

Projektgruppe WS 2018/2019



# FETA

## Fuzzer Evaluation Through Automation

Michael Rodler

Sebastian Surminski

Prof. Dr.-Ing. Lucas Davi

Juniorprofessur für Informatik

<https://www.syssec.wiwi.uni-due.de/>

UNIVERSITÄT  
DUISBURG  
ESSEN

*Offen im Denken*

# Complexity of Software Systems



**Large attack surface for exploits**

# From Bug to Vulnerability

- C/C++ are plagued by memory safety violations (Bugs)
  - Missing bounds-checking (spatial)
  - Object lifetime violations (temporal)
- Memory Error – Memory Corruption – Arbitrary Code Execution
  - Often memory errors can be abused to corrupt memory
  - Code-pointers, data-pointers etc.
  - Hijack program execution – Return-Oriented Programming (ROP)

# Example: Buffer Overflow

```
char* input = get_input();  
char buffer[128];  
int len = *((int*)input);  
memcpy(buffer, input + sizeof(len), len);
```

Fixed Buffer Length!

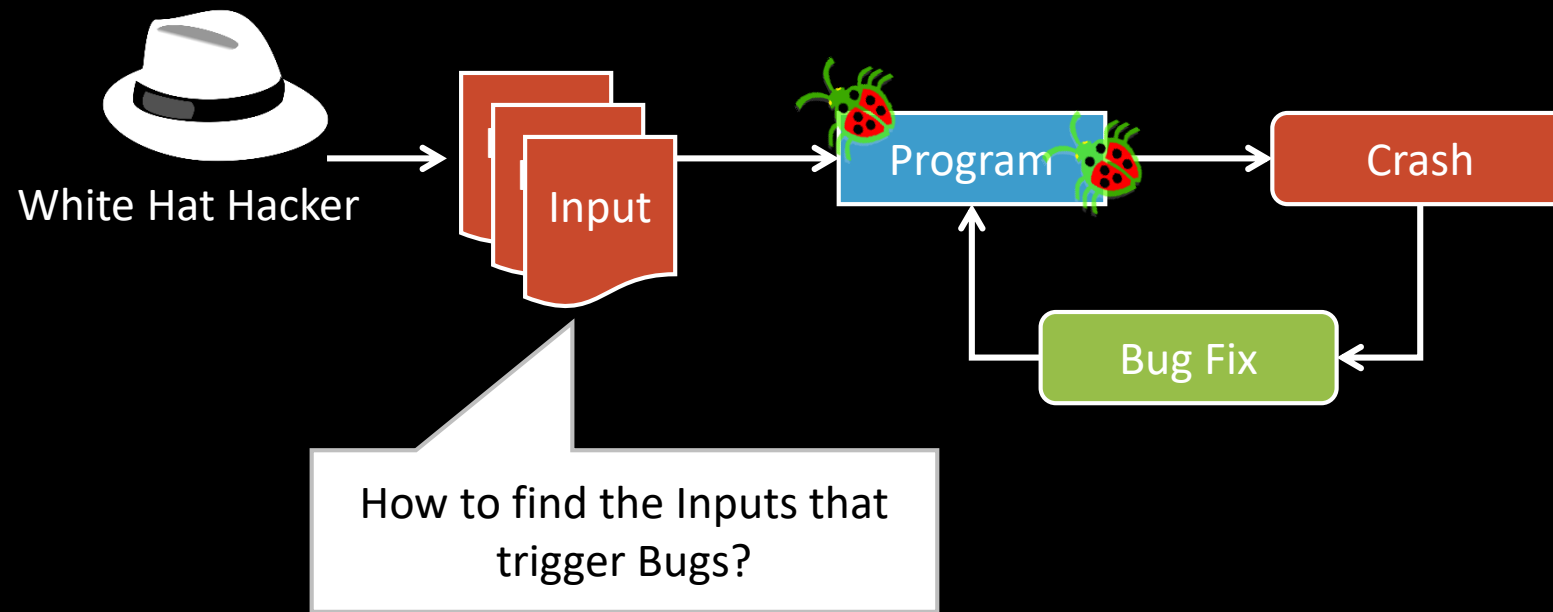
Attacker controlled length!

Stack



Hijack Program Execution

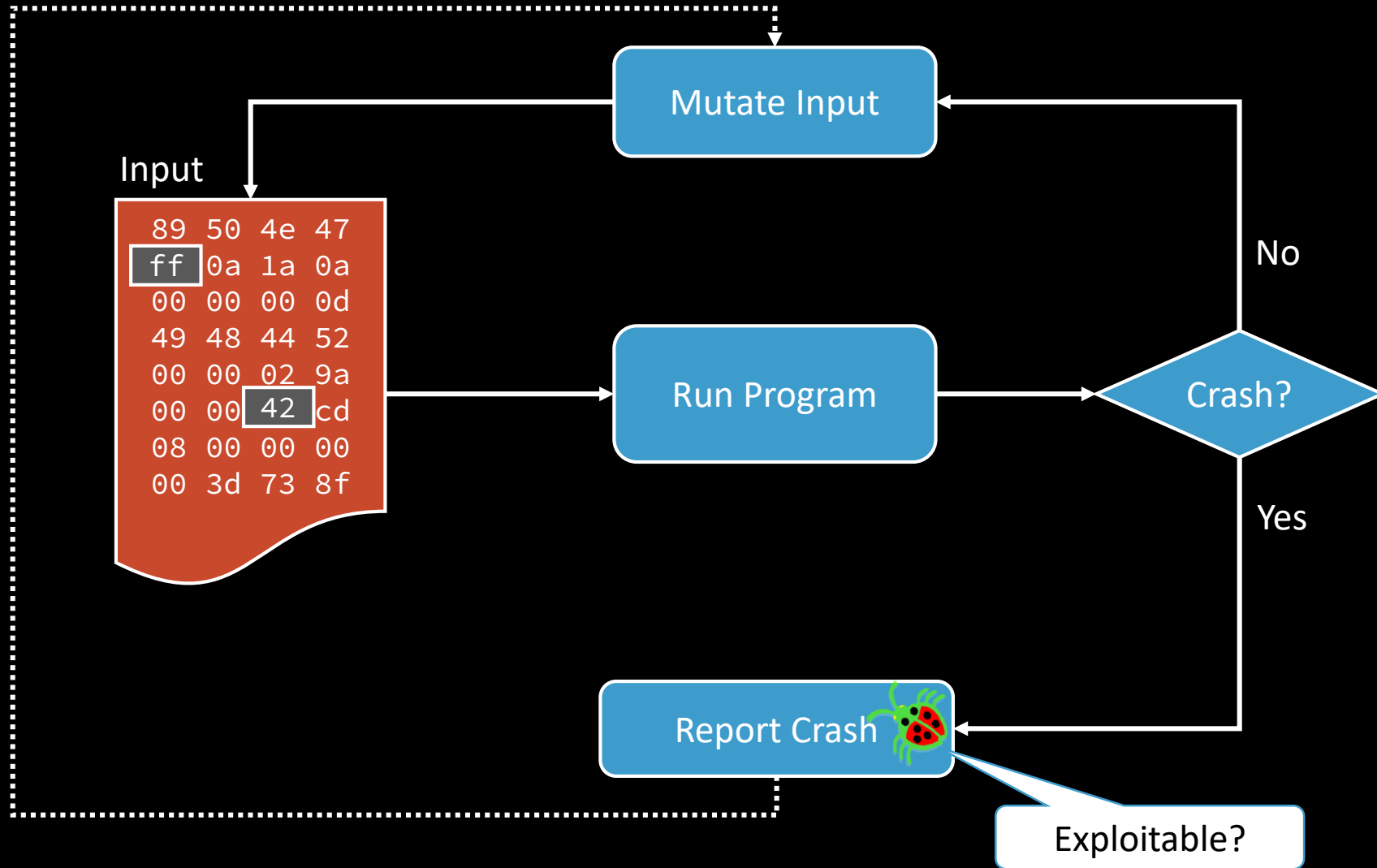
# Vulnerability Discovery




# Fuzzing

Brute-Force Vulnerability Discovery

# Basic Fuzzer Workflow




# Fuzzing – The Good, The Bad, ...



- +Simple
- +Scalable / Parallel
- +Good Results in Practice

- Simple == dumb?
- Resource-hungry
- Manual optimizations





# ... The Ugly

## Program

```
int magic = input();  
if (magic == 0xdeadbeef) {  
    vulnerability();  
}
```

- How long does it take?
- It is not guaranteed to succeed
- Probabilistic process

## Inputs

ff 42 43 ff

ff 42 4b ff

ff 42 ff 4b

ff 42 bf 4b

...

de ad be ef

## Mutation

Bit-flip

Swap Bytes

Bit-flip

???

# Research in Optimizing Fuzzing

Coverage-based Greybox Fuzzing

mboehme / aflfast

forked from mirrorer/afl

Watch 19

Unstar 211

Fork 158

Directed Greybox Fuzzing

aflgo / aflgo

forked from mirrorer/afl

Watch 15

Star 96

Fork 158

Angora: Efficient Fuzzing by Principled Search

Driller: Augmenting

Fuzzing Through Selective Symbolic Execution

shellphish / driller

Watch 40

Unstar 340

Fork 86

VUzzer: Application-aware Evolutionary Fuzzing

T-Fuzz: fuzzing by program transformation

Hui Peng  
Purdue University  
peng124@purdue.edu

Yan Shoshitaishvili  
Arizona State University  
yans@asu.edu

Mathias Payer  
Purdue University  
mathias.payer@nebelwelt.net

Kumar<sup>‡</sup>, Lucian Cojocar\*<sup>†</sup>,  
Science Institute, Vrije Universiteit  
cojocar>@vu.nl; <giuffrida,he  
sterdam Department of Inform  
Institute of Information Techno  
jain,ashish.kumar>@research

vusec / vuzzer

Watch 36

Unstar 205

Fork 80

Code

Issues 6

Pull requests 0

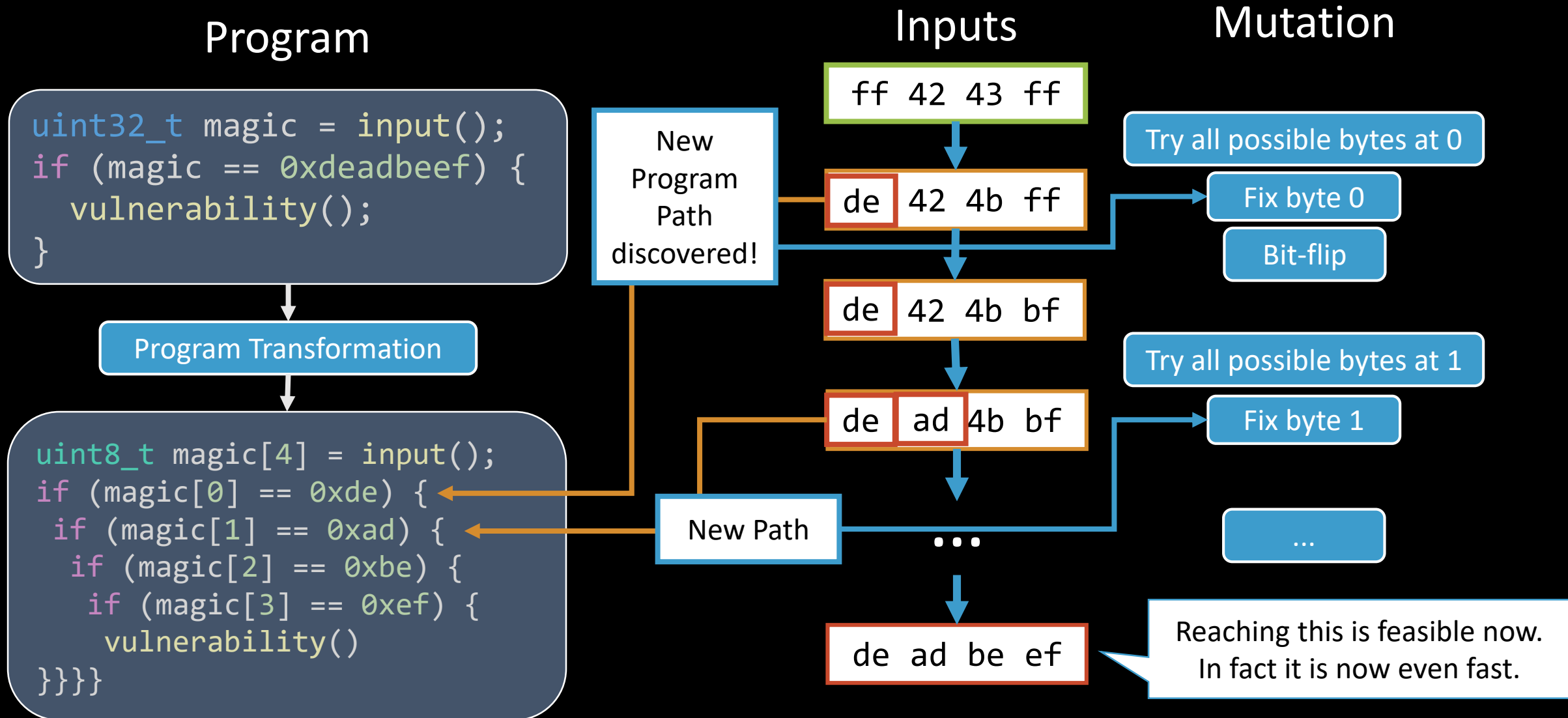
Projects 0

Wiki

Insights

No description, website, or topics provided.

# Optimized Fuzzing



Research Question:

# Evaluation and Comparison of State-of-the-Art Fuzzing Approaches

# Goals and Features

- 1. Automated Setup** of Fuzzing Instances
- 2. Sharing** of Generated Test-Cases
- 3. Comparison and Evaluation** of Fuzzers
- 4. Combine** all Fuzzers against one Target
- 5. Report** Bugs and Write PoC Exploits

# PG Outline



Lectures

Seminar

Implementation

Report

# Lectures – The Basics

- Program Exploitation: „From bugs to vulnerabilities to exploits“
  - Types of vulnerabilities (e.g. buffer overflow, use-after-free)
  - Practical exploit development
- Memory Error Detection
  - Hardening and Debugging Tools (AddressSanitizer, valgrind, etc.)
  - Root-cause analysis
- Basics of Program Analysis and Transformation
  - E.g. symbolic/concolic execution
  - Useful for understanding some optimized Fuzzers

# Seminar Topics – The Fuzzers

- Summarize relevant literature
  - Presentation
  - Related work in final report
- Possible Topics:
  - Improved Fuzzing Search Strategies  
[Chen and Chen, IEEE S&P 2018], [Böhme et al., ACM SIGSAC 2016]
  - Improving Fuzzing with Static Program Analysis  
[Rawar et al., NDSS 2017], [Peng et al., IEEE S&P 2018]
  - Hybrid Fuzzing and Symbolic Execution  
[Stephens et al., NDSS 2016], [Yun et al., USENIX Security 2018]
  - Fuzzing Embedded Systems  
[Chen et al., NDSS 2018], [Muench et al., NDSS 2018]



# Implementation

- Project Management
  - Issues and Milestones
  - Everything in git
- Implementation
  - Choose automation framework
  - Choose/Create Data-Sets for Evaluation
  - Setup and automate Experiments
  - Analyse results & write PoC exploits

# Documentation

- Goal: **Publishable** project report!
  - Continuous writing and feedback loop
- Related work
  - i.e. the Fuzzers explained
  - Comparison of approaches
- Description of Experiments
  - Data-sets
  - Statistics
- PoC Exploits

american fuzzy lop 0.47b (readpng)

<b>process timing</b>		<b>overall results</b>	
run time	: 0 days, 0 hrs, 4 min, 43 sec	cycles done	: 0
last new path	: 0 days, 0 hrs, 0 min, 26 sec	total paths	: 195
last uniq crash	: none seen yet	uniq crashes	: 0
last uniq hang	: 0 days, 0 hrs, 1 min, 51 sec	uniq hangs	: 1
<b>cycle progress</b>		<b>map coverage</b>	
now processing	: 38 (19.49%)	map density	: 1217 (7.43%)
paths timed out	: 0 (0.00%)	count coverage	: 2.55 bits/tuple
<b>stage progress</b>		<b>findings in depth</b>	
now trying	: interest 32/8	favored paths	: 128 (65.64%)
stage execs	: 0/9990 (0.00%)	new edges on	: 85 (43.59%)
total execs	: 654k	total crashes	: 0 (0 unique)
exec speed	: 2306/sec	total hangs	: 1 (1 unique)
<b>fuzzing strategy yields</b>		<b>path geometry</b>	
bit flips	: 88/14.4k, 6/14.4k, 6/14.4k	levels	: 3
byte flips	: 0/1804, 0/1786, 1/1750	pending	: 178
arithmetics	: 31/126k, 3/45.6k, 1/17.8k	pend fav	: 114
known ints	: 1/15.8k, 4/65.8k, 6/78.2k	imported	: 0
havoc	: 34/254k, 0/0	variable	: 0
trim	: 2876 B/931 (61.45% gain)	latent	: 0



# Questions?

# References

- L. Szekeres, M. Payer, T. Wei, and D. Song, “SoK: Eternal War in Memory,” in 2013 IEEE Symposium on Security and Privacy, 2013, pp. 48–62.
- M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury, “Directed Greybox Fuzzing,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 2329–2344.
- M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, “What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices,” in Proceedings 2018 Network and Distributed System Security Symposium, San Diego, CA, 2018.
- P. Chen and H. Chen, “Angora: Efficient Fuzzing by Principled Search,” arXiv [cs.CR], 04-Mar-2018.
- H. Peng, Y. Shoshitaishvili, and M. Payer, “T-Fuzz: fuzzing by program transformation.”
- B. Shastri et al., “Static Program Analysis as a Fuzzing Aid,” in Research in Attacks, Intrusions, and Defenses, 2017, pp. 26–47.
- S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, “Vuzzer: Application-aware evolutionary fuzzing,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), 2017.
- N. Stephens et al., “Driller: Augmenting Fuzzing Through Selective Symbolic Execution,” in NDSS, 2016, vol. 16, pp. 1–16.
- J. Chen et al., “IoTfuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing,” in Proceedings 2018 Network and Distributed System Security Symposium, San Diego, CA, 2018.
- I. Yun, S. Lee, M. Xu, Y. Jang, and T. Kim, “QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing,” in 27th USENIX Security Symposium (USENIX Security 18), 2018.