Checking the World’s Software

David Brumley
Carnegie Mellon University
dbrumley@cmu.edu
http://security.ece.cmu.edu/
An epic battle

White Hat vs. Black Hat
Exploit bugs

Bug

White Hat

Black Hat
Bug-Fixed!

White Hat

Black Hat
1. inp=`perl -e '{print "A"x8000}'`
2. for program in /usr/bin/*; do
3.   for opt in {a..z} {A..Z}; do
4.     timeout -s 9 1s
        $program -$opt $inp
3.   done
4. done

1009 Linux programs. 13 minutes.
52 new bugs in 29 programs.
Fact:
Windows, Mac, and Linux all have 100,000’s of bugs
Which bugs are exploitable?
Highly Trained Experts
CMU creates experts
<table>
<thead>
<tr>
<th>Place</th>
<th>Team</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plaid Parliament of Pwning</td>
<td>1703.091</td>
</tr>
<tr>
<td>2</td>
<td>More Smoked Leet Chicken</td>
<td>1411.414</td>
</tr>
<tr>
<td>3</td>
<td>Dragon Sector</td>
<td>939.359</td>
</tr>
<tr>
<td>4</td>
<td>Eindbazen</td>
<td>939.359</td>
</tr>
<tr>
<td>5</td>
<td>GoN</td>
<td>747.866</td>
</tr>
<tr>
<td>6</td>
<td>blue-lotus</td>
<td>729.386</td>
</tr>
<tr>
<td>7</td>
<td>HackingForSoju</td>
<td>725.533</td>
</tr>
<tr>
<td>8</td>
<td>ufologists</td>
<td>672.858</td>
</tr>
<tr>
<td>9</td>
<td>CLGT</td>
<td>610.626</td>
</tr>
<tr>
<td>10</td>
<td>The Cat is #1!!</td>
<td>589.327</td>
</tr>
</tbody>
</table>
DEFCON 2013

<table>
<thead>
<tr>
<th>Team</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPP</td>
<td>15002</td>
</tr>
<tr>
<td>men in black hats</td>
<td>7924</td>
</tr>
<tr>
<td>raon_ASRT</td>
<td>7107</td>
</tr>
<tr>
<td>more smoked leet chicken</td>
<td>4160</td>
</tr>
<tr>
<td>routards</td>
<td>2503</td>
</tr>
<tr>
<td>sutegoma2</td>
<td>1540</td>
</tr>
<tr>
<td>shellphish</td>
<td>1223</td>
</tr>
<tr>
<td>Alternatives</td>
<td>1095</td>
</tr>
<tr>
<td>The European Nopsled Team</td>
<td>859</td>
</tr>
<tr>
<td>9447</td>
<td>506</td>
</tr>
<tr>
<td>blue lotus</td>
<td>441</td>
</tr>
<tr>
<td>Samurai</td>
<td>12</td>
</tr>
<tr>
<td>APT8</td>
<td>0</td>
</tr>
<tr>
<td>clgt</td>
<td>0</td>
</tr>
<tr>
<td>pwnies</td>
<td>0</td>
</tr>
<tr>
<td>pwningyeti</td>
<td>0</td>
</tr>
<tr>
<td>Robot Mafia</td>
<td>0</td>
</tr>
<tr>
<td>shell corp</td>
<td>0</td>
</tr>
<tr>
<td>[Technopandas]</td>
<td>0</td>
</tr>
<tr>
<td>WOWHacker-BIOS</td>
<td>0</td>
</tr>
</tbody>
</table>

DEFCON 2014

Final Scores

<table>
<thead>
<tr>
<th>Team</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaid Parliament of Pwning</td>
<td>11263</td>
</tr>
<tr>
<td>HITCON</td>
<td>7833</td>
</tr>
<tr>
<td>Dragon Sector</td>
<td>4421</td>
</tr>
<tr>
<td>Reckless Abandon</td>
<td>4020</td>
</tr>
<tr>
<td>blue-lotus</td>
<td>3233</td>
</tr>
<tr>
<td>(Mostly) Men in Black Hats</td>
<td>2594</td>
</tr>
<tr>
<td>raon_ASRT</td>
<td>2281</td>
</tr>
<tr>
<td>StratumAuahuur</td>
<td>1529</td>
</tr>
<tr>
<td>[CBA]9447</td>
<td>1519</td>
</tr>
<tr>
<td>KAIST GoN</td>
<td>1334</td>
</tr>
</tbody>
</table>
The US is lagging.
<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plaid Parliament of Pwning</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>More Smoked Leet Chicken</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Dragon Sector</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Eindbazen</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>dcua</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>CCCAC</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>w3stormz</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Samurai</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>LSE</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>CodeRed</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Stratum0</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>disekt</td>
<td></td>
</tr>
</tbody>
</table>
picoCTF: Teaching 10,000 High School Students to Hack
PRESENTING

TOASTER WARS

HIGH SCHOOL HACKING COMPETITION

APRIL 26TH 2013 – MAY 6TH 2013

Sponsorship provided by the NSA.

TOASTER WARS

When a robot from space crashes in your backyard it's up to you to uncover the secret he carries...

The Adventure Will Run: April 26th 2013 - May 6th 2013. Registration is open now!

CREATED BY

Toaster Wars is a collaboration of the Paid Parliament of Pwning (PPP) of CyLab and Osiris of the Entertainment Technology Center.

Both teams are student-run and based at Carnegie Mellon University.
House Story/Problem Design

Step 1:
- Booting and Auto Debug
  - Enter Debug Mode
  - FAT/IF Statement

Step 2:
- Robot Sending Message to Base
- Network/WireShark
- File Structure
- Rename/Toast
- Secret Competition
- CMD
- Reveal Robot History

Other:
- Sleep in the bed
- Learn some interesting facts

Legend:
- Story
- Problem
- Event Info (Story)

After the first problem solved, the robot will begin to auto-debug. Solving 2/4 of the following problems will result in the access to the next level, which two other information will be shown due to debug-completion.
~10,000 each year

~2000 teams

57 problems

> 12 hours working/average

$25k scholarships

*Largest* cyber exercise ever
Will you encourage your students to compete next year?

Yes

No
Impact beyond high school

Promote awareness
1. Cisco
2. Project lead the way
3. HS-CTF
4. BoilerQuest
5. ACTF
6. BCACTF
7. ...

Foundation for Cyber
1. US Naval Academy
2. US Air Force Academy
3. US Military Academy
4. US Coast Guard Academy
Which bugs are *exploitable*?

even the best people are not enough.....
DEF CON 2012 scoreboard

Time (3 days total)
Does not scale
I skate to where the puck is *going to be*, not where it has been.

--- Wayne Gretzky
Hockey Hall of Fame
The Vision: 
Automatically Check the World’s Software for Exploitable Bugs
Checking the world's software

Program Analysis

Safe

Buggy

Exploitable
We owned the machine in seconds
Symbolic Execution

[Boyer75, Howden75, King76]

Program → Verification

Correctness Property

Un-exploitability Property

Correct Safe path
Incorrect Exploitable

Verification, but with a twist
Basic symbolic execution

```python
if x < 42
if x > 42
```

```
x = input()
```

```
if x > 42
```

```
if x * x = MAXINT
```

```
if x < 42
jmp stack[x]
```

```
x can be anything
```

```
x > 42
```

```
(x > 42)
∧ (x * x != MAXINT)
```

```
(x > 42)
∧ (x * x != MAXINT)
∧ !(x < 42)
```
if $x < 42$

if $x > 42$

if $x < 42$

jmp stack[$x$]

x = input()

Path formula

(true for inputs that take path)

x can be anything

$x > 42$

$(x > 42) \land (x \times x \neq \text{MAXINT})$

$(x > 42) \land (x \times x \neq \text{MAXINT}) \land \neg (x < 42)$
Basic symbolic execution

```
x = input()
if x > 42
    if x*x = MAXINT
        x = 43
    else
        jmp stack[x]
else
    if x < 42
        jmp stack[x]
```

Satisfiable

\( x = 43 \)

SMT Solver

satisfiability modulo theories

\((x > 42) \land (x^2 \neq \text{MAXINT}) \land \lnot (x < 42)\)
Control flow hijack

*attacker gains control of execution*

- buffer overflow
- format string attack
- heap metadata overwrite
- use-after-free
- ...

Same principle, different mechanism
Basic execution semantics of compiled code

Process Memory

- Code
- Data
- Stack
- Heap

Fetch, decode, execute

Instruction Pointer points to next instruction to execute

Processor

EIP

read and write

Control Flow Hijack: EIP = Attacker Code
Checking non-exploitability

Un-exploitability property: 
\( EIP \neq \text{user input} \)

\[
x = \text{input()}
\]

\[
\text{if } x > 42 \\
\text{if } x \times x = \text{MAXINT} \\
\text{if } x < 42 \\
\text{jmp stack}[x]
\]

Exploit
Safe

SMT

\[(x > 42) \\
\land (x \times x = \text{MAXINT}) \\
\land \ldots \land \ EIP \neq \text{input}\]
checking Debian for exploitables bugs

37,391 programs

3 years CPU time

16 billion verification queries

~$0.28/confirmed bug

~$21/exploit

2,606,506 crashes

13,875 unique bugs

152 new hijack exploits

* [ARCB, ICSE 2014, ACM Distinguished Paper], [ACRSWB, CACM 2014]
mining data

Q: How long do queries take on average?
A: 3.67ms on average with 0.34 variance

Q: Should I optimize hard or easy formulae?
A: 99.99% take less than 1 second and account for 78% of total time

Q: Do queries get harder?
A: Good question...
solve time vs. $\log (#\text{AST node})$

Size not strongly correlated with hardness
Hard (time >= 1.0 sec) formulae at greater depths are usually larger but not necessarily harder to solve.

- 500 sec timeout
- No clear upward trend
Only 39 programs create hard formulas

\[
a/10 \text{ replaced with } (a \times \text{0xcccccccccd}) \gg 3
\]
Symbolic execution is not perfect. It won’t find all problems.

But each report is actionable.
Our Vision: Automatically Check the World’s Software for Exploitable Bugs
A General (or professor) walks into a cramped cubicle, telling the lone security analyst (or graduate student) that she has one week to find a zero-day exploit against a certain popular OS distribution, all the while making it sound as if this task is as easy as catching the next bus. Although our analyst has access to several program analysis tools for finding bugs \([8, 10, 11, 21]\) and generating exploits \([4, 9]\), she still faces a harsh reality: the target OS distribution contains thousands of programs, each with potentially tens or even hundreds of yet undiscovered bugs. What tools should she use for this mission? Which programs should she analyze, and in what order? How much time should she dedicate to a given program? Above all, how can she maximize her likelihood of success within the given time budget?
Binary Programs

Billions of programs

Program Analysis

AEG  Fuzz  ...

Potentially many analyses

Safe

Buggy

Exploitable
An *analysis campaign* comprises a sequence of *epochs*:
1. Takes a list of (program, analysis) pairs as input.
2. At the beginning of each epoch, picks one (program, analysis) pair to run based on data collected from previous epochs.

**Goal:** Maximize the number of unique bugs found.

Campaign time
Explore vs. Exploit

Two *competing goals* during a fuzz campaign:

**Explore** each \((p_i, s_i)\) sufficiently often so as to identify pairs that can yield new bugs

**Exploit** knowledge of \((p_i, s_i)\) that are likely to yield new bugs by fuzzing them more

**Good News:**
- Clearly a *Multi-Armed Bandit (MAB)* problem!

**Research Opportunities:**
- Not a traditional MAB. Needs new algorithms
Multi-Armed Bandits
Theoretical foundation

Evaluated 26 new and existing randomized scheduling algorithms wrt fuzzing

48,000 hours of fuzzing to get

1.55x more bugs than current industry best (CERT BFF)

MAXIMIZE SECURITY ROI
Research

This talk:

• CMU security program
• Grow the security community
• Automatic exploit generation
• Maximizing ROI
• and there is always more...

Move beyond publications
Tools for:

- **Developers** to check their final builds (a best practice!)
- **Businesses**, IT depts, and compliance testers to check software they use, but didn’t develop
- **Everyone** to assess the weighted security score of that new app they see, as well as all apps installed on their system
The Vision: 
*Automatically* 
Check the World’s Software for *Exploitable* Bugs

It seems *wrong* to not try.
Thank You!

Questions?
END