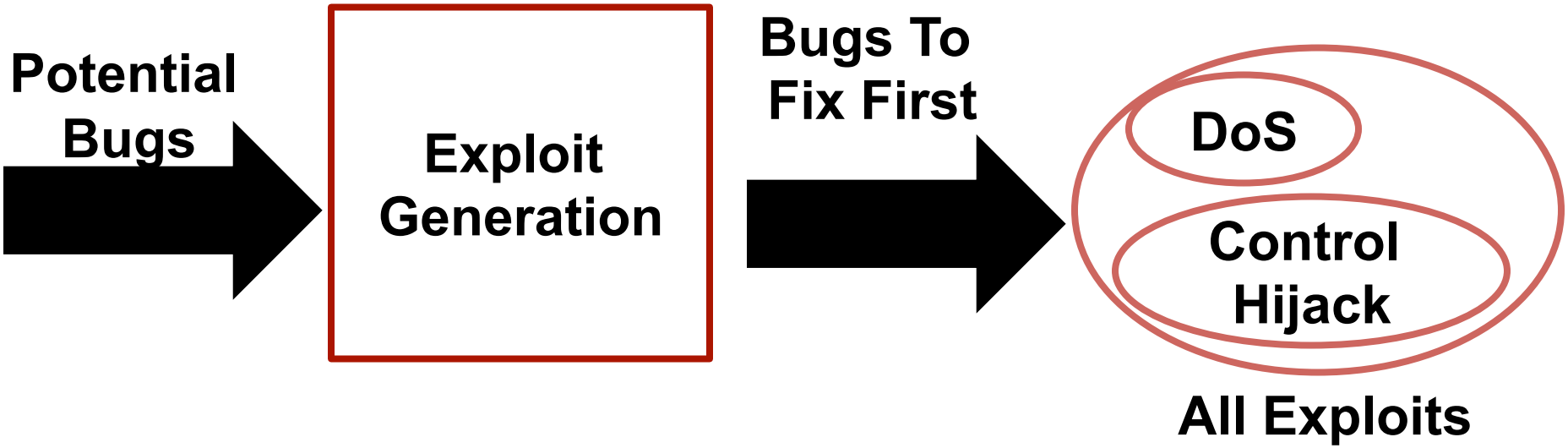
Given program, find bugs and demonstrate exploitability

Demo

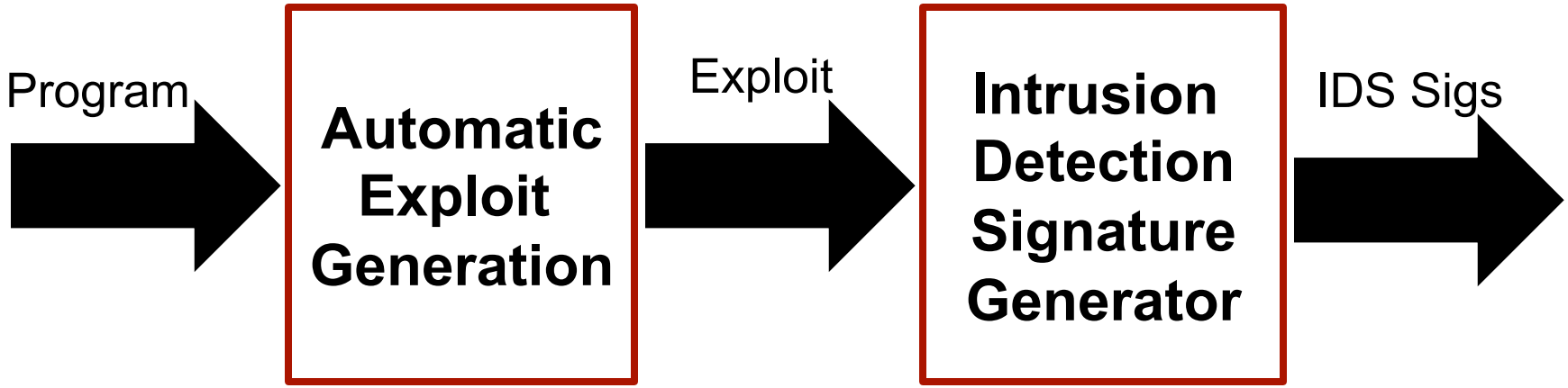
We owned the system in about 1 second

Evil David

Ubuntu has over 53,000 bugs to fix. Which one should be fixed first?



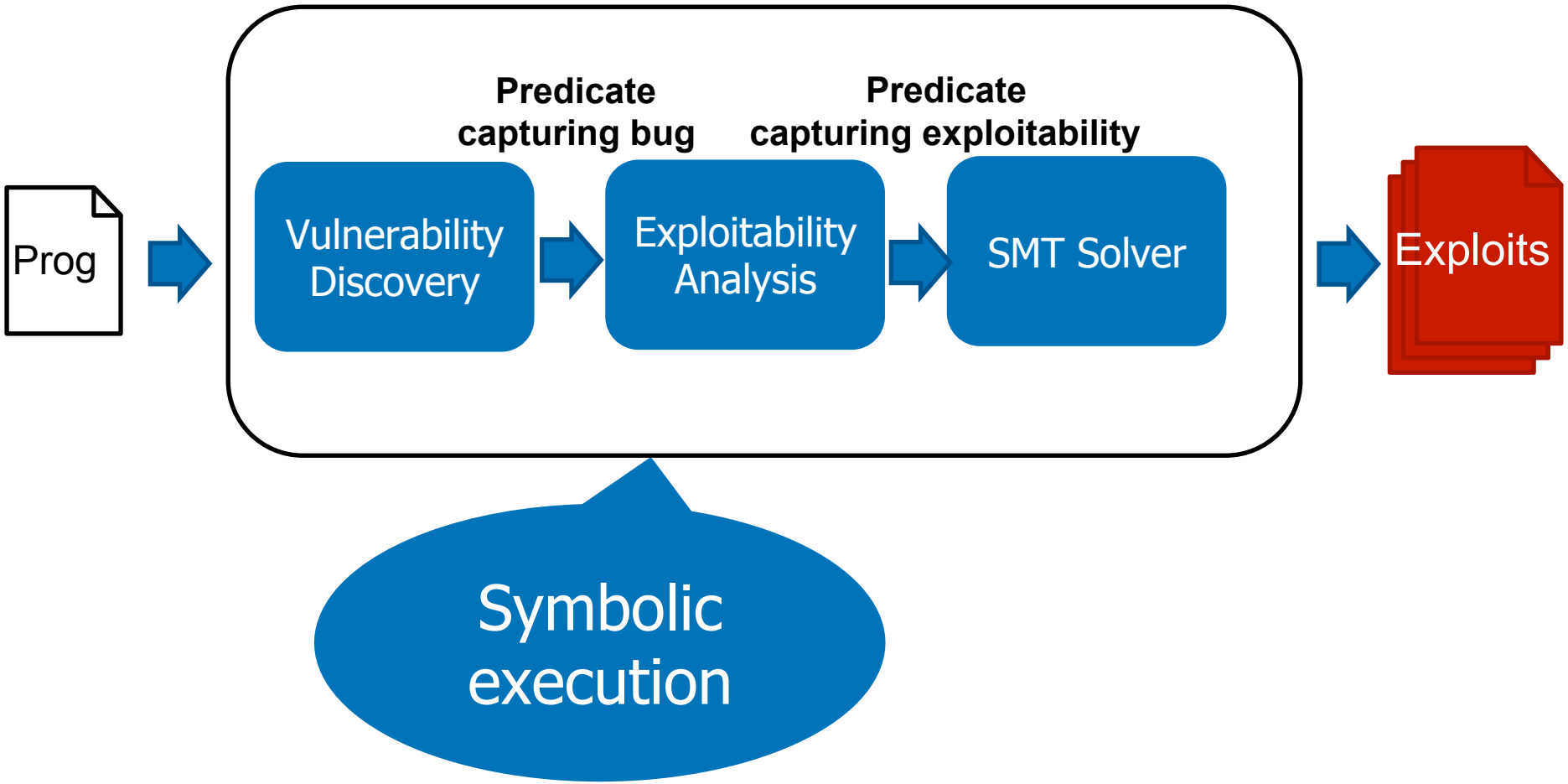
Get ahead of the attack curve



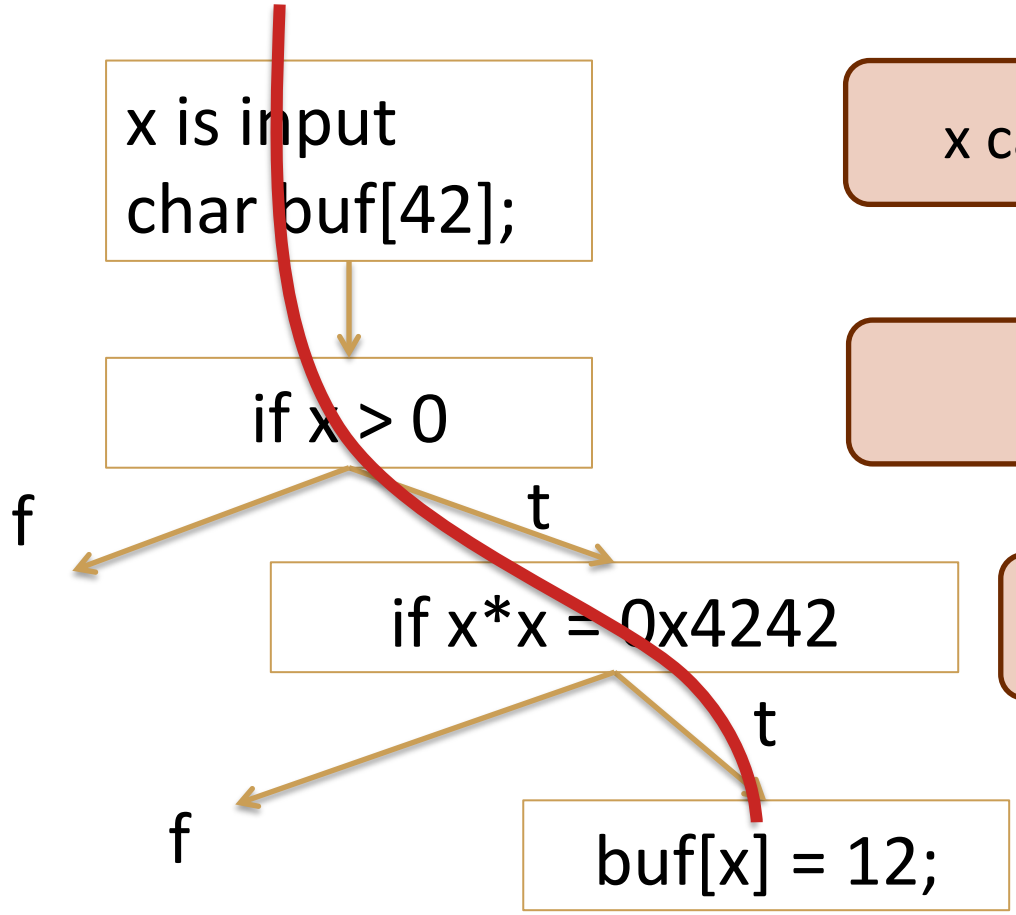
Applications within the DoD and Intelligence Communities

- ***Offense***: build better penetration testing tools
- ***Assurance***: Audit software for security vulnerabilities
- ***Defense***: Once we know what an exploit looks like, we can filter it in our networks
- ***Non-traditional domains***: Side applications to CPS and embedded systems security.

Automatic Exploit Generation: Approach Overview



Symbolic Execution: How it works



x can be anything

(x > 0)

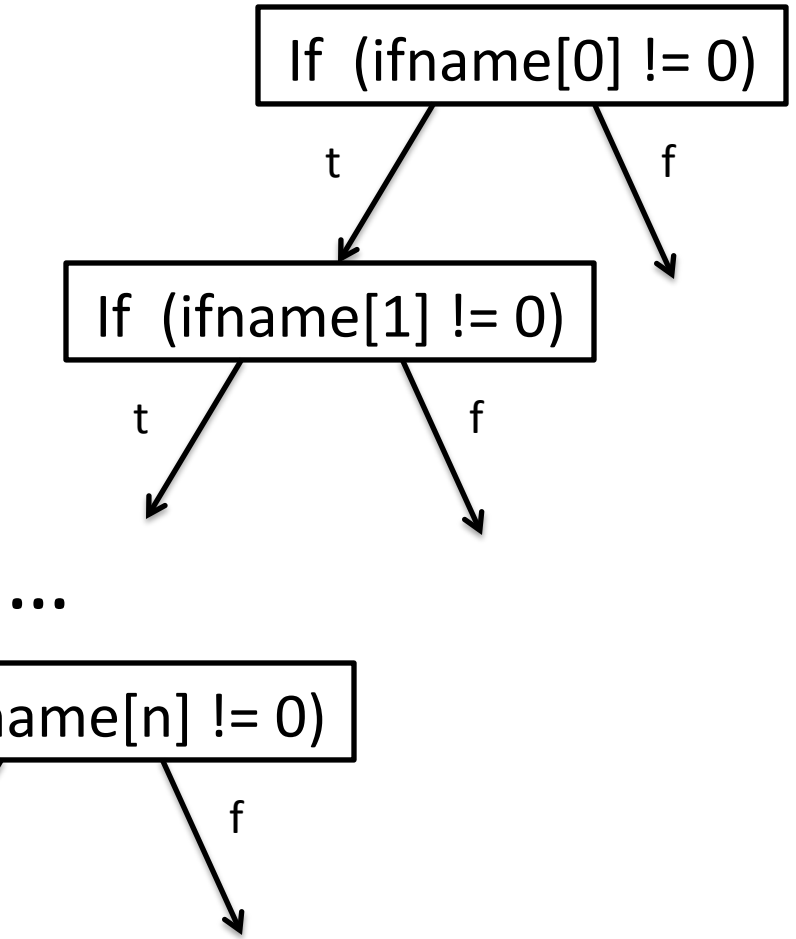
(x > 0) Λ (x*x = 0x4242)

Bug!
(x > 0) Λ (x*x = 0x4242)
Λ x > sizeof(buf)

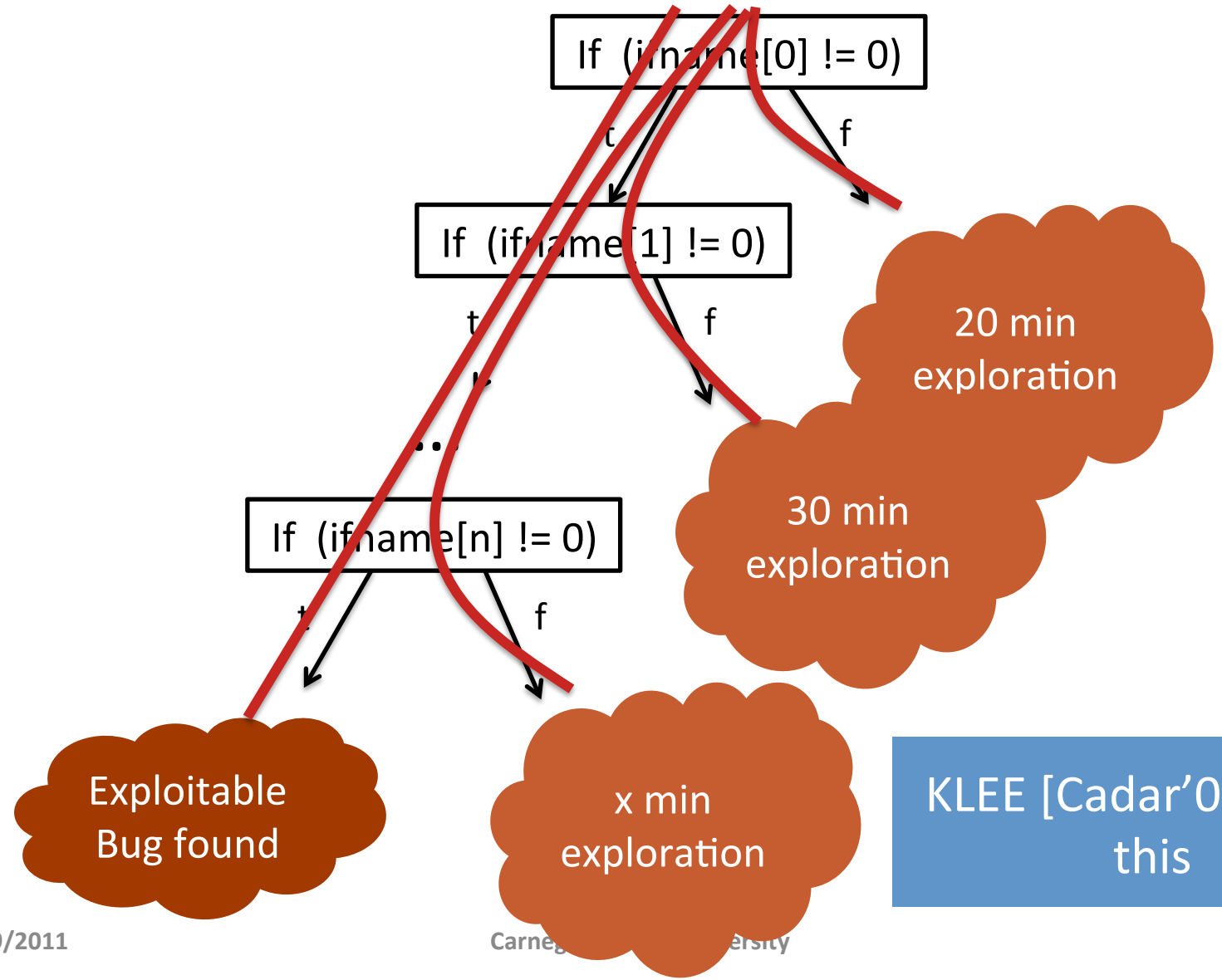
Symbolic Execution: Our Example

```
strcpy(ifr_name, ifname);
```

```
for (i = 0 ; ifname[i] != 0 ; i++)
    ifr_name[i] = ifname[i];
ifr_name[i] = 0;
```



Existing approaches on our example



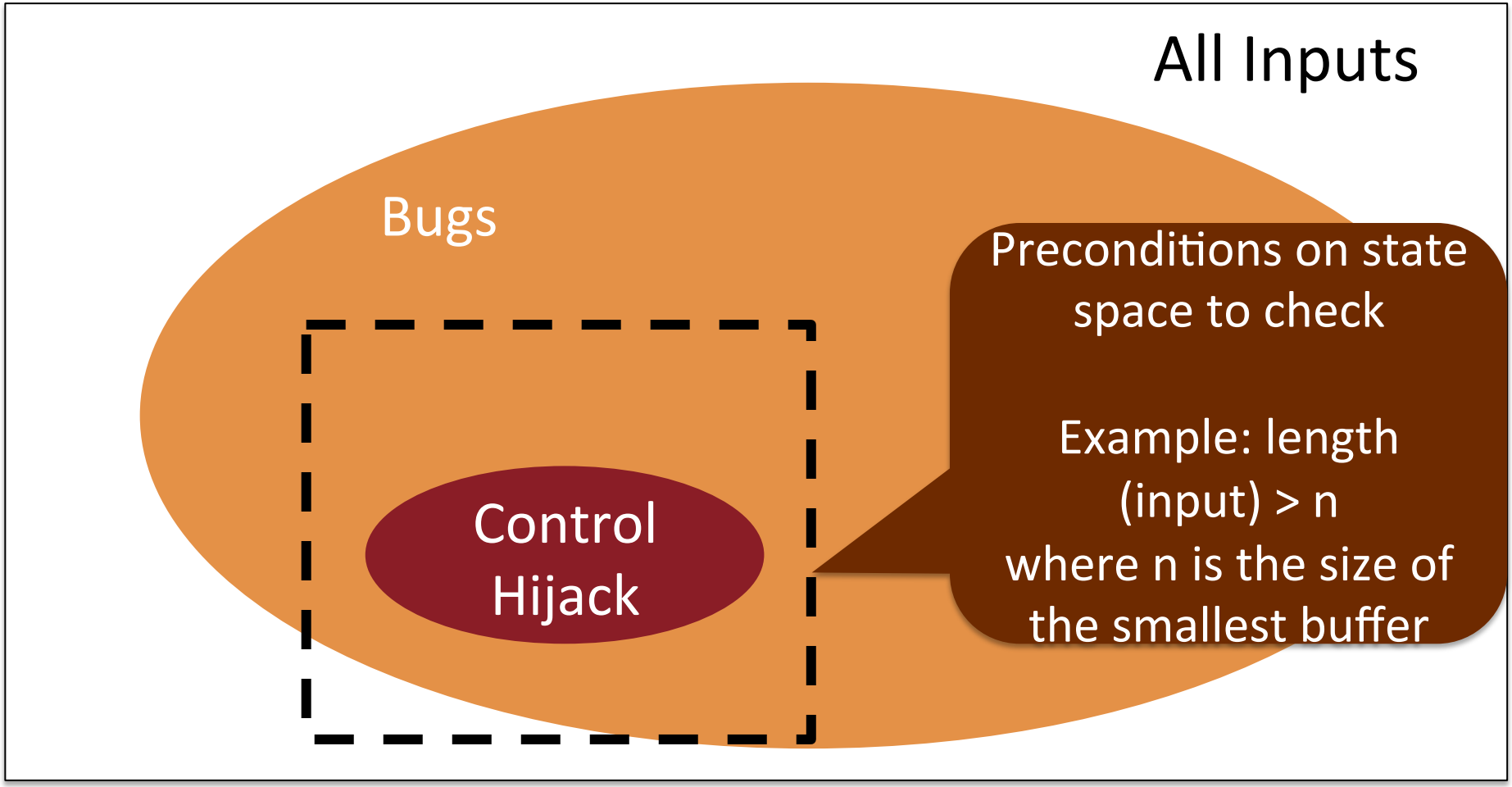
KLEE [Cadar'08] does this

Traditional symbolic execution: cover all paths (Slow to find exploitable bugs)

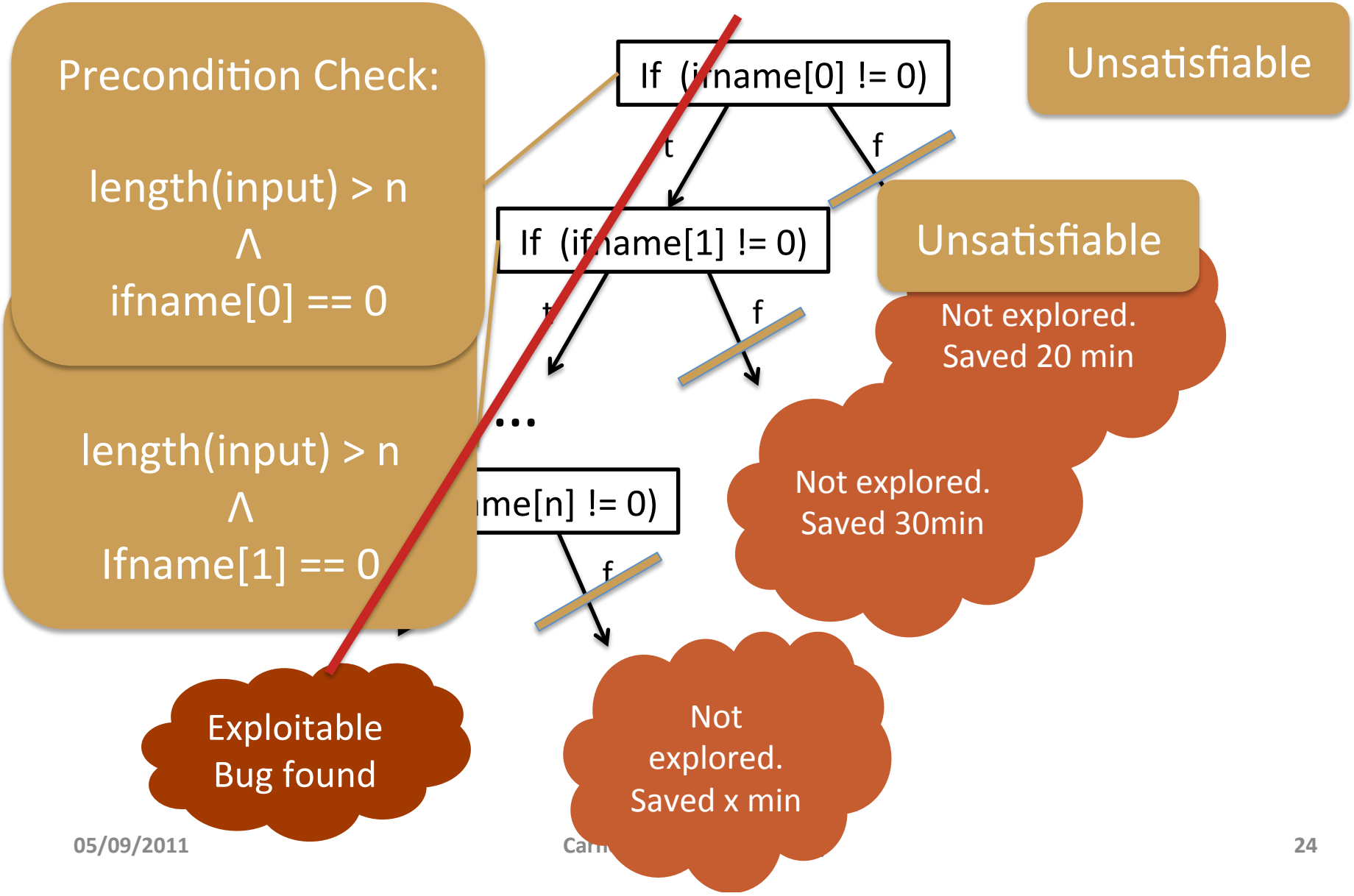
Traditional symbolic execution:
cover all paths
(Slow to find exploitable bugs)

Our Intuition for Exploit Generation:
only explore buggy paths (Fast)

Insight: ***Precondition Symbolic Execution***



AEG: Preconditioned Symbolic Execution



Preconditions

- Parameterized by static analysis
  - Size of the largest statically allocated buffer
  - Input Prefixes
  - Variable types
  - …

Given the bug, how to create an exploit?

Technique:
Add constraints to Hijack Control

Control Hijack for bug found:

length(input) > sizeof(ifr_name)

∧

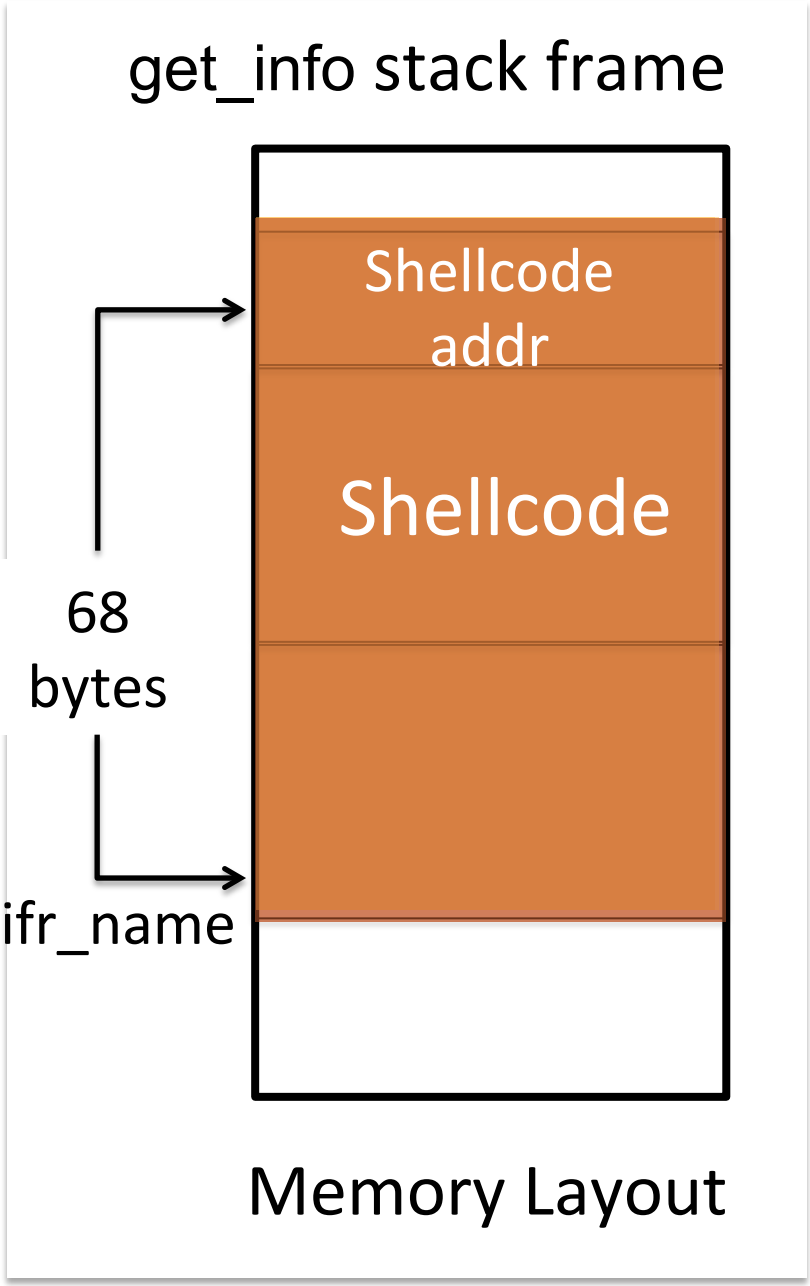
length(input) > 68 bytes

∧

input[0-63] == <shellcode>

∧

input[64-67] == <shellcode addr>



Generating Exploits

Control Hijack for bug found:

length(input) > sizeof(ifr_name)
∧
length(input) > 68 bytes
∧
input[0-63] == <shellcode>
∧
input[64-67] == <shellcode addr>

SMT Solver

Example:

```
02 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
01 01 01 01 70 f3 ff bf  31 c0 50 68 2f 2f 73 68
68 2f 62 69 6e 89 e3 50  53 89 e1 31 d2 b0 0b cd
80 01 01 01 00
```

More Challenges Addressed

- Optimize formulas: 10-100x faster results from commodity SMT solver
- Other preconditions and path prioritization heuristics
- Other attacks (format string, return-to-libc)
  - Reliability: e.g., nopsled etc
- Handling the "environment" problem
  - modelling system calls, library calls etc.

Published Results: AEG on Source+Binary in Linux

Name	Advisory ID	Time	Exploit Type	Exploit Class
Iwconfig	CVE-2003-0947	1.5s	Local	Buffer Overflow
Htget	CVE-2004-0852	< 1min	Local	Buffer Overflow
Htget	-	1.2s	Local	Buffer Overflow
Ncompress	CVE-2001-1413	12. 3s	Local	Buffer Overflow
Aeon	CVE-2005-1019	3.8s	Local	Buffer Overflow
Tipxd	OSVDB-ID#12346	1.5s	Local	Format String
Glftpd	OSVDB-ID#16373	2.3s	Local	Buffer Overflow
Xserver	CVE-2007-3957	31.9s	Remote	Buffer Overflow
Aspell	CVE-2004-0548	15.2s	Local	Buffer Overflow
Corehttp	CVE-2007-4060	< 1min	Remote	Buffer Overflow
Exim	EDB-ID#796	< 1min	Local	Buffer Overflow
Socat	CVE-2004-1484	3.2s	Local	Format String
Xmail	CVE-2005-2943	< 20min	Local	Buffer Overflow
Expect	OSVDB-ID#60979	< 4min	Local	Buffer Overflow
Expect	-	19.7s	Local	Buffer Overflow
Rsync	CVE-2004-2093	< 5min	Local	Buffer Overflow

What AEG is Not: Complete

- We do not claim to find all exploitable bugs
- Given an exploitable bug, we do not guarantee we will always find an exploit

But AEG is sound: if AEG outputs an exploit, the bug is guaranteed to be exploitable

What about DEP and ASLR?

What if the exploitdb exploit doesn't work on my system?

Evil David

Automatic Exploit Hardening

Turn exploit that works on undefended system to work against incomplete DEP and ASLR implementations

Program	Reference	Time	OS
Free CD to MP3	OSVDB-69116	130 sec	Win
FatPlayer	CVE-2009-4962	133 sec	Win
A-PDF Converter	OSVDB-67241	378 sec	Win
A-PDF Converter	OSVDB-68132	357 sec	Win
MP3 CD Converter	OSVDB-69951	158 sec	Win
Rsync	CVE-2004-2093	65 sec	Linux
Opendchub	CVE-2010-1147	125 sec	Linux
Gv	CVE-2004-1717	237 sec	Linux
Proftpd	CVE-2006-6563	40 sec	Linux

Future Directions

Applications

Program Analysis

Formal Methods

Exploit Theory

- Scalability
  - Efficient FSE
  - Faster SMT
- More types of vulnerabilities
  - Logic
  - Heap
- Marry exploit generation with hardening
- Integrate with fuzzing and other bug funding techniques
- Other Application Settings
  - CPS/Smart Grid

Thank you

dbrumley@cmu.edu
http://security.ece.cmu.edu/
http://bap.ece.cmu.edu