Secure Product Lifecycle

Security Testing: Fuzzing

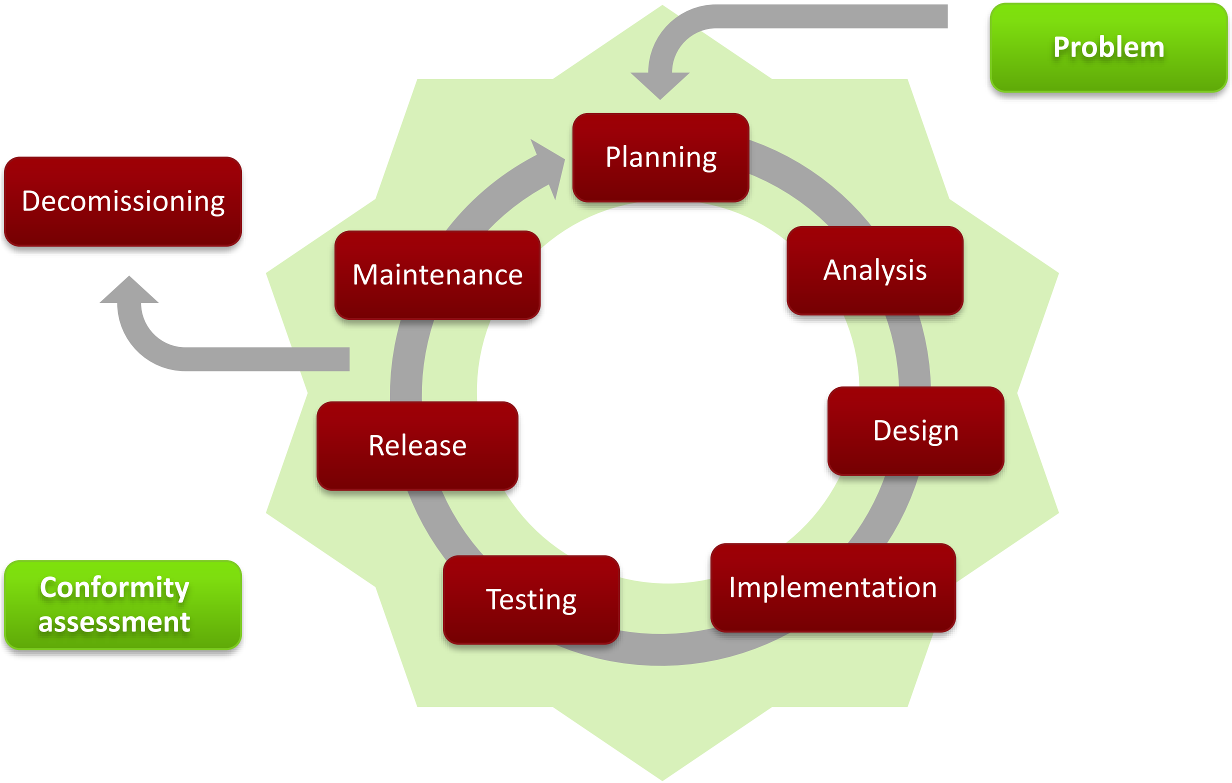
Srđan Ljepojević

Agenda

- Context
- Motivation
- What is fuzzing?
- Advantages and Challenges
- Fuzzing in Standards
- Conclusion

Yagoba GmbH, Austria, www.yagoba.com, info@yagoba.com

Secure **Product** Lifecycle



Secure **Development** Lifecycle

Requirements
- Document security objectives
- Secure requirements review

Design
- Threat modeling
- Secure design review

Implementation
- Developer training
- Coding standards
- Secure code review
- Static code analysis

Testing
- **Negative testing**
- Known vulnerabilities
- Penetration testing

Deployment
- Secure update process
- Vulnerability management
- Security incident response process

Motivation

- Testing is an important aspect of security assessments and certification

- Why fuzzing?
- CWE Top 25 in 2022

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0

https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

What bugs can you find with fuzzing?

CWE Top 2023

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
6	CWE-20	Improper Input Validation	15.50	35	-2
7	CWE-125	Out-of-bounds Read	14.60	2	-2
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.11	16	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.73	0	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.41	5	0
11	CWE-862	Missing Authorization	6.90	0	+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1
14	CWE-190	Integer Overflow or Wraparound	5.89	4	-1
15	CWE-502	Deserialization of Untrusted Data	5.56	14	-3

https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html#tableView

What bugs can you find with fuzzing?

CWE-193	Off-by-One Error	**CWE-415**	Double Free	**CWE-662**	Improper Synchronization	**CWE-590**	Free Memory Not on the Heap
CWE-823	Use of Out-of-Range Pointer Offset	**CWE-1102**	Reliance on Machine-Dependent Data Representation	**CWE-839**	Numeric Range Comparison Without Minimum Check	**CWE-562**	Return of Stack Variable Address
CWE-786	Access of Memory Location Before Start of Buffer	**CWE-195**	Signed to Unsigned Conversion Error	**CWE-131**	Incorrect Calculation of Buffer Size	**CWE-587**	Assignment of a Fixed Address to a Pointer
CWE-680	Integer Overflow to Buffer Overflow	**CWE-129**	Improper Validation of Array Index	**CWE-1223**	Race Condition for Write-Once Attributes	**CWE-588**	Attempt to Access Child of a Non-Structure Pointer
CWE-466	Return of Pointer Value Outside of Expected Range	**CWE-366**	Race Condition Within a Thread	**CWE-368**	Context Switching Race Condition	**CWE-362**	Signal Handler Race Condition
CWE-119	Improper Restriction of Operations Within the Bounds of a Memory Buffer	**CWE-367**	Time-of-Check Time-of-Use (TOCTOU) Race Condition	**CWE-421**	Race Condition During Access to Alternate Channel	**CWE-1105**	Insufficient Encapsulation of Machine-Dependent Functionality
CWE-758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	**CWE-843**	Access of Resource Using Incompatible Type ("Type Confusion")	**CWE-1257**	Improper Access Control Applied to Mirrored or Aliased Memory Ranges		

https://www.code-intelligence.com/blog/what-bugs-can-you-find-with-fuzzing

In news

Büro-Drucker mit löcheriger Firmware – Sicherheitsniveau wie vor Jahrzehnten

Forscher fanden rund 50 Schwachstellen in Druckern von Brother, HP, Lexmark, Kyocera, Ricoh und Xerox. Einige sind weiterhin ungepatcht.

Lesezeit: 2 Min. In Pocket speichern

Security

Don't worry about those 40 Linux USB security holes. That's not a typo

Move along. Nothing to see here. By the way, try this flash drive in your laptop, ta

By Thomas Claburn in San Francisco 7 Nov 2017 at 20:49 65 SHARE ▼

NEWS

Zscaler finds 117 Microsoft 365 bugs via SketchUp 3D file type

Microsoft published patches to address all 117 Microsoft 365 Apps flaws disclosed Tuesday, and the tech giant has disabled support for SketchUp, or SKP, 3D model files.

Researchers find 36 new security flaws in LTE protocol

South Korean researchers apply fuzzing techniques to LTE protocol and find 51 vulnerabilities, of which 36 were new.

By Catalin Cimpanu for Zero Day | March 23, 2019 -- 08:00 GMT (08:00 GMT) | Topic: Security

Vulnerability in Volkswagen Discover Media Infotainment System Addressed by the Company

The medium severity vulnerability in Volkswagen Discover Media was found by a user who presented the details to the company that confirmed the impact of the vulnerability.

by thecyberexpress — June 27, 2023 Reading Time: 3 mins read

Cyber Security News | News | Vulnerabilities

Hacker Discovered "God Mode" Whilst Fuzzing Some Old x86 CPU's

August 12, 2018 Harikrishna Mekala 1796 Views Chips, god mode cpu hack, god mode x86, hacking cpu x86,

New fuzzing tool finds 26 USB bugs in Linux, Windows, macOS, and FreeBSD

Eighteen of the 26 bugs impact Linux. Eleven have been patched already.

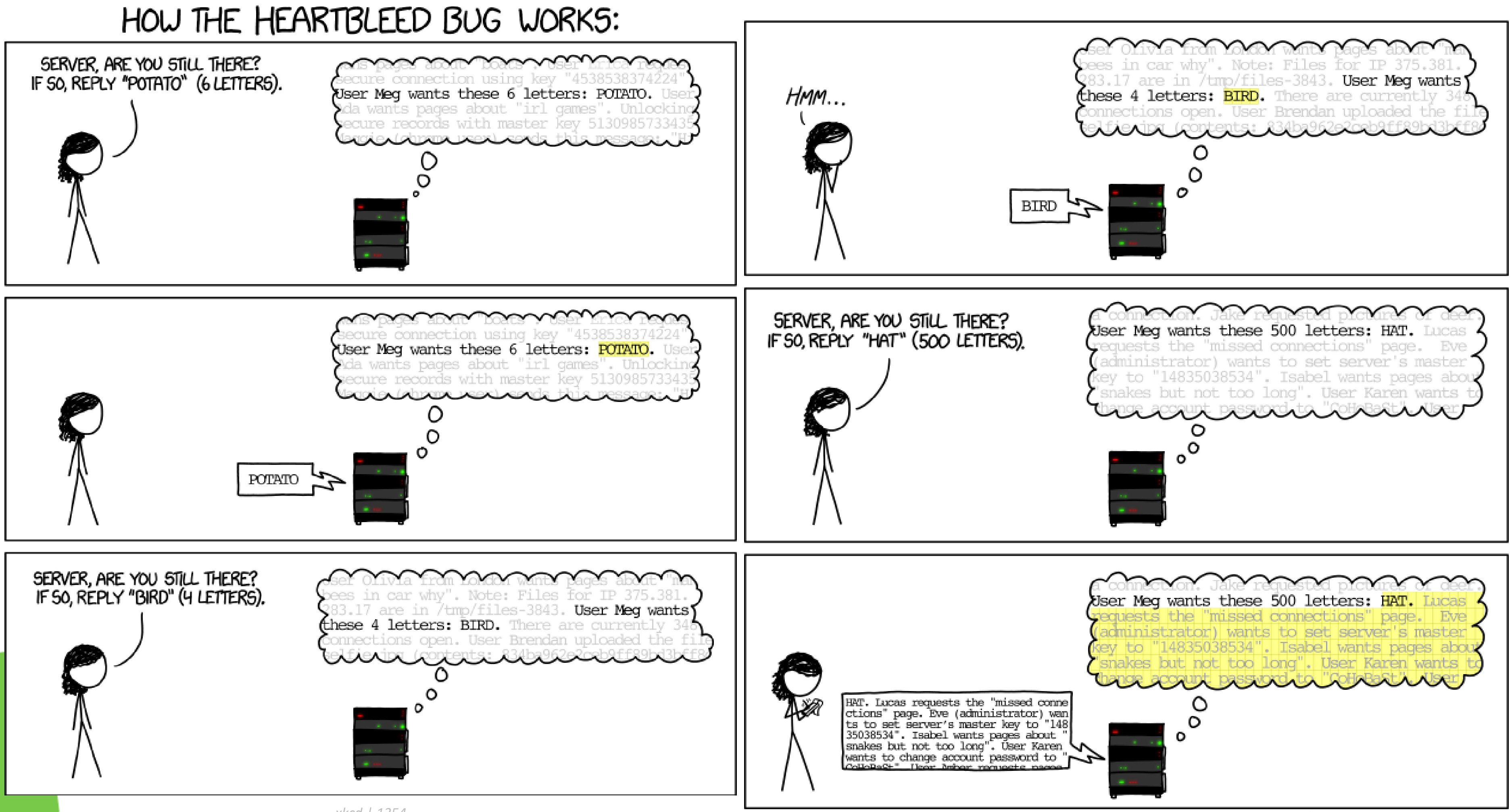
By Catalin Cimpanu for Zero Day | May 27, 2020 -- 11:23 GMT (12:23 BST) | Topic: Security

BrokenType: Google-Tool spürt Font-Exploits in Windows auf

Google veröffentlicht sein Fuzzing-Werkzeug, mit dem man zwischen 2015 und 2017 fast 40 Schriftarten-Sicherheitslücken in Windows aufgespürt hatte.

Heartbleed

- OpenSSL vulnerability (introduce 2012, disclosed 2014)
- Heartbeat extension
  - Heartbeat request: Payload + length
  - Heartbeat answer: Payload
- Improper input validation in the source code
  - → buffer over-read
- Memory after payload could store
  - Session cookies, passwords
  - Cryptographic keys, …
- Impact
  - Worked in both directions!
  - Compromised crypto keys, credentials
  - Launch of Google Project Zero
  - 500 million dollars
- AFL + ASan could have detected Heartbleed *(Hanno Boeck, 2015)*


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
User Meg wants these 6 letters: POTATO.
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Meg wants these 4 letters: BIRD.
HMM...
BIRD
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt".
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt".

OSS-Fuzz

- Continuous fuzzing since 2016
- Identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 as of August 2023
- For open-source developers
- Free of charge

FUZZING

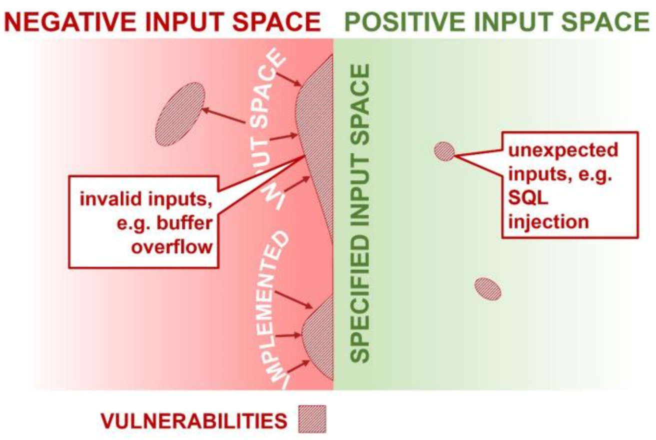
Software Testing

Positive

- Functional testing
- Testing for the functional correctness

Negative

- Security testing
- Testing the robustness of a system
- Test with anomalous inputs to show absence of undesired functionality that may lead to crashes, exposure of protected information, etc.

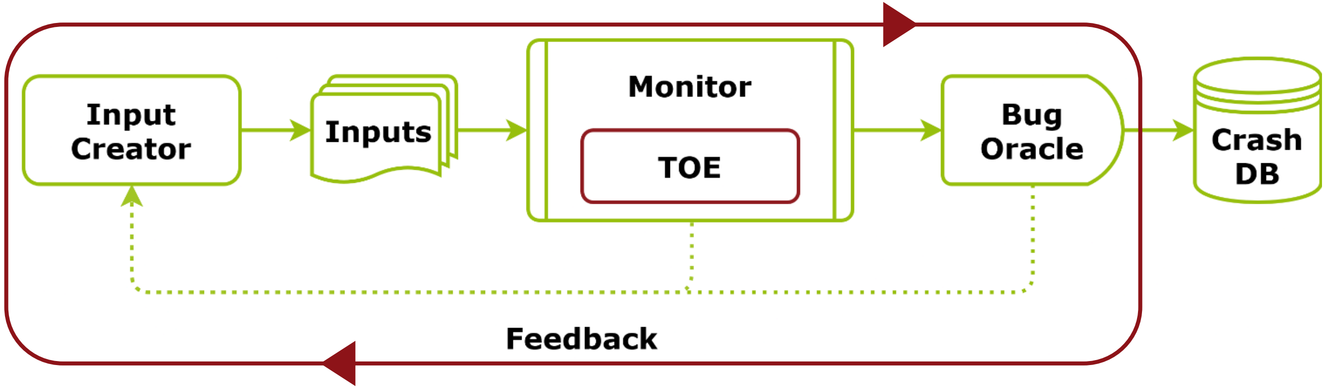


BSI: Fuzzing Primer

Fuzzing in a nutshell

xkcd | 1137

- Part of negative testing
- Automated way of finding security vulnerabilities
- Provide **invalid, unexpected, or random data** as inputs
- **Monitor** the device or program under test for exceptions such as **crashes, memory corruptions, assertion failures, etc.**
- *Fuzzer* – tool that performs fuzz testing



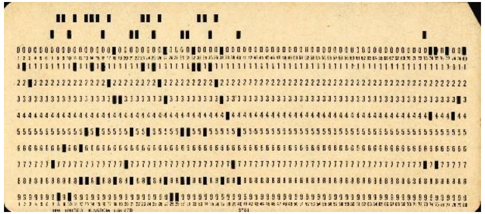
How it all began?

- *"On a dark and stormy night …"* – Miller et al. 1990
- Spurious characters on the line
- Interferences were not surprising, but that the spurious characters caused programs to crash
- Naïve approach, but impressive:
  - 90 programs tested, 24% crashed
- Key message: *"on receiving unusual input, they might exit with minimal error messages, but they should not crash."*
- Triggered a significant area of research and commercial tools

https://pixabay.com/photos/lightning-rain-storm-thunderstorm-4702140/

History

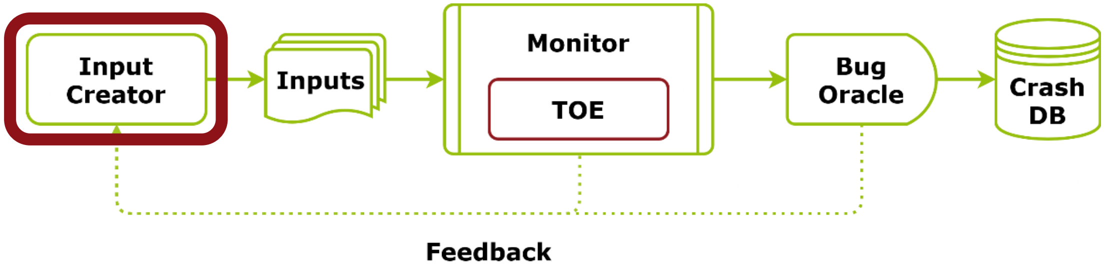
- 1950s
  - Random punch cards used to find bugs
- 1980s
  - Tests with random files and command-line parameters
  - Reliability testing of Unix programs
- 1990s
  - Barton Miller et al. coined the term *"fuzz"*: *"… generates a stream of random characters to be consumed by a target program"*
- 2000s
  - Various test suites have been developed (e.g., PROTOS, SPIKE)
- 2005
  - MS includes fuzzing in the Security Development Lifecycle



https://flickr.com/photos/93001633@N00/5151286161

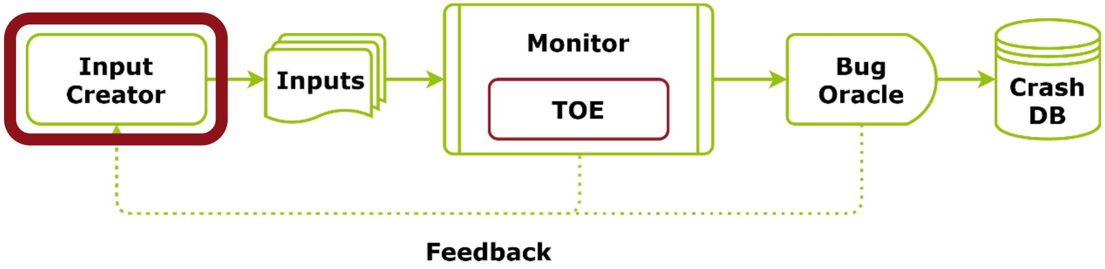
CONCEPT OF FUZZING

Input Creation



- Categorization based on how input is created:
  - Mutation-based
  - Generation-based

Input Creation



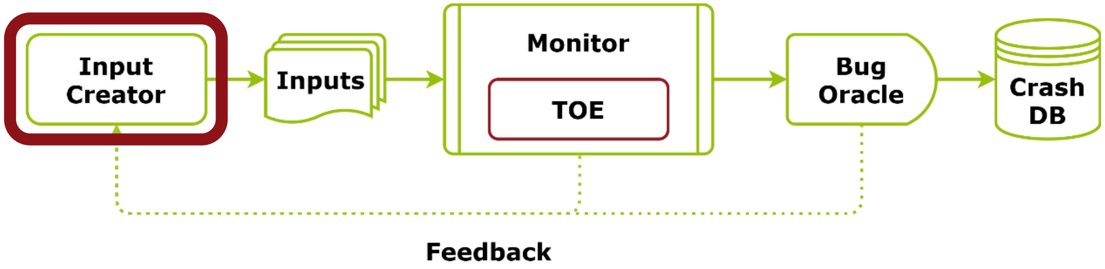
Random

- Simple, but most input data will fail to penetrate the target code
- Probability for generating a “mostly” correct test case is very low
- Basic input validation checks will reject inputs
  - Version numbers
  - Checksums

main.py

```python
import random


class RandomFuzzer:
    def __init__(self, length=10):
        self.length = length

    def fuzz(self):
        for _ in range(3):
            data = self.gen_data()
            self.send(data)

    def gen_data(self):
        data = ""
        for _ in range(self.length):
            # Random ASCII value
            data += chr(random.randrange(32, 127))

        return data

    def send(self, data):
        print(data)


if __name__ == "__main__":
    fuzzer = RandomFuzzer(10)
    fuzzer.fuzz()
```

Input Creation



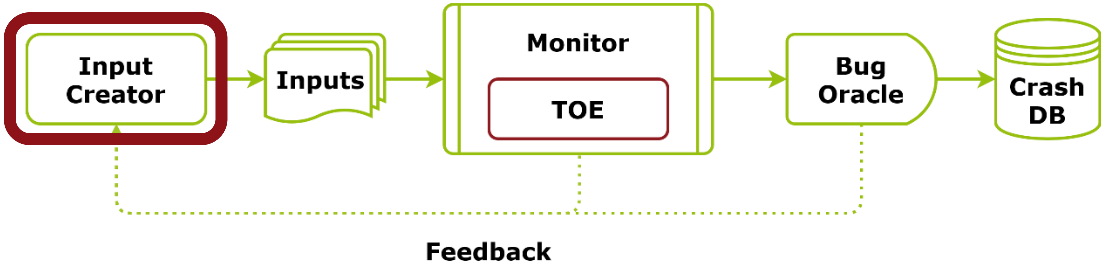
Template or mutation-based

- Modify valid inputs to create test cases
- Corpus might be produced by human or automated
- Problems
  - Protocols with integrity validation (checksums)
  - Stateful protocols (session IDs)
  - Encrypted protocols
- Example: Radamsa

```
[ blackarch ~ ]# echo "1+(3-4)*5" | radamsa --seed 200 -n 7
2+(-1589572-2147483648)*0
2+(-7982779841430270358865270-2147483648)*2147549184
1+(3-4294967293)*5
1+(658871+(3-4)(0-5)*1
2147483647+(3-1)*6
1+(3-2147483652)*5
[ blackarch ~ ]#
```

```
[ blackarch ~ ]# echo "1+(3-4)*5" | radamsa --seed 200 -n 7 | bc
2
-17143412334515231113907253670182910
-21474836449
(standard_in) 4: syntax error
2147483659
-10737418244
[ blackarch ~ ]#
```

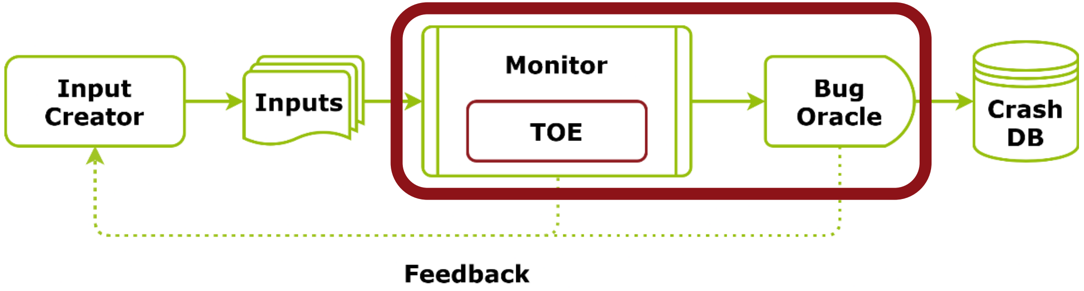
Input Creation



Generation-based

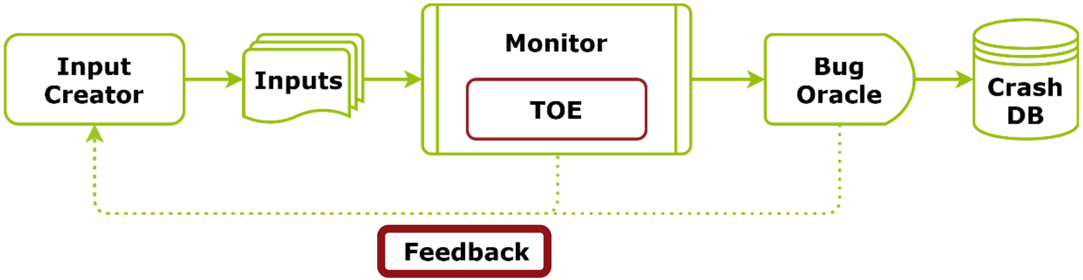
- Generate input from scratch
- Require TOE data knowledge:
  - Use specification, grammar, valid corpus
- Understand protocol, file format, API, …
- Rules: structure and type of packet/message
- Rules are known and can be broken
- Protocol inference (proprietary protocols):
  - NW traces and reverse engineering

Monitoring



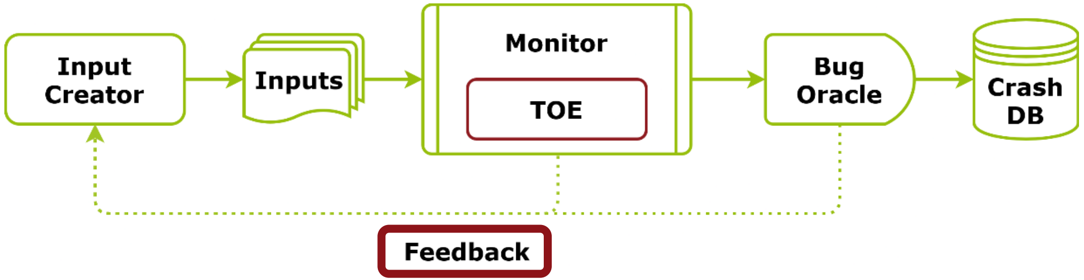
- Improved error-detection capabilities
  - Crashes, hangs, data races, or non-termination
- *AddressSanitizers, DataFlowSanitizer, ThreadSanitizer, LeakSanitizer, …*
  - Drawbacks
    - Performance and memory overhead
    - Recompile code

Feedback



- Categorization based on TOE knowledge/feedback:
  - Blackbox
  - Greybox
  - Whitebox

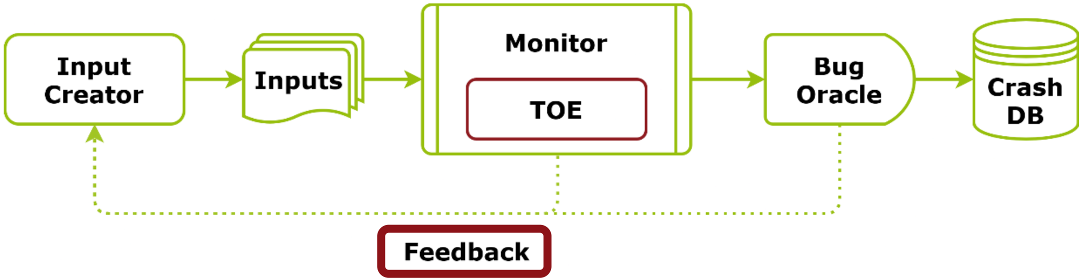
Feedback



Blackbox

- No TOE knowledge
- No to minimal feedback
  - Number of crashes/bugs found
  - Time spent

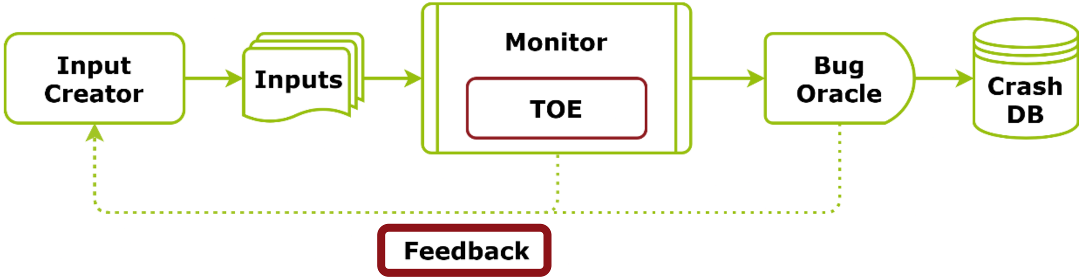
Feedback



Whitebox

- Full TOE access
- Data created based on info analysing the internals of the TOE and the information gathered when executing the TOE
- Approaches:
  - Concolic execution (concrete + symbolic)
  - Taint analysis

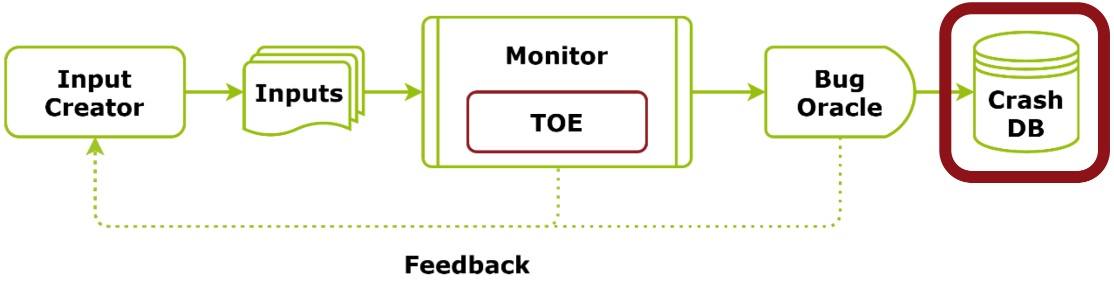
Feedback



Greybox

- Grey-box fuzzing is a variant of white-box fuzzing that can only obtain some partial information from each fuzz run
- Program instrumentation to get lightweight feedback
- Approaches:
  - Lightweight static analysis and code coverage
    - Branch/Node Coverage

Triage



- Crashes are (typically) analyzed manually
- Triage
  - Deduplication (pruning test cases triggering the same bug)
  - Test case minimization (reduce the size of the input)

https://pixabay.com/illustrations/ai-generated-man-magnifying-glass-8583124/

Categorization

- Input creation:
  - Mutational
  - Generational
- Information they have about the TOE/feedback:
  - Blackbox
  - Greybox
  - Whitebox

ADVANTAGES OF FUZZING

Advantages

- Automatic discovery
- Fast
- *(Usually)* Low effort
- Proof of crash/unexpected behaviour
- Covers edge cases
- Interesting inputs due to randomness
- Various bug types
- Highly effective

COMPLEXITY & CHALLENGES OF FUZZING

Challenges

- Fuzzing Success?
  - How can we assess residual security risk if the fuzzing campaign was unsuccessful?
  - What is the time budget?
  - How to evaluate fuzzers?
- Various Targets:
  - Different TOE Types (file, network, UI, web, kernel I/O, or multi-threaded)
  - Stateful fuzzing
- Usability
- …

https://pixabay.com/illustrations/climbing-climber-ice-pick-rope-4514507/

Challenges: Software Fuzzing

- Input creation: balance between
  - Exploring new paths (control flows), and
  - Executing the same path with different input (data flow)
- Efficient mutation operators
- Kernel fuzzing
  - Crashes bring the whole system
- Protocol fuzzing
  - Proprietary protocols
  - Great deal of work to understand the specification

Challenges: Hardware Fuzzing

- SW fuzzing relies on the detection of crashes,
  but on IoT devices memory corruptions are less visible
- Bug oracles: Must be even more sophisticated
  - Liveness checks
- Complex protocols (USB) and various interfaces (wired, wireless)
- Performance
- Resetting a device after a crash
- Instrumentation support for platform limited

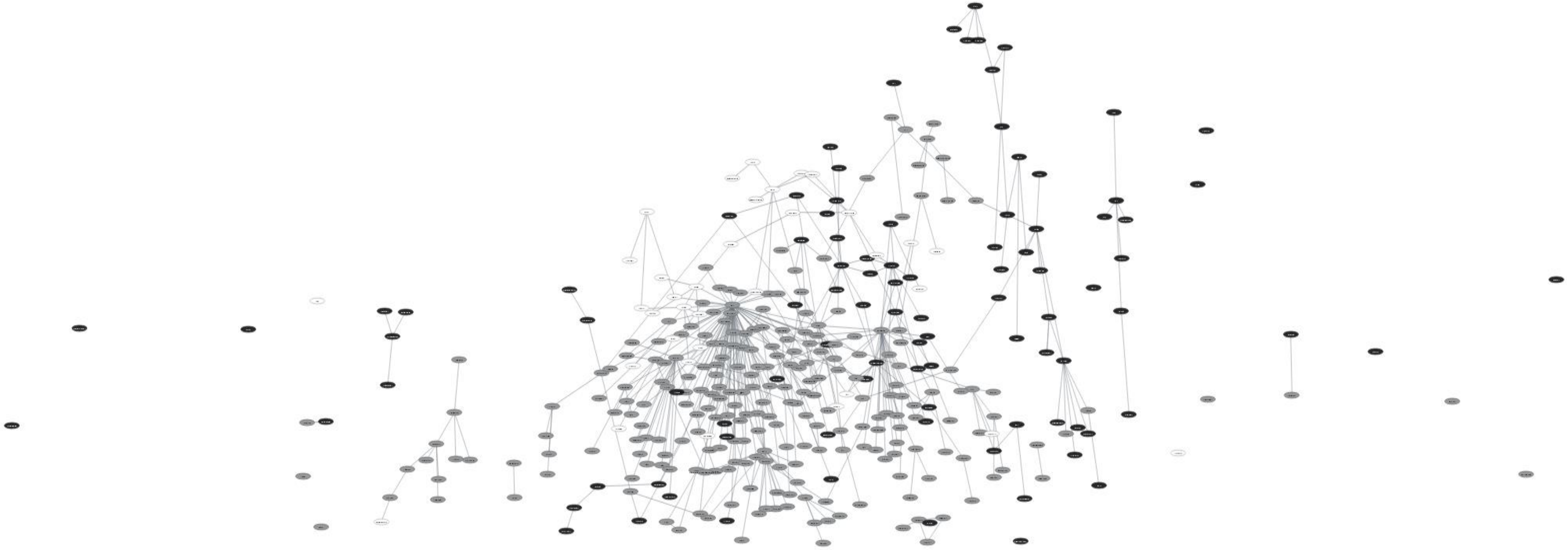
Protocol Inference

- Protocol necessary for generation-based fuzzers
- Challenge
  - IoT, embedded, and industrial network devices with proprietary protocols
  - Reading and analyzing specifications or
    reverse engineering NW traces is time consuming
- Machine learning approaches to infer protocol grammar (research topic)
  - Corpus of real messages
  - Learn protocol grammar
  - Generate test cases

FUZZING TOOLS/FRAMEWORKS

Yagoba GmbH, Austria, www.yagoba.com, info@yagoba.com

dharma
Trinity java-afl
DOMFuzz MiniFuzz
BuzzFuzz antiparser T-Fuzz GPF
boofuzz tlsfuzzer GLADE python-afl
CapFuzz KiF KernelFuzzer WinAFL
Sharefuzz IoTFuzzer FileFuzz
LTEfuzz cross fuzz FLAX AFLFast
Sulley
LibFuzzer Fuzzotron PROTOS
netzob SPIKEfile LZFuzz
Skyfire Vuzzer
AFL
kAFL BitFuzz zzuf
beSTORM Radamsa ref fuzz AFLGo
FairFuzz
kelinci KLEE Peach SPIKE BFF jFuzz
perf fuzzer MozPeach honggfuzz
Hawkeye
Angora BlendFuzz
NAUTILUS
classfuzz PeriScope
KameleonFuzz
lllfuzzer
PULSAR TLS-Attacker Defensics SNOOZE
LangFuzz fsfuzzer
Fuzzbox IFuzzer
fuzzowski
SmartFuzz
CodeAlchemist



https://fuzzing-survey.org/

Open Source

- OSS-Fuzz
- american fuzzy lop
- Radamsa - a flock of fuzzers
- APIFuzzer - fuzz test without coding
- Jazzer - fuzzing for the JVM
- ForAllSecure Mayhem for API
- Sulley Fuzzing Framework
- boofuzz
- Bfuzz
- FuzzDB
- Ffuf
- go-fuzz

FUZZING IN STANDARDS

Fuzzing in Standards

- **UL2900-1** and **UL2900-2-1:** Healthcare and Wellness Systems - Software Cybersecurity for Network-Connectable Products
- Malformed input testing
  - *“The product shall continue to operate as intended when subject to invalid or unexpected inputs on its external interfaces …”*
- Consider
  - File inputs
  - Remote interfaces
  - Supported protocols
- Approach
  - Generational malformed input tools for specific protocols
    - > 1 Mio unique / independent tests cases or 8 hours
  - Template malformed input testing may be used (proprietary protocols)
    - > 5 Mio unique / independent test cases or 8 hours

Fuzzing in Standards

- **Common Criteria**
  - *Attacks based on forcing the TOE to cope with unusual or unexpected circumstances should always be considered.*
- **DIN SPEC 27027** (Mindestanforderungen an IoT-fähige Geräte)
  - *It is recommended that IT-security implementations of IoT-devices are tested by means of fuzzing.*
- **IEC 62443**: Security for Industrial Automation and Control Systems

Medical Standards

- **MDCG 2019-16** Guidance on Cybersecurity for medical devices
- **Cybersecurity in Medical Devices:** Quality System Considerations and Content of Premarket Submissions by the U.S. Food and Drug Administration (FDA)
- **IEC 81001-5-1** Health software and health IT systems safety, effectiveness and security. Part 5-1: Security — Activities in the product life cycle.
- **AAMI TIR 57:2016** Principles For Medical Device Security - Risk Management

https://www.code-intelligence.com/what-is-fuzz-testing

Road Vehicle Standards

- **SO 26262** Road vehicles – Functional Safety
- **UNECE WP.29** United Nations World Forum for Harmonization of Vehicle Regulations
- **Automotive SPICE for Cybersecurity Guidelines**
- **ISO/SAE 21434** Road Vehicles — Cybersecurity Engineering

https://www.code-intelligence.com/what-is-fuzz-testing

Even more standards

- **ISO/IEC/IEEE 29119** Software and Systems Engineering - Software Testing
- **ISO/IEC 12207** Systems and Software Engineering – Software Life Cycle Processes
- **ISO 27001** Information Technology – Security Techniques – Information Security Management Systems
- **IT-Grundschutz (Germany)** Based on ISO 27001
- **ISO 22301** Security and Resilience — Business Continuity Management Systems
- **NIST** Guidelines on Minimum Standards for Developer Verification of Software
- **NIST SP 800-95** Web Services — standard for software testing (USA) and others
- **SA-11:** Developer Security Testing And Evaluation

https://www.code-intelligence.com/what-is-fuzz-testing

CONCLUSIONS

Conclusions

- Fuzzing verifies code that processes input at trust boundaries
- Naïve approach (large input space), but effective
- Open challenges
  - Monitoring and bug oracle (fault or error detection)
  - HW fuzzing
  - Protocol inference
  - …
- Standards require “fuzz testing”, and “reliability testing”
- Should also be done during development

Resources

- Hanno Boeck: “How Heartbleed could've been found”. https://blog.hboeck.de/archives/868-How-Heartbleed-couldve-been-found.html, April 2015.
- Xiaogang Zhu et al., ‘Fuzzing: A Survey for Roadmap’, *ACM Computing Surveys* 54, https://doi.org/10.1145/3512345.
- Manes et al.: „The Art, Science, and Engineering of Fuzzing: A Survey“, arXiv 1812.00140, 2019.
- Miller et al.: “An Empirical Study of the Reliability of UNIX Utilities”, Commun. ACM 33(12), 1990.
- Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler: „The Fuzzing Book“. https://www.fuzzingbook.org/
- Jun Li, Bodong Zhao, and Chao Zhang, ‘Fuzzing: A Survey’, *Cybersecurity* 1, no. 1: 1–13, https://doi.org/10.1186/s42400-018-0002-y.
- Marcel Böhme, Cristian Cadar, and Abhik Roychoudhury, ‘Fuzzing: Challenges and Reflections’, *IEEE Software* 38, no. 3: 79–86, https://doi.org/10.1109/MS.2020.3016773.
- Patrice Godefroid, ‘Fuzzing: Hack, Art, and Science’, *Communications of the ACM* 63, no. 2: 70–76, https://doi.org/10.1145/3363824.
- Valentin J. M. Manes et al., ‘The Art, Science, and Engineering of Fuzzing: A Survey’, https://doi.org/10.48550/arXiv.1812.00140.
- Hongliang Liang et al., ‘Fuzzing: State of the Art’, *IEEE Transactions on Reliability* 67, no. 3: 1199–1218, https://doi.org/10.1109/TR.2018.2834476.
- Recent Papers Related to Fuzzing - https://github.com/wcventure/FuzzingPaper