Automatically Identifying Exploitable Bugs

David Brumley
dbrumley@cmu.edu
Carnegie Mellon University
Joint work with:

Brent Hao  
Thanasssis Averginos  
Sang Kil Cha  
Ed Schwartz

Made possible with support in part from:

[Logos of DARPA, CyLab, NSF, Lockheed Martin, Northrop Grumman]
Evil David

Find Bugs

Exploit Systems
iwconfig: setupd wireless config

1 int get_info(int skfd, char *ifname, ...){
2   ...
3   if(iw_get_ext(skfd, ifname, SIOCGIWNNAME, &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 }

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

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, 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 }
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

- US: 6%
- India: 14%
- China: 22%
- Other: 58%
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

Program → Automatic Exploit Generation → Exploit → Intrusion Detection Signature Generator → IDS Sigs
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

1. **Vulnerability Discovery**
2. **Exploitability Analysis**
3. **SMT Solver**

- **Predicate capturing bug**
- **Predicate capturing exploitability**

Symbolic execution

Outputs: Exploits
Symbolic Execution: How it works

x is input
char buf[42];

if x > 0

if x*x = 0x4242

buf[x] = 12;

x can be anything

(x > 0)

(x > 0) \land (x*x = 0x4242)

Bug!

(x > 0) \land (x*x = 0x4242)
\land x > \text{sizeof(buf)}
Symbolic Execution: Our Example

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

```
If (ifname[0] != 0)
  t
  If (ifname[1] != 0)
    t
    If (ifname[n] != 0)
      t
      f
      f
    f
  f
...
Existing approaches on our example

If (iname[0] != 0)

If (iname[1] != 0)

If (iname[n] != 0)

Exploitable Bug found

20 min exploration

30 min exploration

x min exploration

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*

- **Bugs**
  - Control Hijack

- **All Inputs**
  - Preconditions on state space to check
  
  Example: length (input) > n where n is the size of the smallest buffer
AEG: Preconditioned Symbolic Execution

Precondition Check:

\[ \text{length(input)} > n \land \text{ifname[0]} == 0 \]

\[ \text{length(input)} > n \land \text{ifname[1]} == 0 \]

Exploitable Bug found

If (\text{ifname[0]} != 0)

\[ t \]

Not explored. Saved 20 min

If (\text{ifname[1]} != 0)

\[ t \]

Not explored. Saved 30 min

Unsatisfiable

Unsatisfiable

Not explored. Saved x min

05/09/2011
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:

\[
\text{length(input)} > \text{sizeof(ifr\_name)} \\
\wedge \\
\text{length(input)} > 68 \text{ bytes} \\
\wedge \\
\text{input[0-63]} == \text{<shellcode>} \\
\wedge \\
\text{input[64-67]} == \text{<shellcode addr>}
\]
Generating Exploits

Control Hijack for bug found:

\[
\text{length(input) > sizeof(ifr_name)} \\
\land \\
\text{length(input) > 68 bytes} \\
\land \\
\text{input[0-63] == <shellcode>} \\
\land \\
\text{input[64-67] == <shellcode addr>}
\]

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 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Advisory ID</th>
<th>Time</th>
<th>Exploit Type</th>
<th>Exploit Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iwconfig</td>
<td>CVE-2003-0947</td>
<td>1.5s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Htget</td>
<td>CVE-2004-0852</td>
<td>&lt; 1min</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Htget</td>
<td></td>
<td>1.2s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Ncompress</td>
<td>CVE-2001-1413</td>
<td>12.3s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Aeon</td>
<td>CVE-2005-1019</td>
<td>3.8s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Tipxd</td>
<td>OSVDB-ID#12346</td>
<td>1.5s</td>
<td>Local</td>
<td>Format String</td>
</tr>
<tr>
<td>Glftpd</td>
<td>OSVDB-ID#16373</td>
<td>2.3s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Xserver</td>
<td>CVE-2007-3957</td>
<td>31.9s</td>
<td>Remote</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Aspell</td>
<td>CVE-2004-0548</td>
<td>15.2s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Corehttp</td>
<td>CVE-2007-4060</td>
<td>&lt; 1min</td>
<td>Remote</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Exim</td>
<td>EDB-ID#796</td>
<td>&lt; 1min</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Socat</td>
<td>CVE-2004-1484</td>
<td>3.2s</td>
<td>Local</td>
<td>Format String</td>
</tr>
<tr>
<td>Xmail</td>
<td>CVE-2005-2943</td>
<td>&lt; 20min</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Expect</td>
<td>OSVDB-ID#60979</td>
<td>&lt; 4min</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Expect</td>
<td></td>
<td>19.7s</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>Rsync</td>
<td>CVE-2004-2093</td>
<td>&lt; 5min</td>
<td>Local</td>
<td>Buffer Overflow</td>
</tr>
</tbody>
</table>
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

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

Applications

• Scalability
  – Efficient FSE
  – Faster SMT

• More types of vulnerabilities
  – Logic
  – Heap

• Marry exploit generation with hardening

Program Analysis

• Integrate with fuzzing and other bug funding techniques

Formal Methods

• Other Application Settings
  – CPS/Smart Grid

Exploit Theory
Thank you
dbrumley@cmu.edu
http://security.ece.cmu.edu/
http://bap.ece.cmu.edu