Vulnerability Discovery
Dr. Edward J. Schwartz
eschwartz@cert.org

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Vulnerability Discovery Project

Increase **assurance** of 1st and 3rd party DoD software through **enhanced** vulnerability discovery techniques
Team

Software Engineering Institute

- Edward Schwartz, PhD, CERT
- David Warren, CERT
- Allen Householder, CERT

Collaborators

∀.Secure

- David Brumley, PhD
- Thanassis Avgerinos, PhD
- Tyler Nighswander
Towards vulnerability discovery as a science

Intelligent fusion of vulnerability discovery techniques
Towards Vulnerability Discovery as a Science
Background: Mutational Fuzzing of Software

Testing of programs by randomly mutating program inputs (seeds)
Challenge: How Many Software Vulnerabilities are There?
Problem: Distinguishing One Vulnerability From Another

I don’t know how to specify a vulnerability, but I know how to fix one
The Idea: Patches Define Vulnerabilities

Any crash that is fixed by the patch is also affected by vulnerability V
Example Ground Truth

\[ P^1 \]

\[ P^2 \]
Patching ImageMagick

Fuzzed old ImageMagick with the CERT BFF fuzzer

- 1 week
- 130,000 crashes found

Manually patched all vulnerabilities

- Took approximately one month
- 31 patches/vulnerabilities
Vulnerability Discovery Science

Analyze fuzzing parameters

• What mutators work best?
• When should we stop fuzzing?
• What effect do compiler settings have?
• …
• Paper submitted to NDSS 2016
Background: Mutational Fuzzing of Software

How should we mutate the seed?
Fuzzing Mutators
When to Stop Fuzzing?
When to Stop Fuzzing?
Collaboration with ForAllSecure
CMU Spinoff of David Brumley’s research group

Dr. David Brumley  
Dr. Thanassis Avgerinos  
Alex Rebert
Expertise

- Concolic execution
- Automatic exploitation
- Binary analysis
- Complements SEI’s expertise in fuzzing

Previous collaboration

- With same group at CMU
Background: Concolic Execution

```c
rows, cols = input()
if rows > 0 && cols > 0
    p = malloc(rows*cols*4)
    p[rows*cols-1] = 0xFF
    exit

(rows > 0 && cols > 0)  
rows=5  
cols=10
```
Background: Concolic Execution

\[
\begin{align*}
\text{rows, cols} &= \text{input()} \\
\text{if } \text{rows} > 0 \&\& \text{cols} > 0 \Rightarrow \\
\text{p} &= \text{malloc(rows*cols*4)} \\
\text{p}[\text{rows*cols-1}] &= 0xFF \\
\end{align*}
\]
Background: Fuzzing

```c
rows, cols = input()
if rows > 0 && cols > 0
    p = malloc(rows*cols*4)
    rows = 0x00004014
    cols = 0x00000034
    p[rows*cols-1] = 0xFF
    exit
```

Most mutations do not trigger the overflow vulnerability

Overflow?

0xd040f >= 0x341040

No

(rows=0x00000014, cols=0x00000014)
Concolic Execution vs. Fuzzing

Concolic Execution

- Precise
- Slow
- Hard constraints

Fuzzing

- Blunt
- Fast
- Easy constraints
The Synergistic Mayhem AFL Research Tool

- Concolic execution: Mayhem (ForAllSecure+SEI)
- Fuzzing: AFL
- Periodically synchronize seed files between them

Challenges

1. Where to go?
   - We don’t know the location of vulnerabilities
2. How much should we use concolic execution?
   - $\sim 10^4$ times slower than fuzzing
   - Brute force vs. high cost
SMART Evaluation

Edge Coverage After Two Days with Blank Seeds

- readelf
- objdump
- convert-png
- mplayer-mp3
- pdf2svg-pdf

Mayhem  SMART
Summary

- Developing new techniques for discovering and mitigating vulnerabilities in the DoD
- Developed vulnerability uniqueness model and used ground truth to explore the effect of fuzzing parameters
- ForAllSecure: Hybrid fuzzing and concolic tester

Team Members

- Edward Schwartz, PhD, CERT
- David Warren, CERT
- Allen Householder, CERT

ForAllSecure, Inc.:

- David Brumley, PhD
- Thanassis Avgerinos, PhD
- Tyler Nighswander
Compiler Flags and Settings

Changing compilers and optimizations

Address sanitizer
The Crash Uniqueness Problem

100 M Tests → 100 K Crashes → 100 Vuls
The State of the Art: Stack Hashing

Hash( ) = b1946ac9 2492d234 7c6235b4 d2611184

Crash

malloc
SGIDecode
ReadSGIImage
ReadImage
main
Does Stack Hashing Work?

Imprecision: Expected Number of Duplicate Vulnerabilities
Recall: Expected Number of Vulnerabilities Discovered

Ideal
Does Stack Hashing Work?

Recall: Expected Number of Vulnerabilities Discovered

Imprecision: Expected Number of Duplicate Vulnerabilities

Room for Improvement in Imprecision

Plain Hashing

Ideal
Does Stack Hashing Work?

Recall: Expected Number of Vulnerabilities Discovered

Imprecision: Expected Number of Duplicate Vulnerabilities

Plain Hashing

ASAN Hashing

Ideal
Importance of Seed Selection
Importance of Seed Selection
Challenge: Multiple Vulnerabilities

```c
int main(int argc, char* argv[]) {
    int x = atoi(argv[1]);
    if (x&1) vulA(1);
    if (x&2) vulB(1);
}
```

Which vulnerability causes `main(3)`?

<table>
<thead>
<tr>
<th>Vuls</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crashes</td>
<td>45859</td>
<td><strong>79626</strong></td>
<td>6860</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
The Patch Tree: Ability to Test Patches Independently

P^1 \rightarrow P^2 \rightarrow P^3 \rightarrow P^4 \rightarrow P^5
Guided Fuzzing
Fuzzing vs. Concolic Execution
Combining Fuzzing and Concolic Execution