DevSecOps:
Delivering Reliable and Secure Software Systems via Automated Bug Finding and Hardening

Jack Davidson
Department of Computer Science
University of Virginia
Contact: jwd@virginia.edu, http://sdg.cs.virginia.edu
Outline of Talk

- Brief History of Software Development Methodologies
  - Waterfall
  - Agile
  - DevOps
- Left Shift from DevOps to DevSecOps
- Fuzzing
- DevSecOps with Helix++
- Research Challenges and Resources
Waterfall Model (Royce)
Waterfall Model

Task & Events
PI Meetings
Integration Events
Demo & Eval
P.1.0 Ch. Ladder
P.2.0 N-Variant Diversity
P.2.1 IR Refinement
P.2.2 F-grained Diversity
P.2.3 M-grained diversity
P.2.4 C-grained diversity
P.2.5 Application Designer
P.3.0 Variant Instaniatiation
P.3.1 IRDB
P.3.2 Dynamic Rewriting
P.3.3 Static Rewriting
P.4.0 Arguments
P.4.1 Diversity Verification
P.4.2 Security
P.5.0 Integ. & Eval.
P.6.0 TA2 Collaboration
P.7.0 Administrative
P.8.0 Transition Activities

Legend
## Waterfall Model

### Table: Cost Summary Table

<table>
<thead>
<tr>
<th>SOW Task</th>
<th>Duration (Months)</th>
<th>Monthly Cost</th>
<th>Subtotal</th>
<th>Total Cost</th>
<th>$000s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.9.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Phase 1 Total

|                      | 12 | 140 | 140 | 377 | 238 | 786 | 500 | 0 | 1684 |

### Phase 2 Total

|                      | 12 | 139 | 140 | 366 | 232 | 768 | 507 | 0 | 1670 |

### Phase 3 Total

|                      | 18 | 165 | 311 | 992 | 0 | 1243 | 735 | 0 | 1798 |

### Project Total

|                      | 42 | 444 | 605 | 1675 | 511 | 2791 | 2540 | 0 | 5320 |

---

Figure 9: Cost Summary Table.
Waterfall Model

- Linear sequencing
- Changing anything is hard
- DOD requires an Engineering Change Proposal (ECP)
  - Takes months to get approved
Agile History

- Meeting at Snowbird, UT February 21, 2001
- Created statement of values

**TECHNOLOGY**

The Winter Getaway That Turned the Software World Upside Down
How a group of programming rebels started a global movement
Manifesto for Agile Software Development: Four Values and Twelve Principles

Values

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
DevOps
Waterfall to DevOps

Application Lifetime
- Years
- Months

Development Process
- Waterfall
- Agile

Application Architecture
- Monolithic
- N-Tier

Deployment & Packaging
- Physical Servers
- Virtual Servers

Application Infrastructure
- Hosted
- Data Center

Source: Nicholas Chaillan, OSD
Outline of Talk

- Brief History of Software Development Methodologies
  - Waterfall
  - Agile
  - DevOps
- Left Shift from DevOps to DevSecOps
- Fuzzing
- DevSecOps with Helix++
- Research Challenges and Resources
Security Vulnerabilities 2019

Even if your software is secure, other software is not
Infrastructure and libraries are not immune!

- Web, name, authentication servers
- Anti-virus software
- Commonly used critical libraries, e.g. OpenSSL, libC,
- Remote desktop

Source: CVE details / NIST National Vulnerability Database
Problem: Traditional DevOps ➔ Fragile Software

Cost to fix [security-critical] bugs grows exponentially throughout software development lifecycle

Stage in software development lifecycle

Source: NIST
DevOps to DevSecOps

Application Lifetime
- Months

Development Process
- Agile

Application Architecture
- N-Tier

Deployment & Packaging
- Virtual Servers

Application Infrastructure
- Data Center

Weeks
- Days

DevSecOps

Microservices

Containers

Cloud

Credit: Nicholas Chaillan OSD
DevSecOps Technology Stack

“Continuous Integration & Continuous Delivery” Orchestration

Credit: Nicholas Chaillan OSD
Outline of Talk

- Brief History of Software Development Methodologies
  - Waterfall
  - Agile
  - DevOps
- Left Shift from DevOps to DevSecOps
- Fuzzing
- DevSecOps with Helix++
- Research Challenges and Resources
Software Fuzz-testing

**fuzz** /fʌz/

n. 1 data that has been fuzzed; data that has been mutated, malformed, invalid data
v. 2 to fuzz; to input/feed malformed data into a hardware or software system with the intention of crashing the system and exposing exploitable vulnerabilities

An Empirical Study of the Reliability of Unix Studies, Barton T. Miller ’89

---

Software Fuzz-testing

**fuzz** /fʌz/

n. 1 data that has been fuzzed; data that has been mutated, malformed, invalid data
v. 2 to fuzz; to input/feed malformed data into a hardware or software system with the intention of crashing the system and exposing exploitable vulnerabilities

An Empirical Study of the Reliability of Unix Studies, Barton T. Miller ’89
Software Fuzz-testing

- Today’s leading **automated bug-finding** approach
- **Uncover bugs** by bombarding program with inputs
- **Coverage-guided search**: breed only the **winners**
  - Measure each input’s code coverage via tracing
  - Keep and mutate only those reaching new code

Trace & maximize code coverage

Source: S. Nagy
Cyber Grand Challenge Competition

2014 Competition to build cyber reasoning system.
Competitors presented with Challenge Binaries (CBs)
Had to find vulnerabilities, prove existence of vulnerabilities, mitigate them, and weaponize them
Of the seven finalists, six independently chose to use fuzzing to find vulnerabilities
Why Binary-only Fuzzing?

Advantages

- Source code often not available
  - Closed source
  - Third-party libraries

- Multi-language systems not supported by all fuzzing tools
  - Legacy (Assembly, Ada, Fortran, etc.)
  - Emerging languages (Rust, Go, etc.)

- Build systems are complex and fragile
  - Difficult to modify build system
  - Specific software tools required to build application

- WYSINWYX – What you see is not what you execute

Disadvantages

- Quality (speed, coverage, etc.) can suffer. Depends on reverse engineering tools
Helix++ Binary Rewriting Infrastructure

- No source code required, no vendor cooperation needed
- Helix key technology in DARPA Cyber Grand Challenge
  - #2 overall, #1 in efficient defenses

- Applied to wide variety of software
- Network facing daemons, drone software, large DOD applications, GUIs, desktop/laptop utilities

- Input Binary
- Repository of transformations (hardening, diversity, fuzzing)
  - SCFI, SLX, GLX, HLX, BILR, MIXR, ZAFL
- Best-of-breed Reverse Engineering Tools
- IRDB SDK
- IR Database
- Layout Engines
- Diversified, hardened, fuzz-enabled functionally-equivalent binaries

University of Virginia
School of Engineering & Applied Science
The Helix ++ ZAFL Platform – Fast Binary-only Fuzzing

- **Statically**-inserted, **inline**d instrumentation with **liveness awareness**

ZAFL Speed Comparison

**Compiler:** 24%, **Assembler:** 34%

**AFL-Dyninst:** 88%, **AFL-QEMU:** 256%

**ZAFL:** 32%, **ZAFL+Transforms:** 27%
## Comparative Evaluation of Binary Rewriters

<table>
<thead>
<tr>
<th>Tool</th>
<th>Null EXE</th>
<th>Null Func</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddiasm</td>
<td>88.97%</td>
<td>85.91%</td>
</tr>
<tr>
<td>e9patch</td>
<td>100.00%</td>
<td>78.34%</td>
</tr>
<tr>
<td>egalito</td>
<td>98.50%</td>
<td>29.39%</td>
</tr>
<tr>
<td>mctoll</td>
<td>0.89%</td>
<td>0.89%</td>
</tr>
<tr>
<td>multiverse</td>
<td>26.31%</td>
<td>10.82%</td>
</tr>
<tr>
<td>reopt</td>
<td>76.43%</td>
<td>33.91%</td>
</tr>
<tr>
<td>retrowrite</td>
<td>9.98%</td>
<td>9.24%</td>
</tr>
<tr>
<td>revng</td>
<td>26.46%</td>
<td>23.50%</td>
</tr>
<tr>
<td>uroboros</td>
<td>6.45%</td>
<td>2.87%</td>
</tr>
<tr>
<td>zipr</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Source: A Broad Comparative Evaluation of X86-64 Binary Rewriters, E. Schulte, M. Brown, V. Folts, CSET 2022
Outline of Talk

- Brief History of Software Development Methodologies
  - Waterfall
  - Agile
  - DevOps
- Left Shift from DevOps to DevSecOps
- Fuzzing
- DevSecOps with Helix++
- Research Challenges and Resources
DevOps ➔ DevSecOps

ZAFL Binary Fuzzing

Input Pool

Instrument ➔ Fuzz ➔ Evaluate Input ➔ Bug Reports

Test

Develop, Patch, Certify, Deploy

Feature Requests, Security Vulns, Bug Reports

Test

Coders ➔ Code

Testers ➔ Unit Tests

Integrators ➔ Integration tests

Certifiers ➔ Acceptance tests

Build ➔ Testing

Integration

Certification
Apply to Microservices

Helix++ enables the production of **hardened** and **diversified** container images
Continuous and Proactive MTD

Key Concepts
Continuous analysis to refine understanding
Continuous fuzzing to identify weaknesses

To deployment, A/B testing

Deployed Live Enterprise System

Telemetry data
Predictive telemetry
Weaknesses

Continuous analysis and anti-fragility refinements

Fuzzing Pods
Continuous Analysis
Anti-Fragility Transforms
Zipr Toolchain

Diversified/Hardened Variant Pool

Proactive and Reactive Improvements
Outline of Talk

- Brief History of Software Development Methodologies
  - Waterfall
  - Agile
  - DevOps
- Left Shift from DevOps to DevSecOps
- Fuzzing
- DevSecOps with Helix++
- Research Challenges and Resources
DevSecOps Research Questions

- What are the metrics for CI fuzzing? Are there different metrics besides coverage?
- Use fuzzing to enhance test suites. Effective or not?
- Leverage production environment to enhance CI fuzzing. Continuous and proactive, reactive improvements (A/B testing)
- Is a Moving Target Defense (MTD) feasible and effective at the microservices level?
- Ensemble fuzzing: Leverage different fuzzing approaches (mutation, coverage metrics, etc.)
- Specialized, targeted fuzzer to test patches
- Differential testing across different build configurations
Resources

- Zafl fuzzer is open source
  - [https://git.zephyr-software.com/opensrc/libzafl/-/wikis/Home](https://git.zephyr-software.com/opensrc/libzafl/-/wikis/Home)
- Turbo (CI/CD) pipeline tools are open source
  - [https://git.zephyr-software.com/opensrc/turbo](https://git.zephyr-software.com/opensrc/turbo)
- Same Coverage, Less Bloat: Accelerating Binary-Only Fuzzing with Coverage-Preserving Coverage-Guided Tracing, S. Nagy, A. Nguyen-Tuong, J. Hiser, J. Davidson, M. Hicks, CCS 2021, [https://doi.org/10.1145/3460120.3484787](https://doi.org/10.1145/3460120.3484787)
- A Broad Comparison of x86-64 Binary Rewriters, E. Schulte, M. Brown, V. Folts, CSET 2021, [https://doi.org/10.1145/3546096.3546112](https://doi.org/10.1145/3546096.3546112)
Acknowledgments

- This material is based upon work supported by the National Science Foundation under
  - Grant No. 2115130—Helix++: Securing Open Science Platforms
  - DARPA CGC, Contract No. FA8750-14-C-0110, and
  - DARPA CFAR Contract No. FA8750-15-C-0118
- Thanks to Stefan Nagy for slides and graphs
- Thanks to Jason Hiser and Anh Nguyen-Tuong for slides and advice
Questions