Tutorial

Automated Security Testing with Fuzzing

Christopher Huth

BOSCH

about:me

- At Bosch since 07/2014
- Product Security Officer for Corporate Research
- Senior cybersecurity expert and research activity lead
  - (Embedded) Fuzzing
  - Red Team
  - Physical Layer Security
  - DevSecOps
  - Cybersecurity in general

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

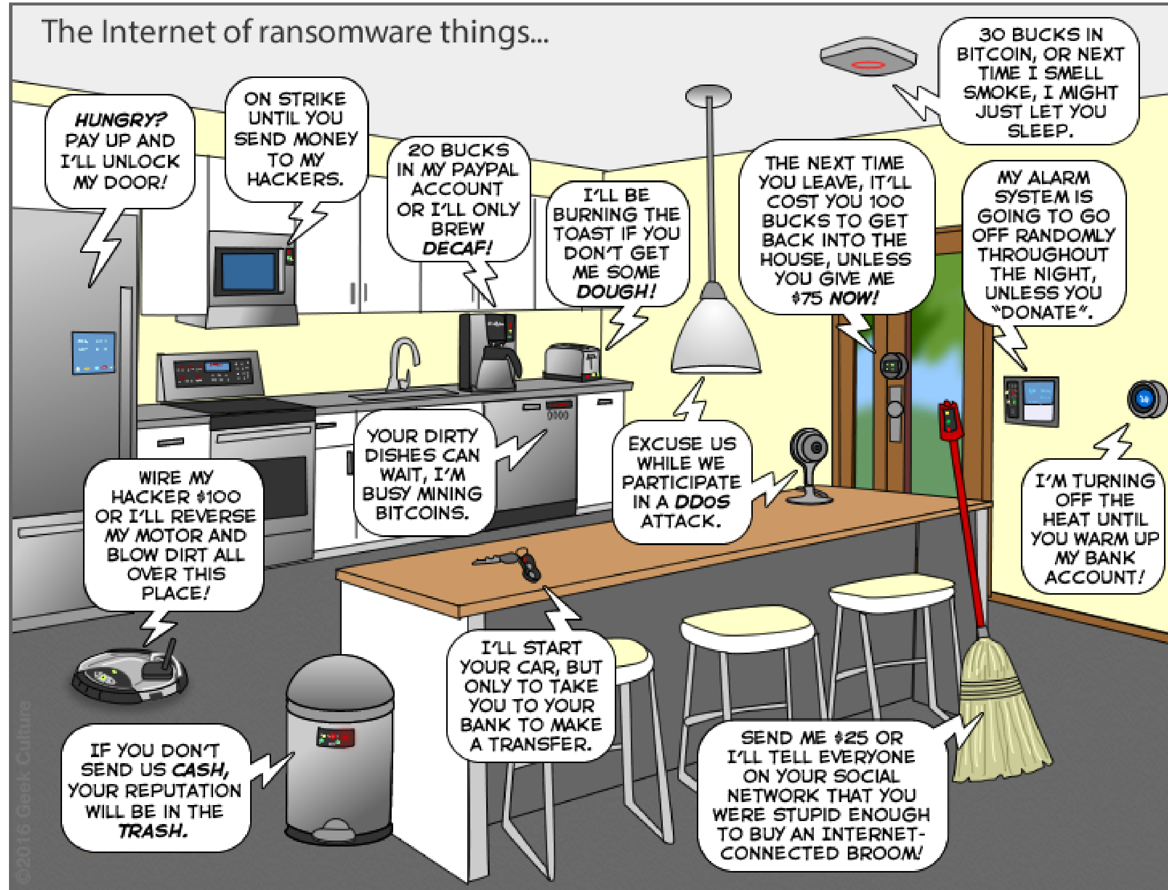
Agenda

1. **Motivation**
2. Theory
    1. What is fuzzing?
    2. How to talk about fuzzing?
    3. What can be fuzzed?
    4. What fuzzing types are there?
3. Practice
    1. Toy example – set up a fuzz test
    2. Real world example – optimize a fuzz test
4. Challenges and good practices

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Motivation

The Joy of Tech™ by Nitrozac & Snaggy



You can help us keep the comics coming by becoming a patron! www.patreon/joyoftech

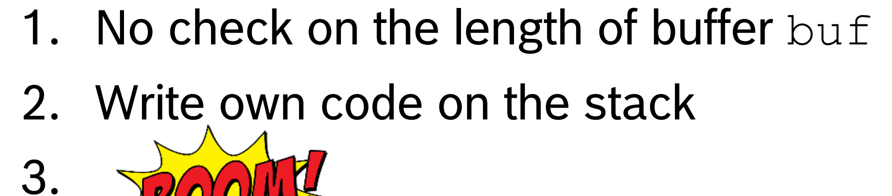
joyoftech.com

One of the first bugs discovered by fuzzing (>30 years old)

Example Vulnerability

```
while ((cc = getch()) != c)
{
  buf[i++] = cc;
  ...
}
```

1. No check on the length of buffer `buf`
2. Write own code on the stack
3.



- With testing for positive cases, bug can remain hidden
  - E.g. a unit test with a normal input executes both lines => 100% coverage

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Agenda

1. Motivation
2. Theory
   1. **What is fuzzing?**
   2. How to talk about fuzzing?
   3. What can be fuzzed?
   4. What fuzzing types are there?
3. Practice
   1. Toy example – set up a fuzz test
   2. Real world example – optimize a fuzz test
4. Challenges and good practices

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Fuzzing in Software Engineering and System Security

- Software engineering consisting of
  - requirements, design, construction, **testing**, and maintenance
    - **Manual testing** bound to human resources, e.g. time
    - **Automated testing** limited in creativity

A **test case** (or **test**) is an input and an expected result.
Test generation, test execution, and checking test result can be automated very well.

- System security
  - **Security by proof (static)**
    - Software is *provably secure*
      - certain bugs *cannot occur*
      - always *acts according to a model*
    - Requires (expensive) mathematical proof

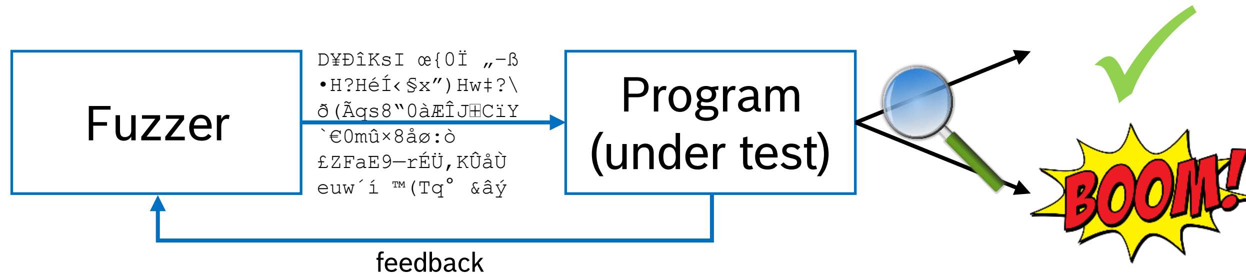
"Beware of bugs in the above code; I have only proved it correct, not tried it." – Donald Knuth

  - **Security by testing (dynamic)**
    - Software is *tested*
      - Successful attacks are *unlikely*
      - Successful attacks have *high complexity*
    - *No guarantee* that software is bug free

"Testing shows the presence, not the absence of bugs." – Edsger W. Dijkstra

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

What is Fuzzing?



- Dynamic (code runs)
- Bug must be executed
- Bug should be observable
- (Semi-)random* input

* Input should be 'valid enough' to pass early sanitizations and 'invalid enough' to explore unknown test cases.

- Fuzzing was coined in 1989, when Miller *et al.* used a random testing tool to investigate the reliability of UNIX tools.
- Fuzzing automatically generates
  - random data
  - and provides this data as input to a software under test.
  - Software under test is monitored, e.g. for crashes.

An Empirical Study of the Reliability

of

UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

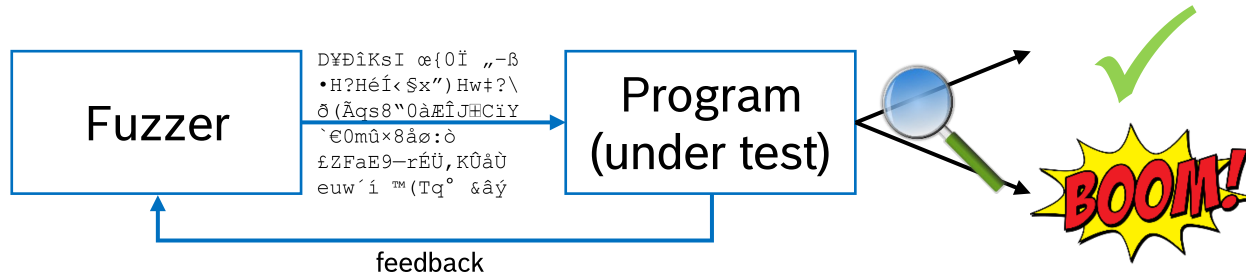
Bryan So
so@cs.wisc.edu

Summary

Operating system facilities, such as the kernel and utility programs, are typically assumed to be reliable. In our recent experiments, we have been able to crash 25-33% of the utility programs on any version of UNIX that was tested. This report describes these tests and an analysis of the program bugs that caused the crashes.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

What is Fuzzing?



- Dynamic (code runs)
- Bug must be executed
- Bug should be observable
- (Semi-)random* input

* Input should be 'valid enough' to pass early sanitizations and 'invalid enough' to explore unknown test cases.

- Fuzzing became *really* popular in the wild with AFL (2014 - 2017) written by Michal Zalewski.
- *Coverage-guided* fuzzing automatically generates
  - unexpected, malformed, or random data
  - and provides this data as input to a software under test.
  - Software under test is monitored, e.g. for crashes or hangs.

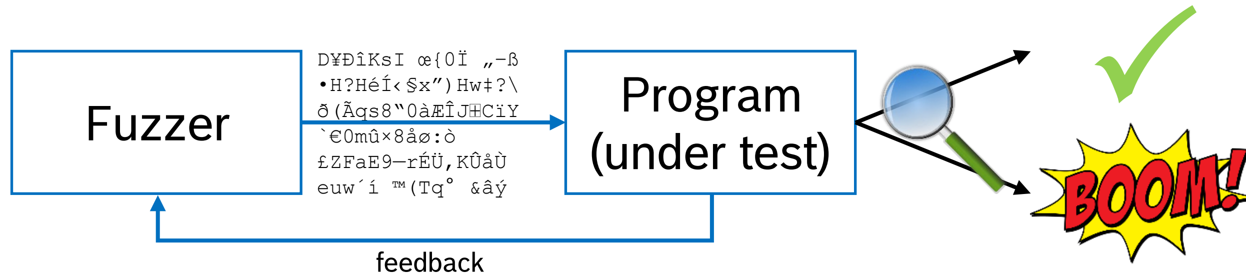
```
                 american fuzzy lop 0.47b (readpng)

  process timing                                      overall results
         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
  cycle progress                       map coverage
   now processing : 38 (19.49%)          map density : 1217 (7.43%)
  paths timed out : 0 (0.00%)         count coverage : 2.55 bits/tuple
  stage progress                       findings in depth
   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)
  fuzzing strategy yields                           path geometry
    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
```

https://en.wikipedia.org/wiki/American_Fuzzy_Lop

https://lcamtuf.coredump.cx/afl/

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

What is Fuzzing?



- Dynamic (code runs)
- Bug must be executed
- Bug should be observable
- (Semi-)random* input

* Input should be 'valid enough' to pass early sanitizations and 'invalid enough' to explore unknown test cases.

- Overall goal is to validate a robust program behavior.
  - When a program accepts data from an untrusted input (or faulty in general), unwanted and observable behavior should be avoided.
  - In more detail, fuzzing metrics, such as code coverage and time, can be maximized.
- Fuzz testing can detect bugs which can lead to vulnerabilities, i.e., discovers symptoms for exploitable bugs.
- The generated input that triggers a bug is saved, and thus provides a reproducible test case, e.g., for regression testing.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

What is Fuzzing? – toy example

```
1   #include <stdio.h>
2
3   #define BUFFERMAXSIZE 5
4
5   int main( ) {
6       char buffer[BUFFERMAXSIZE];
7       int bufferIndex = 0;
8
9       while(bufferIndex < BUFFERMAXSIZE){
10          buffer[bufferIndex] = getc(stdin);
11          bufferIndex++;
12      }
13
14      if (bufferIndex >= 3){
15          if (buffer[0] == 'B'){
16              if (buffer[1] == 'U'){
17                  if (buffer[2] == 'G'){
18                      if (buffer[3] == '!'){
19                          printf("Memory leak found!\n");
20                          printf("%c", buffer[BUFFERMAXSIZE+10]);
21                      }
22                  }
23              }
24          }
25      }
26
27      printf("SUCCESSFUL TERMINATION!\n");
28      return 0;
29  }
```

Without feedback (blackbox), 1h

```
1   :  #include <stdio.h>
2   :
3   :  #define BUFFERMAXSIZE 5
4   :
5   1  :  int main( ) {
6   1  :      char buffer[BUFFERMAXSIZE];
7   1  :      int bufferIndex = 0;
8   :
9   11 :      while(bufferIndex < BUFFERMAXSIZE){
10  5  :          buffer[bufferIndex] = getc(stdin);
11  5  :          bufferIndex++;
12  :      }
13  :
14  1  :      if (bufferIndex >= 3){
15  1  :          if (buffer[0] == 'B'){
16  0  :              if (buffer[1] == 'U'){
17  0  :                  if (buffer[2] == 'G'){
18  0  :                      if (buffer[3] == '!'){
19  0  :                          printf("Memory leak found!\n");
20  0  :                          printf("%c", buffer[BUFFERMAXSIZE+10]);
21  :                      }
22  :                  }
23  :              }
24  :          }
25  :      }
26  :
27  1  :      printf("SUCCESSFUL TERMINATION!\n");
28  1  :      return 0;
29  :  }
```

With feedback (fuzzer learns), 5m + bug found

```
1   :  #include <stdio.h>
2   :
3   :  #define BUFFERMAXSIZE 5
4   :
5   4  :  int main( ) {
6   4  :      char buffer[BUFFERMAXSIZE];
7   4  :      int bufferIndex = 0;
8   :
9   44 :      while(bufferIndex < BUFFERMAXSIZE){
10  20 :          buffer[bufferIndex] = getc(stdin);
11  20 :          bufferIndex++;
12  :      }
13  :
14  4  :      if (bufferIndex >= 3){
15  4  :          if (buffer[0] == 'B'){
16  3  :              if (buffer[1] == 'U'){
17  2  :                  if (buffer[2] == 'G'){
18  1  :                      if (buffer[3] == '!'){
19  0  :                          printf("Memory leak found!\n");
20  0  :                          printf("%c", buffer[BUFFERMAXSIZE+10]);
21  :                      }
22  :                  }
23  :              }
24  :          }
25  :      }
26  :
27  4  :      printf("SUCCESSFUL TERMINATION!\n");
28  4  :      return 0;
29  :  }
```


© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Fuzzing Mythbusting

- Mythbusting [1]
  - **Fuzzing is ~~only~~ for security researchers,~~or~~ security teams**, developers, and testers; i.e. everyone
  - **Fuzzing ~~only~~ finds ~~security vulnerabilities~~** all kinds of bugs
  - **We ~~don't~~ need fuzzers if our project is well unit-tested**
  - ~~**Our project is secure**~~ **if there are no open bugs**, they haven't been found yet.

- Side notes
  - Fuzzing is *THE* bug-finding test method. Championed by Google[2] and Microsoft[3]
  - "Fuzzing is an art" – Easy to get into, hard to master

[1] Points taken from Arya and Chang "ClusterFuzz: Fuzzing at Google Scale", blackhat Europe 2019
[2] Fuzzing found more than 25.000 (ca. 80%) bugs in Chrome and ~22.500 bugs in 340 Open Source Projects
[3] Every Microsoft software is fuzzed; Microsoft offers fuzzing-as-a-service

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Agenda

1. Motivation
2. Theory
   1. What is fuzzing?
   2. **How to talk about fuzzing?**
   3. What can be fuzzed?
   4. What fuzzing types are there?
3. Practice
   1. Toy example – set up a fuzz test
   2. Real world example – optimize a fuzz test
4. Challenges and good practices

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Coverage-guided Fuzzing with AFL

0. Set up test system and write harness.

1. Test target is instrumented, so that coverage during runtime is visible.

2. Fuzzer takes input from queue and applies a chosen mutation.

3. Current test case is injected in software under test via harness.

4. Software under test is observed for unwanted behavior (e.g. crash).

5. Coverage is collected and fed back to fuzzer.

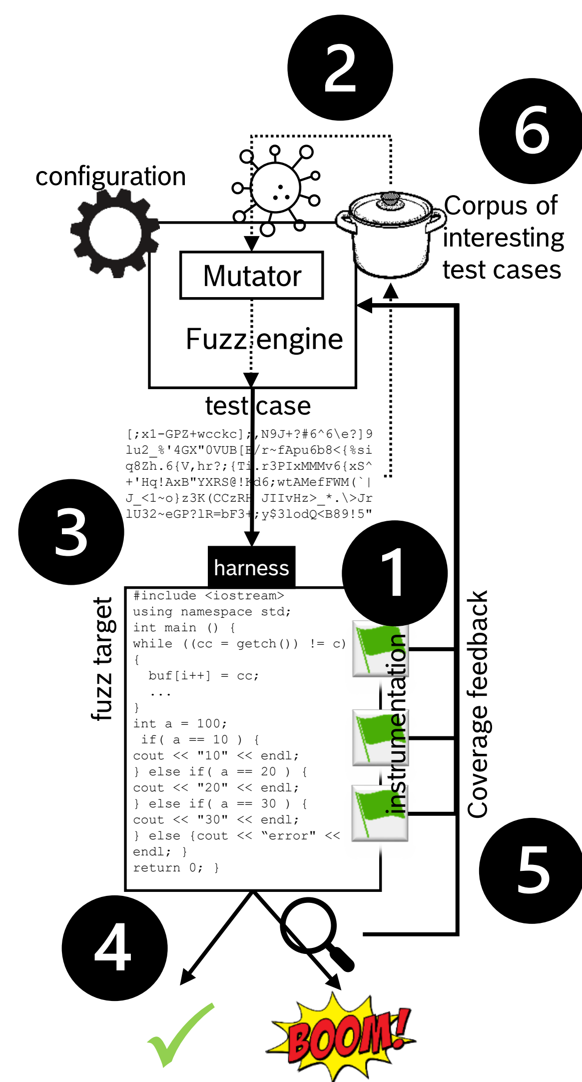
6. Fuzzer updates fitness values of test cases and repeats at 2.

Inputs and mutations are chosen based on some learning, e.g. fitness values.

Inputs gets better and better over time.

Fuzzing is not functional security testing; no assessment of presence/effectiveness of security functions.

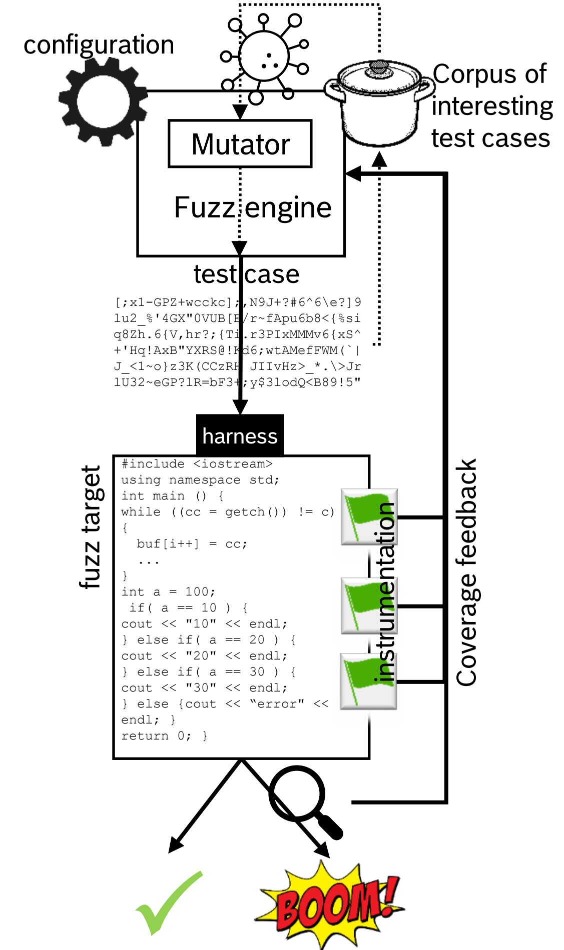
Fuzzing is not penetration testing; think more of fuzzing like robustness testing.



Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Terminology

- **Fuzzing** or **fuzz testing** is the overall term.
- **Fuzzing engine a.k.a. fuzzer** is a program that produces an infinite stream of inputs for a target and orchestrates the execution.
- **Fuzz target** is a binary, a library, an API, or rather anything that can consume bytes for input.
- **Glue code** or **wrapper** or **harness** connects a fuzzer to a fuzz target.
- **Input or test case** is a sequence of bytes that can be fed to a target. The input can be an arbitrary bag of bytes, or some structured data, e.g. serialized proto.
- **Coverage** is some information about the behavior of the target when it executes a given input.
- **Instrumentation** is used to make coverage metric observable, e.g. during compilation.
- **Mutator** is a function that takes bytes as input and outputs a small random mutation of the input.
- **Corpus (*plural: corpora*)** is a set of inputs. Initial inputs are **seeds**.
- **Configurations** tune a fuzzer or campaign for a fuzz target.

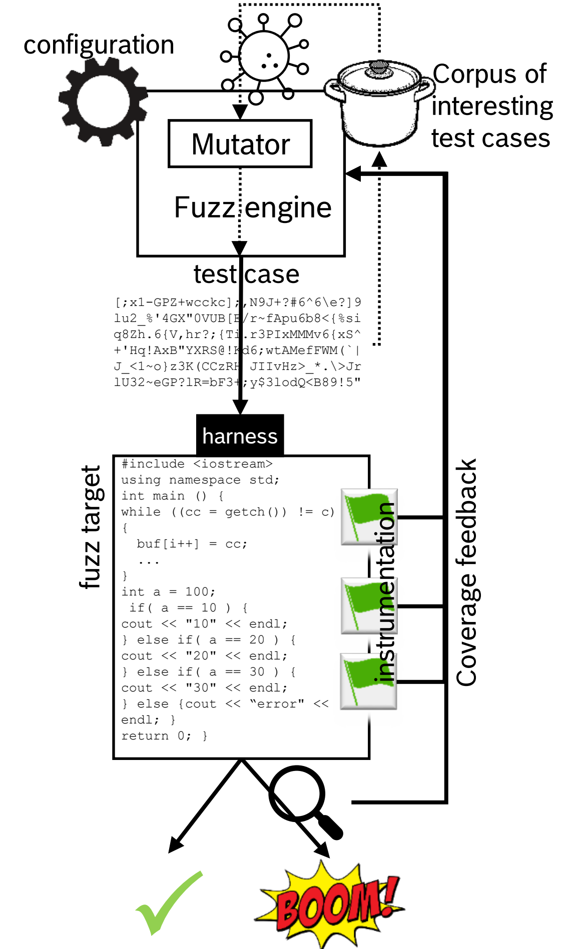


https://github.com/google/centipede#terminology

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Metrics

- Measuring different fuzzers, or fuzzing runs, is hard, because fuzzers are usually non-deterministic.
- An ideal metric would be the **number of** (possibly exploitable) **bugs** identified by crashing inputs, but
  - buggy code locations have to be reached.
  - software needs to be in the right state, so that bug can be triggered.
  - bug needs to be observable.
- Other metrics are:
  - Crashes or hangs
  - Total runtime / timeout
  - Coverage[1], such as line coverage, block coverage, edge coverage, branch coverage, etc.
  - Generated test cases
- For embedded:
  - Power consumption
  - Error codes
  - timeouts



[1] "… code coverage that is achieved is a strong predictor of the bug finding ability of a fuzzer (i.e., there is a strong correlation)", Liyanage et al., *False Peaks: On the Estimation of Fuzzing Effectiveness*

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Agenda

1. Motivation
2. Theory
    1. What is fuzzing?
    2. How to talk about fuzzing?
    3. **What can be fuzzed?**
    4. What fuzzing types are there?
3. Practice
    1. Toy example – set up a fuzz test
    2. Real world example – optimize a fuzz test
4. Challenges and good practices

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

What can be fuzzed?

Anything can be fuzzed that consumes untrusted, complex inputs.

- (Crypto-) Functions
- Parsers of any kind
- Media codecs
- Network protocols
- Compression
- Formatted output
- Compilers and interpreters
- Regular expression matchers
- Text processing
- Databases
- Browsers, text editors
- OS Kernels, drivers, supervisors, VMs

What can a fuzzer detect?

- Crashes during runtime.
  - NULL dereferences, uncaught exceptions, div-by-zero, ...
- Additionally, with sanitizers, a fuzzer can detect
  - use-after-free, buffer overflows
  - uses of uninitialized memory, memory leaks
  - data races, deadlocks
  - int/float overflows, bitwise shifts by invalid amount
- Resource usage bugs
  - Memory exhaustion, hangs or infinite loops, infinite recursion (stack overflows)

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Agenda

1. Motivation
2. Theory
   1. What is fuzzing?
   2. How to talk about fuzzing?
   3. What can be fuzzed?
   4. **What fuzzing types are there?**
3. Practice
   1. Toy example – set up a fuzz test
   2. Real world example – optimize a fuzz test
4. Challenges and good practices

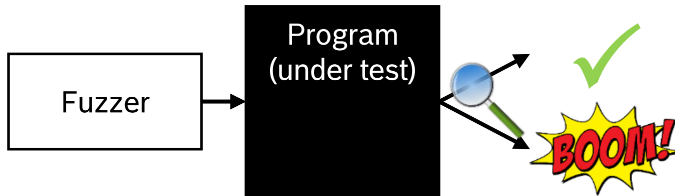
Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Fuzzing Types

Example: Fuzz some source code, i.e. no protocol

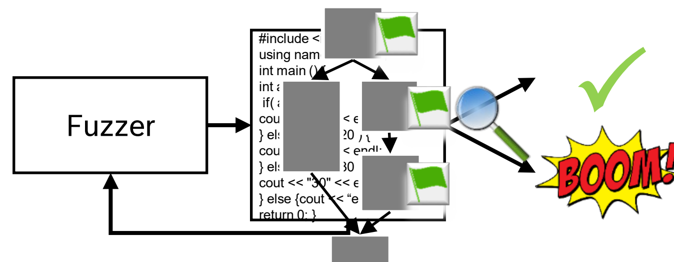
Black-box setting ■

- Only requires the software under test to execute
- Assuming no source code
- Observes whether the program crashed (if at all)



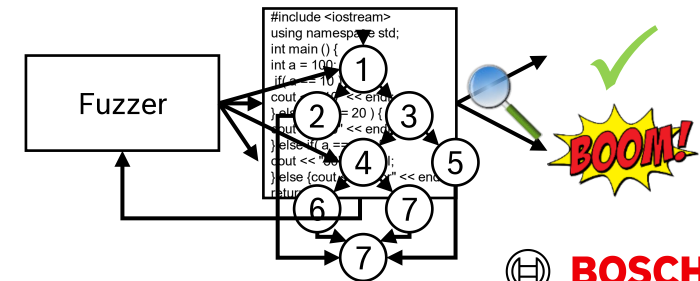
Grey-box setting

- Mixture of black-box and white-box
- Lightweight instrumentation
  - Trace the program structure during monitoring



White-box setting □

- Heavy-weight program analysis, e.g. with additional symbolic execution
- Available source code
- Observe (and modify) semantics of a program's source code (including the binary)



Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Recall toy example

```c
1  #include <stdio.h>
2
3  #define BUFFERMAXSIZE 5
4
5  int main( ) {
6      char buffer[BUFFERMAXSIZE];
7      int bufferIndex = 0;
8
9      while(bufferIndex < BUFFERMAXSIZE){
10         buffer[bufferIndex] = getc(stdin);
11         bufferIndex++;
12     }
13
14     if (bufferIndex >= 3){
15         if (buffer[0] == 'B'){
16             if (buffer[1] == 'U'){
17                 if (buffer[2] == 'G'){
18                     if (buffer[3] == '!'){
19                         printf("Memory leak found!\n");
20                         printf("%c", buffer[BUFFERMAXSIZE+10]);
21                     }
22                 }
23             }
24         }
25     }
26
27     printf("SUCCESSFUL TERMINATION!\n");
28     return 0;
29 }
```

Without feedback (blackbox), 1h

```
1         : #include <stdio.h>
2         :
3         : #define BUFFERMAXSIZE 5
4         :
5       1 : int main( ) {
6       1 :         char buffer[BUFFERMAXSIZE];
7       1 :         int bufferIndex = 0;
8         :
9      11 :         while(bufferIndex < BUFFERMAXSIZE){
10      5 :                 buffer[bufferIndex] = getc(stdin);
11      5 :                 bufferIndex++;
12        :         }
13        :
14      1 :         if (bufferIndex >= 3){
15      1 :                 if (buffer[0] == 'B'){
16      0 :                         if (buffer[1] == 'U'){
17      0 :                                 if (buffer[2] == 'G'){
18      0 :                                         if (buffer[3] == '!'){
19      0 :                                                 printf("Memory leak found!\n");
20      0 :                                                 printf("%c", buffer[BUFFERMAXSIZE+10]);
21        :                                         }
22        :                                 }
23        :                         }
24        :                 }
25        :         }
26        :
27      1 :         printf("SUCCESSFUL TERMINATION!\n");
28      1 :         return 0;
29        : }
```

With feedback (fuzzer learns), 5m + bug found

```
1         : #include <stdio.h>
2         :
3         : #define BUFFERMAXSIZE 5
4         :
5       4 : int main( ) {
6       4 :         char buffer[BUFFERMAXSIZE];
7       4 :         int bufferIndex = 0;
8         :
9      44 :         while(bufferIndex < BUFFERMAXSIZE){
10     20 :                 buffer[bufferIndex] = getc(stdin);
11     20 :                 bufferIndex++;
12        :         }
13        :
14      4 :         if (bufferIndex >= 3){
15      4 :                 if (buffer[0] == 'B'){
16      3 :                         if (buffer[1] == 'U'){
17      2 :                                 if (buffer[2] == 'G'){
18      1 :                                         if (buffer[3] == '!'){
19      0 :                                                 printf("Memory leak found!\n");
20      0 :                                                 printf("%c", buffer[BUFFERMAXSIZE+10]);
21        :                                         }
22        :                                 }
23        :                         }
24        :                 }
25        :         }
26        :
27      4 :         printf("SUCCESSFUL TERMINATION!\n");
28      4 :         return 0;
29        : }
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Fuzzing Types

Example: Fuzz some source code, i.e. no protocol

Black-box setting ■

- To generate test case "BUG!", the fuzzer has to guess.
- One char can have 2^8=256 values; so 256^4=**4 billion** length-four-strings need to be generated.
- Test cases can be generated very fast, e.g. by pulling random numbers, but no feedback.

Grey-box setting ■

- To generate test case "BUG!", the fuzzer has feedback.

#Seeds	From	To	Expected #input required
1	????	B???	$(1 * 4^{-1} * 2^{-8})^{-1} = 1024$
2	B???	BU??	$(1/2 * 4^{-1} * 2^{-8})^{-1} = 2048$
3	BU??	BUG?	$(1/3 * 4^{-1} * 2^{-8})^{-1} = 3072$
4	BUG?	BUG!	$(1/4 * 4^{-1} * 2^{-8})^{-1} = 4096$
5			Total: **10240** inputs

Böhme et al., *Estimating Residual Risk in Greybox Fuzzing*

- Some overhead in observing fuzz target and test case generation.

White-box setting □

- To generate test case "BUG!", all paths can be enumerated.
- Symbolic (or concolic) execution calculates a test case to cover each path, i.e. **5 inputs.**
- Can be slow and even infeasible for large programs, due to exponential path explosion.
- To cope, paths are approximated and hence can be overapproximated.

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

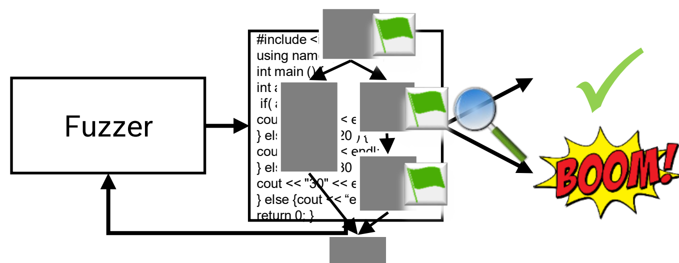
BOSCH

Fuzzing Types
Protocol and source code fuzzing

Protocol fuzzing is active research and more sophisticated (coverage-guided, grey-box, state-approximating) tools appear, like AFLnwe, AFLNet, and StateAFL.

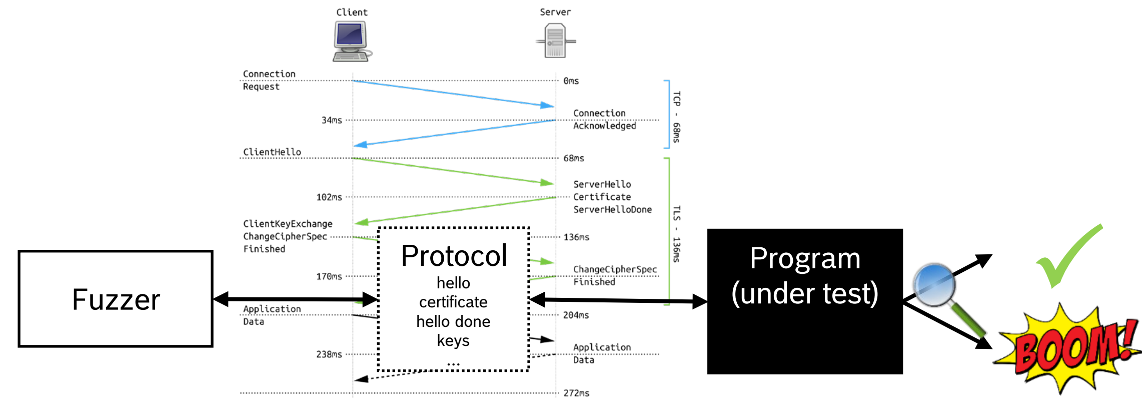
Source code fuzzing

- Program states are secondary
- Good measure is e.g. code coverage



Protocol fuzzing

- Focuses on communication of a program and its states
- Messages are delayed, intercepted, replayed, randomized, forged, etc.
- Fuzzer can be a Man-in-the-Middle
- Protocol fuzzing is usually a black-box test



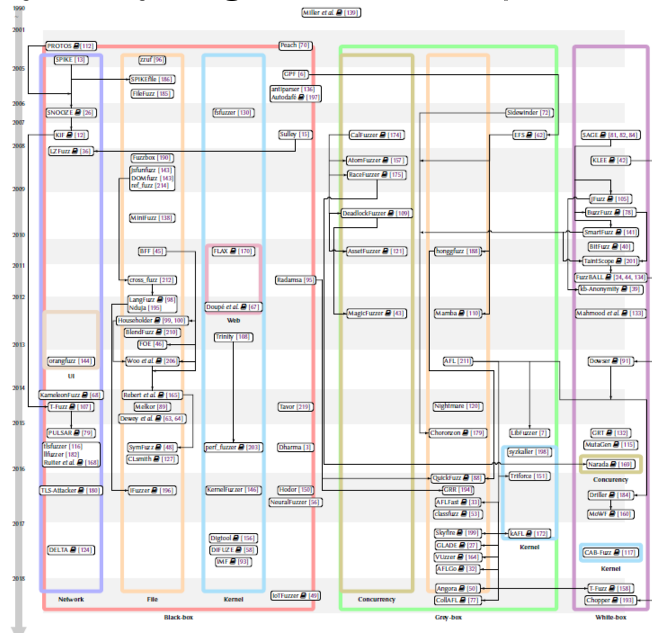
https://en.wikipedia.org/wiki/Transport_Layer_Security

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

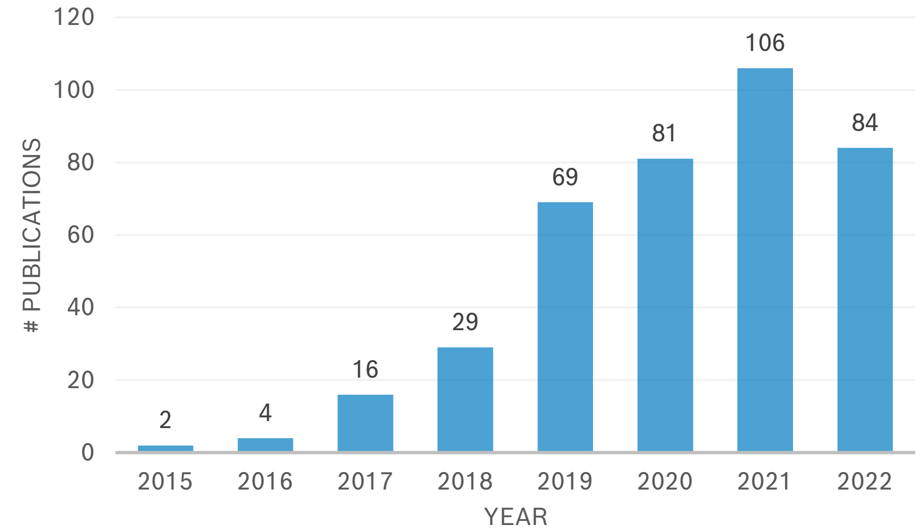
BOSCH

Tools and Publications (far from complete)

- For nearly every target there is a special fuzzing tool.



- Number of fuzzing publications as of 2023-01-20[2]



[2]https://github.com/wcventure/FuzzingPaper/blob/master/README.md

[1]Fuzzing: Art, Science, and Engineering, VALENTIN J.M. MANES, KAIST CSRC, Korea, HYUNGSEOK HAN, KAIST, Korea, CHOONGWOO HAN, Naver Corp., Korea, SANG KIL CHA∗, KAIST, Korea, MANUEL EGELE, Boston University, USA, EDWARD J. SCHWARTZ, Carnegie Mellon University/Software Engineering Institute, USA, MAVERICK WOO, Carnegie Mellon University, USA

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Agenda

1. Motivation
2. Theory
    1. What is fuzzing?
    2. How to talk about fuzzing?
    3. What can be fuzzed?
    4. What fuzzing types are there?
3. Practice
    1. **Toy example – set up a fuzz test**
    2. Real world example – optimize a fuzz test
4. Challenges and good practices

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz

- https://sourcecode-g1.dev.bosch.com/projects/FUZZINGTUTORIAL/repos/ad-curriculum/browse
- We look at two of the most popular fuzzers, AFL++ (a community-driven successor of AFL) and libFuzzer
  - Both are coverage-guided grey-box source code fuzzers

- https://github.com/AFLplusplus/AFLplusplus
- Fork of Google’s AFL
- Active community of researchers
- Active development

- https://llvm.org/docs/LibFuzzer.html
- Part of LLVM compiler suite
- Active development by Google

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – toy example

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char buffer[8];
    int bufferIndex = 0;
    char endOfInput = '\0';
    char currentInput;

    while ((currentInput = getc(stdin)) != endOfInput) {
        buffer[bufferIndex] = currentInput;
        bufferIndex++;
    }

    while (bufferIndex >= 0) {
        printf("buffer[%i]: %c\n", bufferIndex, buffer[bufferIndex]);
        bufferIndex--;
    }
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

```
$ gcc -g -Wall -o toy-example-buffer-overflow toy-example-buffer-overflow.cpp
$ echo -e "0123456\0" > validtest
$ ./toy-example-buffer-overflow < validtest
buffer[7]:
buffer[6]: 6
buffer[5]: 5
buffer[4]: 4
buffer[3]: 3
buffer[2]: 2
buffer[1]: 1
buffer[0]: 0
SUCCESSFUL TERMINATION!
```


© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz – toy example

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char buffer[8];
    int bufferIndex = 0;
    char endOfInput = '\0';
    char currentInput;

    while ((currentInput = getc(stdin)) != endOfInput) {
        buffer[bufferIndex] = currentInput;
        bufferIndex++;
    }

    while (bufferIndex >= 0) {
        printf("buffer[%i]: %c\n", bufferIndex, buffer[bufferIndex]);
        bufferIndex--;
    }
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

```
$ python -c "print('A' * 20000)" > lotsofA
$ ./toy-example-buffer-overflow < lotsofA
Segmentation fault (core dumped)
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – black-box

```
   american fuzzy lop ++2.60d (toy-example-buffer-overflow) [explore] {0}
┌─ process timing ───────────────────────────────────────┬─ overall results ──┐
│        run time : 0 days, 0 hrs, 0 min, 9 sec         │  cycles done : 0   │
│   last new path : n/a (non-instrumented mode)          │  total paths : 1   │
│ last uniq crash : 0 days, 0 hrs, 0 min, 0 sec         │ uniq crashes : 279 │
│  last uniq hang : none seen yet                        │   uniq hangs : 0   │
├─ cycle progress ───────────────────┬─ map coverage ─┴────────────────────┤
│  now processing : 0*0 (0.0%)       │    map density : 0.00% / 0.00%      │
│ paths timed out : 0 (0.00%)        │ count coverage : 0.00 bits/tuple    │
├─ stage progress ───────────────────┼─ findings in depth ──────────────────┤
│  now trying : havoc                │ favored paths : 0 (0.00%)            │
│ stage execs : 60/1024 (5.86%)      │  new edges on : 0 (0.00%)            │
│ total execs : 1452                 │ total crashes : 279 (279 unique)     │
│  exec speed : 265.7/sec            │  total tmouts : 9 (9 unique)         │
├─ fuzzing strategy yields ──────────┴───────────────┬─ path geometry ──────┤
│   bit flips : 8/72, 9/71, 11/69                    │    levels : 1        │
│  byte flips : 1/9, 2/8, 2/6                        │   pending : 1        │
│ arithmetics : 56/504, 94/94, 25/25                 │  pend fav : 0        │
│  known ints : 2/49, 13/203, 16/264                 │ own finds : 0        │
│  dictionary : 0/0, 0/0, 0/0                        │  imported : n/a      │
│   havoc/rad : 0/0, 0/0, 0/0                        │ stability : n/a      │
│   py/custom : 0/0, 0/0                             ├──────────────────────┘
│        trim : n/a, 0.00%                           │          [cpu000:171%]
└────────────────────────────────────────────────────┘^C
```

```
$ mkdir in out
$ cp validtest in/
$ afl-fuzz -i in/ -o out/ -n ./toy-example-buffer-overflow
```

black-box fuzzing

```
$ stat -c %s out/crashes/id\:000240*
1
```

```
$ echo "" | ./toy-example-buffer-overflow
Segmentation fault
```


Christopher Huth | 2023-01-25

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – more paths

```c
#include <stdio.h>

#define BUFFERSIZE 5

int main( ) {

    char buffer[BUFFERSIZE];
    int bufferIndex = 0;

    while(bufferIndex < BUFFERSIZE){
        buffer[bufferIndex] = getc(stdin);
        bufferIndex++;
    }

    if (bufferIndex >= 3){
        if (buffer[0] == 'B'){
            if (buffer[1] == 'U'){
                if (buffer[2] == 'G'){
                    if (buffer[3] == 'G'){
                        printf ("Address crash condition found!\n");
                        *((int *)0) = 0; // do some crashing
                    }
                }
            }
        }
    }

    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

```
$ echo "" | ./more-paths-stackoverflow
SUCCESSFUL TERMINATION!
$ echo "12345678" | ./more-paths-stackoverflow
SUCCESSFUL TERMINATION!
$ ./more-paths-stackoverflow < lotsofA
SUCCESSFUL TERMINATION!

$ echo "BUGG" | ./more-paths-stackoverflow
Address crash condition found!
Segmentation fault
```


© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – fuzz more paths

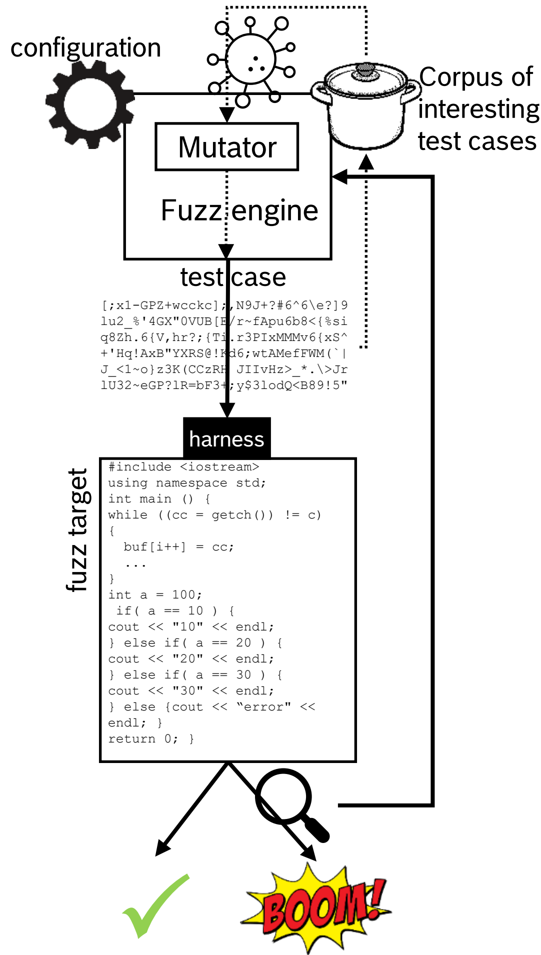
```
  american fuzzy lop ++2.60d (more-paths-stackoverflow) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 58 min, 52 sec     │  cycles done : 3462
│   last new path : n/a (non-instrumented mode)       │  total paths : 1
│ last uniq crash : none seen yet                     │ uniq crashes : 0
│  last uniq hang : none seen yet                     │   uniq hangs : 0
├─ cycle progress ─────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 0*3462 (0.0%)  │    map density : 0.00% / 0.00%
│ paths timed out : 0 (0.00%)      │ count coverage : 0.00 bits/tuple
├─ stage progress ─────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc              │ favored paths : 0 (0.00%)
│ stage execs : 10/256 (3.91%)     │  new edges on : 0 (0.00%)
│ total execs : 891k               │ total crashes : 0 (0 unique)
│  exec speed : 273.2/sec          │  total tmouts : 4653 (4653 unique)
├─ fuzzing strategy yields ────────┴─────────────┬─ path geometry ───────┤
│   bit flips : 0/8, 0/7, 0/5                    │    levels : 1
│  byte flips : 0/1, 0/0, 0/0                    │   pending : 0
│ arithmetics : 0/56, 0/0, 0/0                   │  pend fav : 0
│  known ints : 0/4, 0/0, 0/0                    │ own finds : 0
│  dictionary : 0/0, 0/0, 0/0                    │  imported : n/a
│   havoc/rad : 0/887k, 0/0, 0/0                 │ stability : n/a
│   py/custom : 0/0, 0/0                         ├───────────────────────┘
│        trim : n/a, 0.00%                       │          [cpu000:160%]
└────────────────────────────────────────────────┘
```

```
$ afl-fuzz -i in/ -o out/ -n ./more-paths-stackoverflow
```

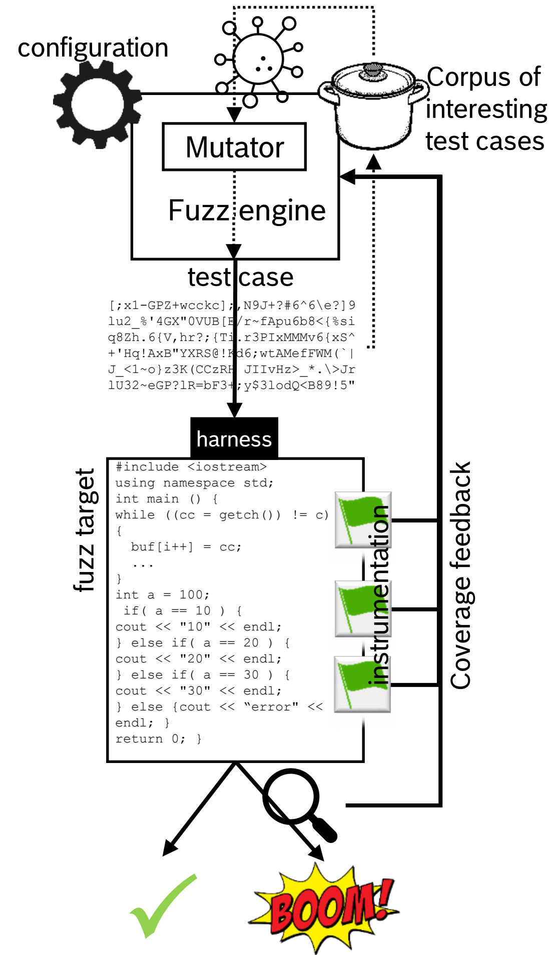
black-box fuzzing

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Terminology (recap)



instrumentation
during compilation



© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let's fuzz – toy example with more paths

```
        american fuzzy lop ++2.60d (more-paths-stackoverflow) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 15 min, 49 sec      │  cycles done : 1939  │
│   last new path : 0 days, 0 hrs, 0 min, 14 sec       │  total paths : 4     │
│ last uniq crash : 0 days, 0 hrs, 0 min, 13 sec       │ uniq crashes : 1     │
│  last uniq hang : none seen yet                      │   uniq hangs : 0     │
├─ cycle progress ───────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 3.7 (75.0%)  │    map density : 0.01% / 0.02%        │
│ paths timed out : 0 (0.00%)    │ count coverage : 1.00 bits/tuple      │
├─ stage progress ───────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc            │ favored paths : 4 (100.00%)           │
│ stage execs : 506/1024 (49.41%)│  new edges on : 4 (100.00%)           │
│ total execs : 1.38M            │ total crashes : 6 (1 unique)          │
│  exec speed : 1170/sec         │  total tmouts : 18 (4 unique)         │
├─ fuzzing strategy yields ──────┴─────────┬─ path geometry ─────────────┤
│   bit flips : 0/64, 0/60, 0/52           │    levels : 4               │
│  byte flips : 0/8, 0/4, 0/1              │   pending : 0               │
│ arithmetics : 1/445, 0/0, 0/0            │  pend fav : 0               │
│  known ints : 0/50, 0/112, 0/44          │ own finds : 3               │
│  dictionary : 0/0, 0/0, 0/0              │  imported : n/a             │
│   havoc/rad : 3/1.38M, 0/0, 0/0          │ stability : 100.00%         │
│   py/custom : 0/0, 0/0                   ├─────────────────────────────┘
│        trim : n/a, 0.00%                 │             [cpu000:195%]
└──────────────────────────────────────────┘^C
```

```
$ afl-gcc -g -Wall -o toy-example-buffer-overflow toy-example-buffer-overflow.cpp
```

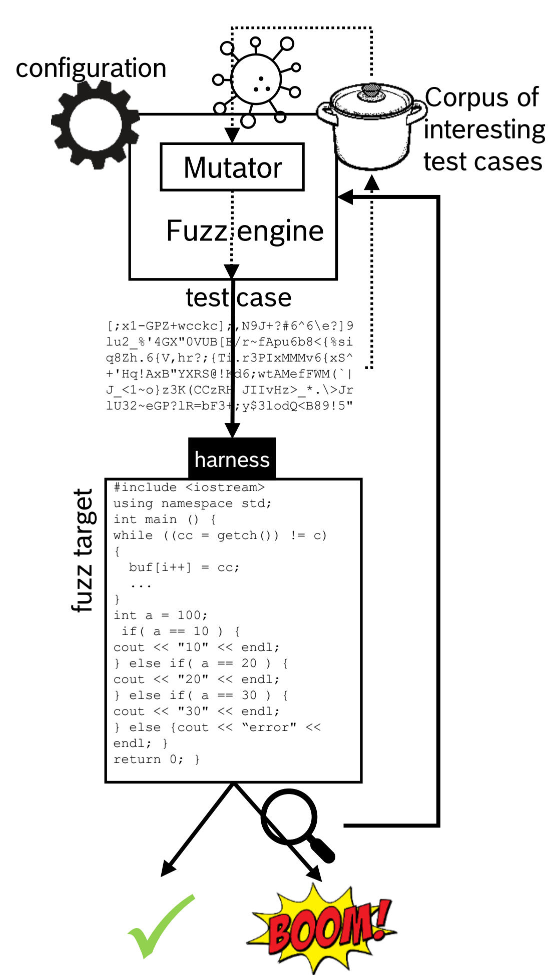
instrumentation

```
$ echo "" > in/empty

$ afl-fuzz -i in/ -o out/ ./more-paths-stackoverflow
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Terminology (recap) – instrumentation



Recall:
Compilers (language -> assembly),
assembler (assembly -> object code),
linker (object code -> executable/library).

For example, `gcc` by default uses `GNU as` assembler. `afl-gcc` is a wrapper around `gcc` which uses `afl-as` by symlinking `afl-as` as `as` and adding the directory to compiler search path via `-B`.
https://tunnelshade.in/blog/2018/01/afl-internals-compile-time-instrumentation/

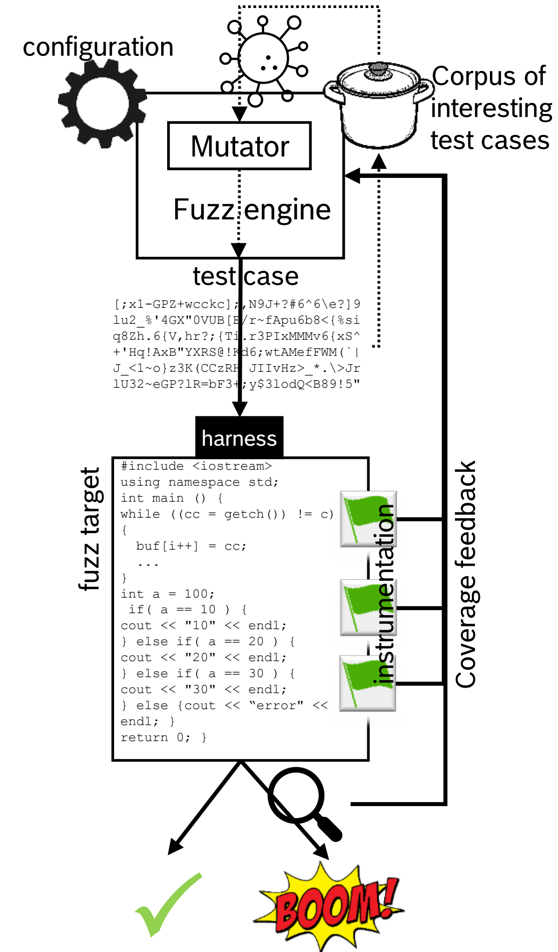
The injected code at a branch point is:

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

Every byte in the output map represents a tuple hit (branch_src, branch_dst). That way e.g. the following execution traces can be distinguished

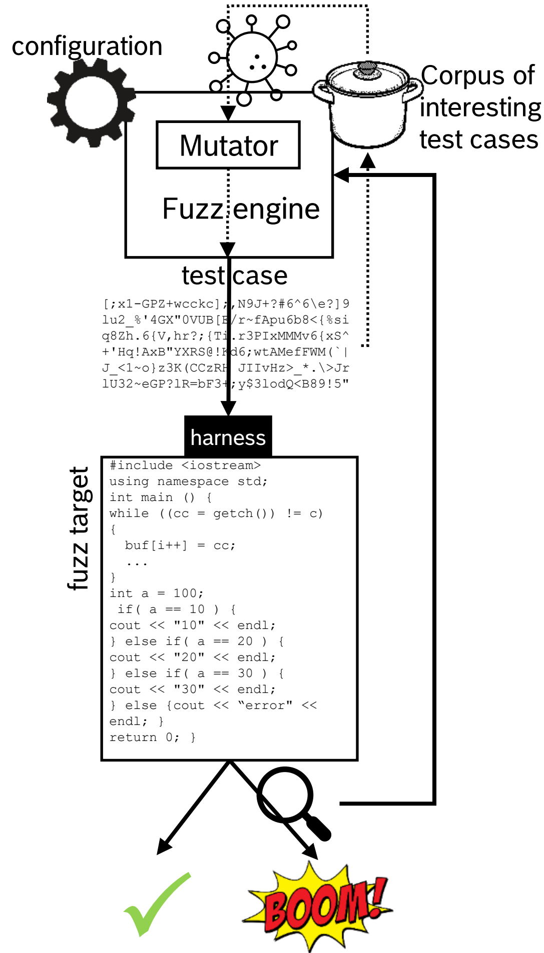
```
A -> B -> C -> D -> E (tuples: AB, BC, CD, DE)
A -> B -> D -> C -> E (tuples: AB, BD, DC, CE)
```

https://lcamtuf.coredump.cx/afl/technical_details.txt



Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Terminology – other instrumentations



In recent years, there were some instrumentation optimizations (for faster runtime, better coverage, CPU independence, more features such as cmplog and autodictionary, …).

If in doubt, choose as AFL++ proposes

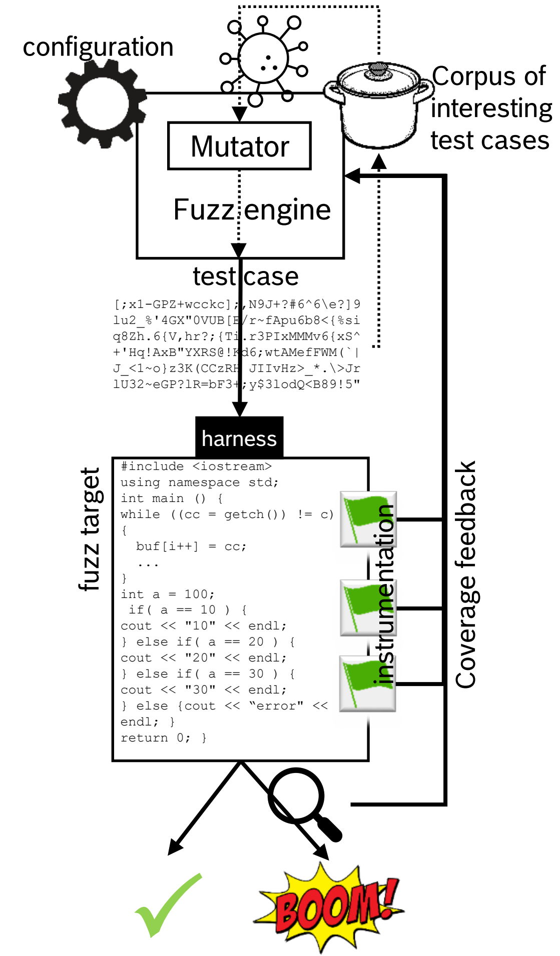
If **clang/clang++ 11+** is available,
use `afl-clang-lto/afl-clang-lto++`

else if **clang/clang++ 3.8+** is available,
use `afl-clang-fast/afl-clang-fast++`

else if **gcc 5+** is available,
use `afl-gcc-fast/afl-gcc-fast++`

else use `afl-gcc/afl-g++` or `afl-clang/afl-clang++`

https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.lto.md
https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.llvm.md
https://github.com/AFLplusplus/AFLplusplus/blob/stable/instrumentation/README.gcc_plugin.md



© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let's fuzz – refactor and another crash

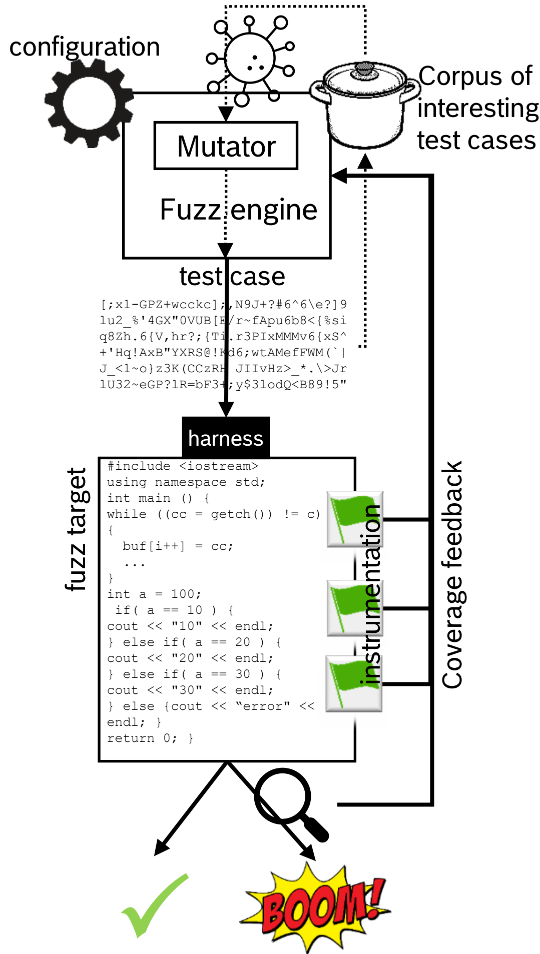
```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define BUFFERMAXSIZE 10

int readIntoBufferFromStdin(char* buffer){
    int numberReadChars = 0;
    while(numberReadChars < BUFFERMAXSIZE){
        buffer[numberReadChars] = getc(stdin);
        numberReadChars++;
    }
return numberReadChars;
}

int main( ) {
    char *buffer;
    buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
    int bufferCurrentIndex = 0;
    bufferCurrentIndex = readIntoBufferFromStdin(buffer);
    tryToTriggerBugWithInput(buffer, bufferCurrentIndex);
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

```c
void tryToTriggerBugWithInput(char* buffer, size_t bufferSize){
    if (bufferSize >= 3){
        if (buffer[0] == 'B'){
            if (buffer[1] == 'U'){
                if (buffer[2] == 'G'){
                    if (buffer[3] == 'G'){
                        printf ("Address crash condition found!\n");
                        *((int *)0) = 0; // do some crashing
                    }else if (buffer[3] == '!'){
                        printf ("Another crash condition found!\n");
                        *((int *)0) = 0; // do some crashing
                    }
                }
            }
        }
    }
}
```

```
$ ./toy-example-buffer-overflow < lotsofA
SUCCESSFUL TERMINATION!
$ echo "BUGG" | ./more-paths-stackoverflow
Address crash condition found!
Segmentation fault
$ echo "BUG!" | ./more-paths-stackoverflow
Another crash condition found!
Segmentation fault
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz – instrumentation and profiling




```
$ gcc -g -Wall -o more-paths-stackoverflow more-paths-stackoverflow.cpp

$ afl-gcc -g -Wall -o more-paths-stackoverflow-instrumented more-paths-stackoverflow.cpp
…
[+] Instrumented 13 locations (64-bit, non-hardened mode, ratio 100%).

$ gcc -g -Wall -fprofile-arcs -ftest-coverage -o more-paths-stackoverflow-profiled more-paths-stackoverflow.cpp

$ ls -l
…
-rwxrwxr-x 1 huth huth 11648 Sep 27 13:15 more-paths-stackoverflow
-rw-rw-r-- 1 huth huth  2320 Sep 27 13:15 more-paths-stackoverflow.gcno
-rwxrwxr-x 1 huth huth 18960 Sep 27 13:15 more-paths-stackoverflow-instrumented
-rwxrwxr-x 1 huth huth 27328 Sep 27 13:15 more-paths-stackoverflow-profiled
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let's fuzz – coverage with corpus

LCOV - code coverage report

Current view: top level - more-paths - more-paths-stackoverflow.cpp (source / functions)
Test: id:000003,src:000002,time:935324,op:havoc,rep:2,+cov.lcov_info_final
Date: 2020-09-27 13:24:57

	Hit	Total	Coverage
Lines:	21	25	84.0 %
Functions:	3	3	100.0 %
Branches:	11	14	78.6 %

```
          Branch data     Line data    Source code
 1                   :             : #include <stdio.h>
 2                   :             : #include <stdint.h>
 3                   :             : #include <stdlib.h>
 4                   :             :
 5                   :             : #define BUFFERMAXSIZE 10
 6                   :             :
 7                   :           4 : int readIntoBufferFromStdin(char* buffer){
 8                   :           4 :         int numberReadChars = 0;
 9         [ +  + ]:          44 :         while(numberReadChars < BUFFERMAXSIZE){
10                   :          40 :                 buffer[numberReadChars] = getc(stdin);
11                   :          40 :                 numberReadChars++;
12                   :             :         }
13                   :           4 : return numberReadChars;
14                   :             : }
15                   :             :
16                   :           4 : void tryToTriggerBugWithInput(char* buffer, size_t bufferSize){
17         [ +  - ]:           4 :         if (bufferSize >= 3){
18         [ +  + ]:           4 :                 if (buffer[0] == 'B'){
19         [ +  + ]:           3 :                         if (buffer[1] == 'U'){
20         [ +  + ]:           2 :                                 if (buffer[2] == 'G'){
21         [ -  + ]:           1 :                                         if (buffer[3] == 'G'){
22                   :           0 :                                                 printf ("Address crash condition found!\n");
23                   :           0 :                                                 *((int *)0) = 0; // do some crashing
24         [ -  + ]:           1 :                                         }else if (buffer[3] == '!'){
25                   :           0 :                                                 printf ("Another crash condition found!\n");
26                   :           0 :                                                 *((int *)0) = 0; // do some crashing
27                   :             :                                         }
28                   :             :                                 }
29                   :             :                         }
30                   :             :                 }
31                   :             :         }
32                   :           4 : }
33                   :             :
34                   :           4 : int main( ) {
35                   :             :         char *buffer;
36                   :           4 :         buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
37                   :           4 :         int bufferCurrentIndex = 0;
38                   :           4 :         bufferCurrentIndex = readIntoBufferFromStdin(buffer);
39                   :           4 :         tryToTriggerBugWithInput(buffer, bufferCurrentIndex);
40                   :           4 :         printf("SUCCESSFUL TERMINATION!");
41                   :           4 :         return 0;
42                   :             : }
43                   :             :
44                   :             :
```

```
$ ls out/queue/
id:000000,time:0,orig:empty
id:000001,src:000000,time:113,op:havoc,rep:16,+cov
id:000002,src:000001,time:43251,op:havoc,rep:2,+cov
id:000003,src:000002,time:935324,op:havoc,rep:2,+cov
```

"empty"
B
BU
BUGU

```
$ afl-cov -d out/ --coverage-cmd "./more-paths-stackoverflow-profiled < AFL_FILE" --code-dir .
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – afl folder structure

```
$ tree -L 2 out/
out/
├── cmdline
├── cov
│   ├── afl-cov.log
│   ├── afl-cov-status
│   ├── diff
│   ├── id-delta-cov
│   ├── lcov
│   ├── pos-cov
│   ├── web
│   └── zero-cov
├── crashes
│   ├── id:000000,sig:11,src:000003,time:123795,op:ext_UI,pos:2
│   └── README.txt
├── fuzz_bitmap
├── fuzzer_stats
├── hangs
├── plot_data
└── queue
    ├── id:000000,time:0,orig:empty
    ├── id:000001,src:000000,time:1587,op:havoc,rep:32,+cov
    ├── id:000002,src:000001,time:114632,op:havoc,rep:4,+cov
    └── id:000003,src:000002,time:114875,op:ext_UI,pos:2,+cov
```

- find coverage e.g. in web format
- crashing inputs are reproducible
- binary instrumentation output in shared memory
- keep queue for regression

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – Seeds

```
american fuzzy lop ++2.60d (more-paths-stackoverflow-instrum...) [explore] {0}
── process timing ──────────────────────────────────── overall results ──
       run time : 0 days, 0 hrs, 0 min, 1 sec       cycles done : 0
  last new path : 0 days, 0 hrs, 0 min, 1 sec       total paths : 4
last uniq crash : 0 days, 0 hrs, 0 min, 1 sec     uniq crashes : 2
 last uniq hang : none seen yet                      uniq hangs : 0
── cycle progress ─────────────────── map coverage ──
  now processing : 2.0 (50.0%)          map density : 0.01% / 0.02%
 paths timed out : 0 (0.00%)         count coverage : 1.00 bits/tuple
── stage progress ─────────────────── findings in depth ──
  now trying : arith 8/8              favored paths : 4 (100.00%)
 stage execs : 42/268 (15.67%)         new edges on : 4 (100.00%)
 total execs : 3226                   total crashes : 2 (2 unique)
  exec speed : 2118/sec                total tmouts : 0 (0 unique)
── fuzzing strategy yields ──────────────────── path geometry ──
   bit flips : 3/96, 0/93, 0/87                  levels : 2
  byte flips : 0/12, 0/9, 0/3                   pending : 2
 arithmetics : 1/446, 0/50, 0/0                pend fav : 2
  known ints : 0/51, 0/168, 0/88              own finds : 3
  dictionary : 0/0, 0/0, 0/0                   imported : n/a
   havoc/rad : 1/2048, 0/0, 0/0               stability : 100.00%
   py/custom : 0/0, 0/0
        trim : n/a, 0.00%                               [cpu000:108%]
                                          ^C
```

```
$ echo "BUG" > in/BUG
$ afl-fuzz -i in/ -o out/ ./more-paths-stackoverflow-instrumented

$ cat crashes/id\:000000…
BUG!
$ cat crashes/id\:000001…
BUGG9GGGGGG
```

afl-tmin test case minimizer

afl-cmin corpus minimization tool

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz – dictionary

```
american fuzzy lop ++2.60d (more-paths-stackoverflow-instrum...) [explore] {0}
┌─ process timing ───────────────────────────────────────┬─ overall results ──┐
│        run time : 0 days, 0 hrs, 2 min, 5 sec          │  cycles done : 243 │
│   last new path : 0 days, 0 hrs, 0 min, 10 sec         │  total paths : 4   │
│ last uniq crash : 0 days, 0 hrs, 0 min, 1 sec          │ uniq crashes : 1   │
│  last uniq hang : none seen yet                        │   uniq hangs : 0   │
├─ cycle progress ────────────────────┬─ map coverage ───┴────────────────────┤
│  now processing : 3.0 (75.0%)       │    map density : 0.01% / 0.02%       │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.00 bits/tuple     │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc                 │ favored paths : 4 (100.00%)          │
│ stage execs : 2250/16.4k (13.73%)   │  new edges on : 4 (100.00%)          │
│ total execs : 127k                  │ total crashes : 5 (1 unique)         │
│  exec speed : 1444/sec              │  total tmouts : 10 (3 unique)        │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : 0/56, 0/52, 0/44                      │    levels : 4         │
│  byte flips : 0/7, 0/3, 0/0                         │   pending : 1         │
│ arithmetics : 0/390, 0/0, 0/0                       │  pend fav : 1         │
│  known ints : 0/43, 0/84, 0/0                       │ own finds : 3         │
│  dictionary : 0/6, 2/33, 0/0                        │  imported : n/a       │
│   havoc/rad : 2/124k, 0/0, 0/0                      │ stability : 100.00%   │
│   py/custom : 0/0, 0/0                              ├───────────────────────┘
│        trim : n/a, 0.00%                            │          [cpu000:186%]
└─────────────────────────────────────────────────────┘^C
```

```
$ rm -rf in/ && mkdir in && echo "" > in/empty
```

```
$ cat bug.dict
"B"
"U"
"G"
```

```
$ afl-fuzz -i in/ -o out/ -x bug.dict ./more-paths-stackoverflow-instrumented
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let's fuzz – fuzzing strategies

```
american fuzzy lop ++2.60d (more-paths-stackoverflow-instrum...) [explore] {0}
┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 2 min, 5 sec        │  cycles done : 243   │
│   last new path : 0 days, 0 hrs, 0 min, 10 sec       │  total paths : 4     │
│ last uniq crash : 0 days, 0 hrs, 0 min, 1 sec        │ uniq crashes : 1     │
│  last uniq hang : none seen yet                      │   uniq hangs : 0     │
├─ cycle progress ──────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 3.0 (75.0%)     │    map density : 0.01% / 0.02%          │
│ paths timed out : 0 (0.00%)       │ count coverage : 1.00 bits/tuple        │
├─ stage progress ──────────────────┼─ findings in depth ─────────────────────┤
│  now trying : havoc               │ favored paths : 4 (100.00%)             │
│ stage execs : 2250/16.4k (13.73%) │  new edges on : 4 (100.00%)             │
│ total execs : 127k                │ total crashes : 5 (1 unique)            │
│  exec speed : 1444/sec            │  total tmouts : 10 (3 unique)           │
├─ fuzzing strategy yields ─────────┴─────────────┬─ path geometry ───────────┤
│   bit flips : 0/56, 0/52, 0/44                  │    levels : 4             │
│  byte flips : 0/7, 0/3, 0/0                     │   pending : 1             │
│ arithmetics : 0/390, 0/0, 0/0                   │  pend fav : 1             │
│  known ints : 0/43, 0/84, 0/0                   │ own finds : 3             │
│  dictionary : 0/6, 2/33, 0/0                    │  imported : n/a           │
│   havoc/rad : 2/124k, 0/0, 0/0                  │ stability : 100.00%       │
│   py/custom : 0/0, 0/0                          ├───────────────────────────┘
│        trim : n/a, 0.00%                        │            [cpu000:186%]
└─────────────────────────────────────────────────┘^C
```

- Flips: deterministic bit and byte flips
- Arithmetics: add or subtract small integers to 8-, 16-, 32-bit values.
- Known ints: replace values with pre-known magic values
- Havoc: multiple mutations together
- Custom: implement your own custom mutator
- Trim: check if shortened input results in same execution path

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – libfuzzer

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define BUFFERMAXSIZE 10

int readIntoBufferFromStdin(char* buffer){
    int numberReadChars = 0;
    while(numberReadChars < BUFFERMAXSIZE){
        buffer[numberReadChars] = getc(stdin);
        numberReadChars++;
    }
return numberReadChars;
}

int main( ) {
    char *buffer;
    buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
    int bufferCurrentIndex = 0;
    bufferCurrentIndex = readIntoBufferFromStdin(buffer);
    tryToTriggerBugWithInput(buffer, bufferCurrentIndex);
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

“System level entry point”

```
void tryToTriggerBugWithInput(char* buffer, size_t bufferSize){
    if (bufferSize >= 3){
        if (buffer[0] == 'B'){
            if (buffer[1] == 'U'){
                if (buffer[2] == 'G'){
                    if (buffer[3] == 'G'){
                        printf ("Address crash condition found!\n");
                        *((int *)0) = 0; // do some crashing
                    }else if (buffer[3] == '!'){
                        printf ("Another crash condition found!\n");
                        *((int *)0) = 0; // do some crashing
                    }
                }
            }
        }
    }
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    char *buffer;
    buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
    tryToTriggerBugWithInput((char*)Data, Size);
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

“Unit level entry point”

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz – libfuzzer

```
$ clang -g -Wall -fsanitize=fuzzer -o more-paths-stackoverflow-libfuzzer more-paths-stackoverflow.cpp

$ ./more-paths-stackoverflow-libfuzzer
INFO: Seed: 4058694895
INFO: Loaded 1 modules   (12 inline 8-bit counters): 12 [0x69bfc0, 0x69bfcc),
INFO: Loaded 1 PC tables (12 PCs): 12 [0x489290,0x489350),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 3 ft: 4 corp: 1/1b exec/s: 0 rss: 23Mb
…
```

```
Another crash condition found!
UndefinedBehaviorSanitizer:DEADLYSIGNAL
==21776==ERROR: UndefinedBehaviorSanitizer: SEGV on unknown address 0x000000000000 (pc 0x000000479b27 bp 0x7fff0e4ac060 sp 0x7fff0e4ac010 T21776)
==21776==The signal is caused by a WRITE memory access.
==21776==Hint: address points to the zero page.
    #0 0x479b27 in tryToTriggerBugWithInput(char*, unsigned long) /home/huth/more-paths/more-paths-stackoverflow.cpp:26:19
    #1 0x479bc2 in LLVMFuzzerTestOneInput /home/huth/more-paths/more-paths-stackoverflow.cpp:37:2
…
BUG!\x00
…
```


© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – Address Sanitizer

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define BUFFERMAXSIZE 10

int readIntoBufferFromStdin(char* buffer){
    int numberReadChars = 0;
    while(numberReadChars < BUFFERMAXSIZE){
        buffer[numberReadChars] = getc(stdin);
        numberReadChars++;
    }
return numberReadChars;
}

int main( ) {
    char *buffer;
    buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
    int bufferCurrentIndex = 0;
    bufferCurrentIndex = readIntoBufferFromStdin(buffer);
    tryToTriggerBugWithInput(buffer, bufferCurrentIndex);
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

“System level entry point”

```
void tryToTriggerBugWithInput(char* buffer, size_t bufferSize){
    if (bufferSize >= 3){
        if (buffer[0] == 'B'){
            if (buffer[1] == 'U'){
                if (buffer[2] == 'G'){
                    if (buffer[3] == 'G'){
                        printf("Memory leak found!\n");
                        printf("%c", buffer[BUFFERMAXSIZE+10]);
                    }else if (buffer[3] == '!'){
                        printf("Memory leak found!\n");
                        printf("%c", buffer[BUFFERMAXSIZE+10]);
                    }
                }
            }
        }
    }
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    char *buffer;
    buffer = (char*)malloc(BUFFERMAXSIZE*sizeof(char));
    tryToTriggerBugWithInput((char*)Data, Size);
    printf("SUCCESSFUL TERMINATION!");
    return 0;
}
```

“Unit level entry point”

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Optimization: Sanitizers (heartbleed example)

Try running the fuzzer:

```
./openssl-1.0.1f-fsanitize_fuzzer
```

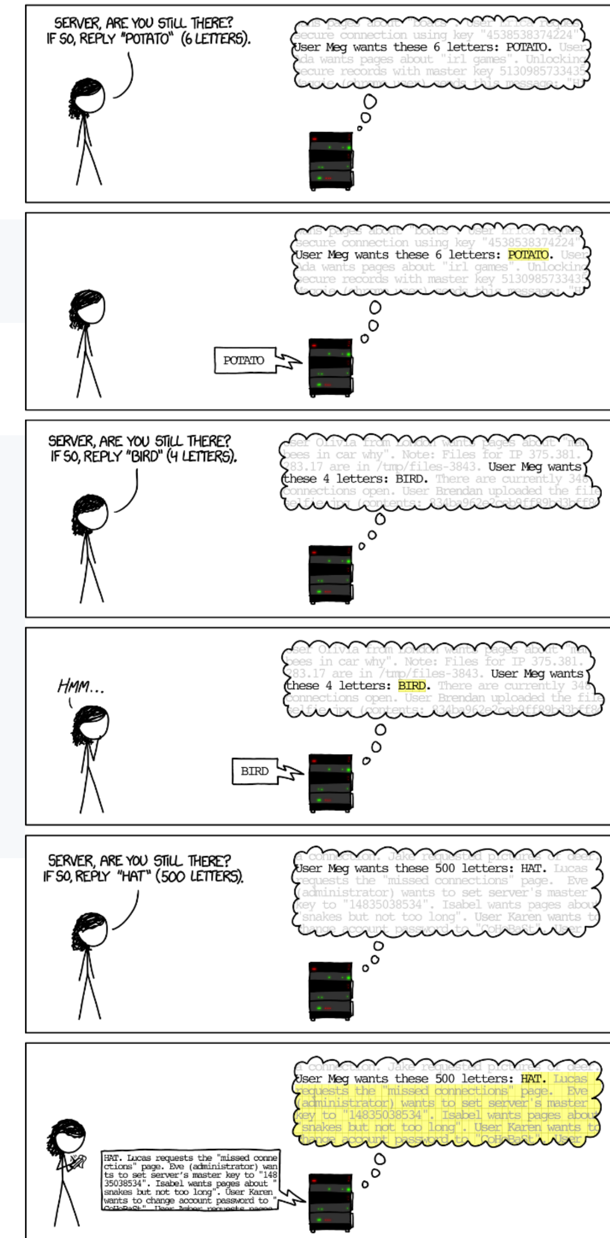
You whould see something like this in a few seconds:

```
==5781==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x629000009748 at pc 0x0000004a9817...
READ of size 19715 at 0x629000009748 thread T0
    #0 0x4a9816 in __asan_memcpy (heartbleed/openssl-1.0.1f+0x4a9816)
    #1 0x4fd54a in tls1_process_heartbeat heartbleed/BUILD/ssl/t1_lib.c:2586:3
    #2 0x58027d in ssl3_read_bytes heartbleed/BUILD/ssl/s3_pkt.c:1092:4
    #3 0x585357 in ssl3_get_message heartbleed/BUILD/ssl/s3_both.c:457:7
    #4 0x54781a in ssl3_get_client_hello heartbleed/BUILD/ssl/s3_srvr.c:941:4
    #5 0x543764 in ssl3_accept heartbleed/BUILD/ssl/s3_srvr.c:357:9
    #6 0x4eed3a in LLVMFuzzerTestOneInput FTS/openssl-1.0.1f/target.cc:38:3
```

Sanitizers 'provoke' a crash on certain behaviour, to make certain bug types detectable for fuzzers, e.g. for a reading buffer overflow.

https://github.com/google/fuzzer-test-suite/blob/master/tutorial/libFuzzerTutorial.md

https://xkcd.com/1354/



© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Let’s fuzz – Address Sanitizer

```
$ clang -g -Wall  -fsanitize=address -fsanitize=fuzzer -o more-paths-stackoverflow-libfuzzer more-paths-stackoverflow.cpp

$ ./more-paths-stackoverflow-libfuzzer
INFO: Seed: 1645798338
INFO: Loaded 1 modules   (12 inline 8-bit counters): 12 [0x77a0a0, 0x77a0ac),
INFO: Loaded 1 PC tables (12 PCs): 12 [0x554c30,0x554cf0),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 3 ft: 4 corp: 1/1b exec/s: 0 rss: 27Mb
…
```

```
==25042==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000085873 at pc 0x00000053cba6 bp 0x7fffc4f8e310 sp 0x7fffc4f8e308
READ of size 1 at 0x602000085873 thread T0
    #0 0x53cba5 in tryToTriggerBugWithInput(char*, unsigned long) /more-paths0/more-paths-stackoverflow.cpp:21:10
    #1 0x53ce04 in LLVMFuzzerTestOneInput /more-paths0/more-paths-stackoverflow.cpp:37:2
…
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Agenda

1. Motivation
2. Theory
   1. What is fuzzing?
   2. How to talk about fuzzing?
   3. What can be fuzzed?
   4. What fuzzing types are there?
3. Practice
   1. Toy example – set up a fuzz test
   2. **Real world example – optimize a fuzz test**
4. Challenges and good practices

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Let’s fuzz – a real world example

- As an example target software we use a snapshot from WOFF2 (font compression) and a harness from fuzzer-test-suite https://github.com/google/fuzzer-test-suite/tree/master/woff2-2016-05-06
  - WOFF (Web Open Font Format) is a font format for web pages. WOFF2 adds e.g. the Brotli compression.
  - WOFF2 is supported in all bigger browsers (Chrome, Edge, Opera, Firefox, Safari)
  - We compile with afl-clang-fast for AFL++ and clang for libFuzzer to reuse the same harness

- https://github.com/AFLplusplus/AFLplusplus
- https://llvm.org/docs/LibFuzzer.html

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz

- Harness from https://github.com/google/fuzzer-test-suite/blob/master/woff2-2016-05-06/target.cc

```
// Copyright 2016 Google Inc. All Rights Reserved.
// Licensed under the Apache License, Version 2.0 (the "License");
#include <stddef.h>
#include <stdint.h>

#include "woff2_dec.h"

// Entry point for LibFuzzer.
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
  std::string buf;
  woff2::WOFF2StringOut out(&buf);
  out.SetMaxSize(30 * 1024 * 1024);
  woff2::ConvertWOFF2ToTTF(data, size, &out);
  return 0;
}
```

- In short, LLVMFuzzerTestOneInput ‘replaces’ the main function of the software under test
  - Indicated by `-fsanitize=fuzzer` during compilation and linking
  - test case provided via `data` and `size`
  - test case injected into software by function `woff2::ConvertWOFF2ToTFF`

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Let’s fuzz

- Clone and build from https://github.com/AFLplusplus/AFLplusplus

- Clone https://github.com/google/fuzzer-test-suite

```
$ export FUZZING_ENGINE=afl
$ export CC=afl-clang-fast
$ export CXX=afl-clang-fast
$ ./build.sh
```

- Download clang (or build from sources) https://github.com/google/fuzzing/blob/master/tutorial/libFuzzerTutorial.md
- clone https://github.com/google/fuzzer-test-suite

```
$ export FUZZING_ENGINE=libfuzzer
$ export CC=clang
$ export CXX=clang++
$ ./build.sh
```

The build script then clones the WOFF2 snapshot and compiles (and instruments) the source code.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Let’s fuzz

```
$ afl-fuzz -i seeds/ -o CORPUS-
  woff2-2016-05-06-afl/ ./woff2-
  2016-05-06-afl
```

```
./woff2-2016-05-06-fsanitize_fuzzer
```

```
      american fuzzy lop ++2.60d (woff2-2016-05-06-afl) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 53 min, 44 sec      │  cycles done : 6626 │
│   last new path : 0 days, 0 hrs, 6 min, 33 sec       │  total paths : 11   │
│ last uniq crash : none seen yet                      │ uniq crashes : 0    │
│  last uniq hang : none seen yet                      │   uniq hangs : 0    │
├─ cycle progress ───────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 2.6625 (18.2%)   │    map density : 0.03% / 0.04%      │
│ paths timed out : 0 (0.00%)        │ count coverage : 1.00 bits/tuple    │
├─ stage progress ───────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                │ favored paths : 2 (18.18%)          │
│ stage execs : 383/384 (99.74%)     │  new edges on : 4 (36.36%)          │
│ total execs : 37.5M                │ total crashes : 0 (0 unique)        │
│  exec speed : 9755/sec             │  total tmouts : 0 (0 unique)        │
├─ fuzzing strategy yields ──────────┴───────────┬─ path geometry ────────┤
│   bit flips : 0/32, 0/30, 0/26                 │    levels : 5          │
│  byte flips : 0/4, 0/2, 0/0                    │   pending : 0          │
│ arithmetics : 0/224, 0/16, 0/0                 │  pend fav : 0          │
│  known ints : 0/23, 0/56, 0/0                  │ own finds : 2          │
│  dictionary : 0/0, 0/0, 0/0                    │  imported : n/a        │
│   havoc/rad : 2/26.1M, 0/11.4M, 0/0            │ stability : 76.00%     │
│   py/custom : 0/0, 0/0                         ├────────────────────────┘
│        trim : 0.00%/218, 0.00%                 │          [cpu000: 95%]
└────────────────────────────────────────────────┘
```

```
INFO: Seed: 1534267175
INFO: Loaded 1 modules   (9611 inline 8-bit counters): 9611 [0x93a710, 0x93cc9b),
INFO: Loaded 1 PC tables (9611 PCs): 9611 [0x6e67e8,0x70c098),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2      INITED cov: 15 ft: 16 corp: 1/1b exec/s: 0 rss: 37Mb
#5      NEW    cov: 16 ft: 17 corp: 2/10b exec/s: 0 rss: 38Mb L: 9/9 MS: 3 ShuffleBytes-CopyPart-CMP- DE: "\x01\x00\x00\x00\x00\x00\x00\x00"-
#8      REDUCE cov: 16 ft: 17 corp: 2/6b exec/s: 0 rss: 38Mb L: 5/5 MS: 3 ChangeBinInt-PersAutoDict-EraseBytes- DE: "\x01\x00\x00\x00\x00\x00\x00\x00"-
#45     REDUCE cov: 16 ft: 17 corp: 2/5b exec/s: 0 rss: 38Mb L: 4/4 MS: 2 ChangeByte-EraseBytes-
#3002   REDUCE cov: 17 ft: 18 corp: 3/9b exec/s: 0 rss: 41Mb L: 4/4 MS: 2 ShuffleBytes-CMP- DE: "wOF2"-
#3059   NEW    cov: 18 ft: 19 corp: 4/17b exec/s: 0 rss: 41Mb L: 8/8 MS: 2 CopyPart-CrossOver-
#3071   NEW    cov: 19 ft: 20 corp: 5/34b exec/s: 0 rss: 41Mb L: 17/17 MS: 2 ChangeByte-InsertRepeatedBytes-
#3377   REDUCE cov: 19 ft: 20 corp: 5/33b exec/s: 0 rss: 41Mb L: 16/16 MS: 1 EraseBytes-
#3744   REDUCE cov: 19 ft: 20 corp: 5/31b exec/s: 0 rss: 42Mb L: 14/14 MS: 2 ChangeByte-EraseBytes-
#4060   REDUCE cov: 19 ft: 20 corp: 5/29b exec/s: 0 rss: 42Mb L: 12/12 MS: 1 EraseBytes-
#5282   REDUCE cov: 20 ft: 21 corp: 6/41b exec/s: 0 rss: 43Mb L: 12/12 MS: 2 ChangeBinInt-ChangeBinInt-
```

Coverage information

Both fuzzers then try to maximize coverage by mutating interesting test cases.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Let’s fuzz

AFL++ fuzzes for an unlimited amount of time.

libFuzzer fuzzes until a crash is found.

Both fuzzers save a reproducible crashing file.

For our WOFF2 example, both can find a multi-byte-write-heap-buffer-overflow.

A crash looks like:

```
ERROR: AddressSanitizer: heap-buffer-overflow
WRITE of size 6707 at 0x62300000534d thread T0
    #0 0x4a95d3 in __asan_memcpy
    #1 0x62fa5c in woff2::Buffer::Read(unsigned char*, unsigned long) src/./buffer.h:86:7
    #2 0x62fa5c in woff2::(anonymous namespace)::ReconstructGlyf src/woff2_dec.cc:500
    #3 0x62fa5c in woff2::(anonymous namespace)::ReconstructFont src/woff2_dec.cc:917
    #4 0x62fa5c in woff2::ConvertWOFF2ToTTF src/woff2_dec.cc:1282
```

https://github.com/google/fuzzer-test-suite/tree/master/woff2-2016-05-06

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Optimization: Seeds

Seeds are initial (small and valid) test cases, so that the fuzzer does not have to start from thin air.

In our example the build.sh downloads the Roboto font as seed.

```
$ afl-fuzz -i seeds/ -o CORPUS-
  woff2-2016-05-06-afl/ ./woff2-
  2016-05-06-afl
```

```
$ ./$woff2-2016-05-06-fsanitize_fuzzer
  CORPUS seeds
```

Roboto-
Regular.woff2

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Optimization: Dictionaries

Dictionaries help the fuzzer by replacing part of the test case by a dictionary entry, rather than e.g. random. Dictionary entries should be often used symbols and words by the target software.

There are multiple pre-built dictionaries available, e.g. for SQL, XML, JSON, ...
https://github.com/AFLplusplus/AFLplusplus/tree/stable/dictionaries

```
$ -x dict=DICTIONARY_FILE
```

```
$ -dict=DICTIONARY_FILE
```

json.dict

```
"0" ",0" ":0" "0:" "-1.2e+3"
"true" "false" "null" "\"\""
",\"\"" ":\"\"" "\"\":" "{}" ",{}"
":{}" "{\"\":0}" "{{}}" "[]" ",[]"
":[]" "[0]" "[[]]"
```


© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Optimization: Parallelization

Run first fuzzer as 'manager' **-M**

```
$ ./afl-fuzz -i seeds -o sync_dir
  -M fuzzer01 [...]
```

then, start up secondary instances

```
$ ./afl-fuzz -i seeds -o sync_dir
  -S fuzzer02 [...]
```

```
$ ./afl-fuzz -i seeds -o sync_dir
  -S fuzzer03 [...]
```

Each fuzzer will keep its state in a separate subdirectory in `sync_dir`, and the master syncs all fuzzing instances.

Run multiple libfuzzer processes in parallel with a shared corpus directory.

`$JOBS` is by default half of available CPU cores

```
$ ./$woff2-2016-05-06-fsanitize_fuzzer
  CORPUS -workers=$JOBS CORPUS
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Optimization: Grammar

- grammar can be implemented in custom mutators (libprotobuf-mutator) and/or harness

```
message Msg {
  optional float optional_float = 1;
  optional uint64 optional_uint64 = 2;
  optional string optional_string = 3;
}
```

```
DEFINE_PROTO_FUZZER(const libfuzzer_example::Msg& message) {
  // Emulate a bug.
  if (message.optional_string() == "FooBar" &&
      message.optional_uint64() > 100 &&
      !std::isnan(message.optional_float()) &&
      std::fabs(message.optional_float()) > 1000 &&
      std::fabs(message.optional_float()) < 1E10) {
    abort();
  }
}
```

  - https://github.com/google/fuzzing/blob/master/docs/structure-aware-fuzzing.md
  - there are some examples (PNG, protocol messages, SQLite, some stateful APIs)

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Agenda

1. Motivation
2. Theory
    1. What is fuzzing?
    2. How to talk about fuzzing?
    3. What can be fuzzed?
    4. What fuzzing types are there?
3. Practice
    1. Toy example – set up a fuzz test
    2. Real world example – optimize a fuzz test
4. **Challenges and good practices**

Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

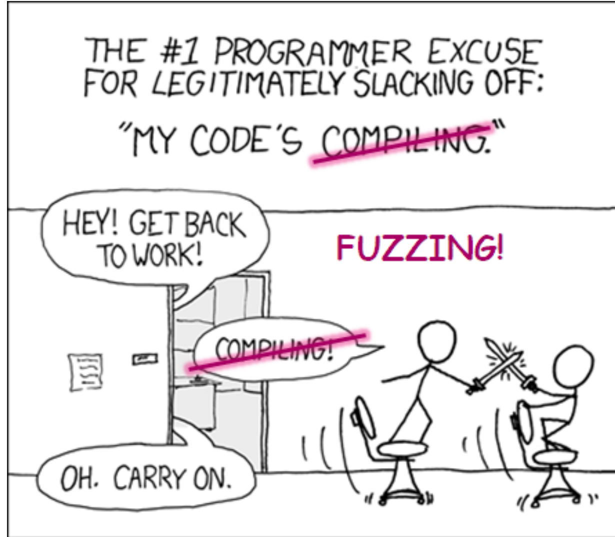
Practical Challenges

- Find a suited fuzz target.
  - E.g. an API function, which consumes untrusted input under the control of a potential attacker.
- Write a fuzz test.
  - Connecting the software under test to the fuzzing engine is manual work.
  - E.g. harness can fuzz on unit, component, or system level.
- Fuzzing results should be observable.
  - E.g. crashes in black-box fuzzing could be hard to detect.
  - Instrumentation can be hard (different compilers, debug vs. productive software, multiple processes)
- Speed up your fuzzing, as it relies on numerous of test case executions.
  - Keeping the current test case corpora at a relevant minimum.
  - Parallelize your fuzz tests while working on the same test corpora (synchronize and do regular clean-ups).
  - Keep I/O communication at a minimum.
- Provide a useful structure of the input.
  - Grammar, dictionary, and initial seeds
- No fixed value for timeout.
  - Typical tests vary from hours over days to weeks.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Challenge: How long should I fuzz?

- Short answer: it depends



https://www.explainxkcd.com/wiki/index.php/303:_Compiling
https://nth10sd.github.io/js-fuzzing-in-mozilla/

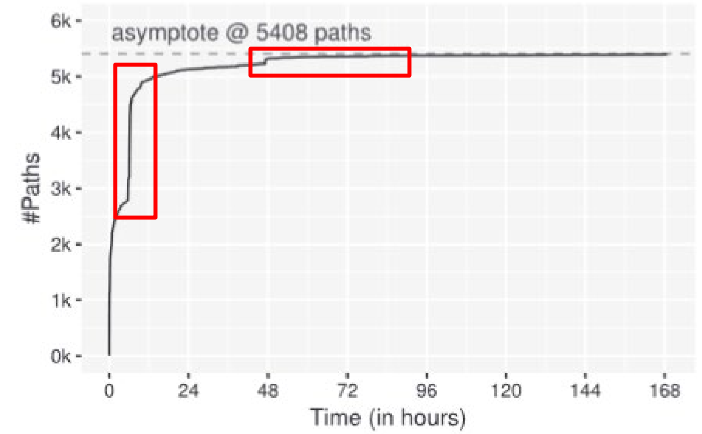
- Long answer:
  - Coverage over time *usually* follows an asymptotic behaviour [2].
    - You can always fuzz more [3].
  - Rule of thumb: to find the next bug you need *exponentially* more resources, not linearly [1].
  - You can only estimate the residual risk when to stop fuzzing (as least as hard as verification problem), but there is no good estimator right now. [4]

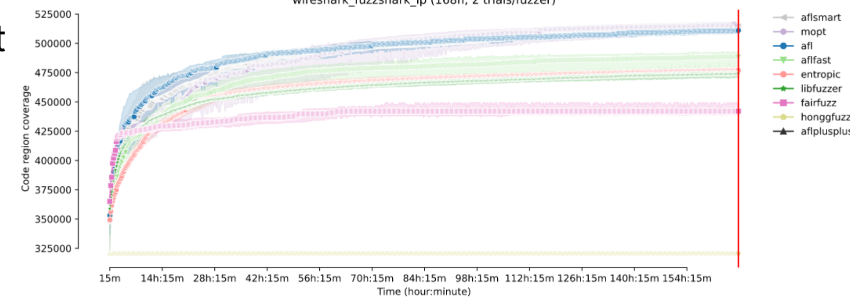
[1] Marcel Böhme and Brandon Falk *Fuzzing: On the Exponential Cost of Vulnerability Discovery*

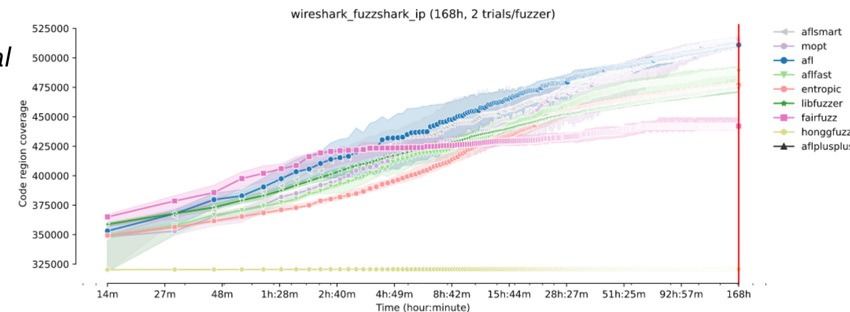
[2] Marcel Böhme *STADS: Software Testing as Species Discovery*

[3] Liyanage et al., *False Peaks: On the Estimation of Fuzzing Effectiveness*

[4] Böhme et al., *Estimating Residual Risk in Greybox Fuzzing*







https://www.fuzzbench.com/reports/2021-08-19-crash-s-7d/index.html

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Good Practices

- Fuzz everything,
  - but don’t generate false positives (e.g. use FuzzedDataProvider or FuzzTest).
- Fuzz at least for a realistic threat, then dig deeper.
  - The fuzz target should consume input which is under malicious control.
  - E.g. HW system level testing: Fuzz components at least over the bus, optionally fuzz component’s internals
  - E.g. SW component level testing: Fuzz components over interface, optionally fuzz internal methods
- Validate your fuzz tests before fuzzing.
  - The fuzz test should consume the generated input, the fuzz test should not crash for valid inputs, and code coverage tracking of the fuzzer should work.
  - Design your tests for testability and observable results.
- Fuzz in parallel and synchronize corpora.
- Help your fuzzer.
  - Use a grammar, dictionary, and/or seeds. There exist prebuilt structures for fuzzing engines.

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

Good Practices

- Use fuzzing features.
  - afl++ has some amazing features, e.g. auto dictionary https://github.com/AFLplusplus/AFLplusplus#important-features-of-afl
- Have multiple fuzz harnesses.
  - Cover your system with multiple wrappers rather than with a single one.
- Combine harnesses and features.
  - E.g. fuzz the same harness with a vanilla fuzzer and with a grammar-based fuzzer; and sync the corpus
- Just start fuzzing.
  - Actual fuzzer does not matter that much.
  - More complex code -> more likely are bugs.
- Automate your fuzzing.

Code | Issues 45 | Pull requests 1 | Actions | Projects | Wiki | Security | ...

Create fuzztest.yml
FUZZTEST #10
Re-run jobs

Summary
Jobs
build

build
failed 14 hours ago in 5m 42s
Search logs

Set up job 1s
Run actions/checkout@v3 4s
Install clang 52s
Build and test 2m 36s
Fuzz 2m 7s

```
127 INFO: Build completed successfully, 162 total actions
128 INFO: Running command line: external/bazel_tools/tools/test/test-
    setup.sh ./fuzz_me_test '--fuzz=FuzzMeFuzz.HarnessString'
129 exec ${PAGER:-/usr/bin/less} "$0" || exit 1
130 Executing tests from //:fuzz_me_test
131 -----------------------------------------------------------------------------
    ------
132 [.] Sanitizer coverage enabled. Counter map size: 22087, Cmp map size:
    262144
133 Note: Google Test filter = FuzzMeFuzz.HarnessString
134 [==========] Running 1 test from 1 test suite.
135 [----------] Global test environment set-up.
136 [----------] 1 test from FuzzMeFuzz
137 [ RUN      ] FuzzMeFuzz.HarnessString
138 [*] Corpus size:     1 | Edges covered:    130 | Fuzzing time:
    1.100805ms | Total runs:  1.00e+00 | Runs/secs:     1
139 [*] Corpus size:     2 | Edges covered:    130 | Fuzzing time:
    1.930784ms | Total runs:  2.00e+00 | Runs/secs:     2
140 [*] Corpus size:     3 | Edges covered:    132 | Fuzzing time:
    2.585246ms | Total runs:  3.00e+00 | Runs/secs:     3
141 [*] Corpus size:     4 | Edges covered:    132 | Fuzzing time:
    9.932545ms | Total runs:  4.00e+00 | Runs/secs:     4
```


Christopher Huth | 2023-01-25
© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.

BOSCH

Further Reading

- Short introduction to multiple analysis methods, focus on libFuzzer, some hands-on part (minimal example and real-world software: suricata) https://academy.code-intelligence.com/p/fuzzing-101
- Recent years have seen the development of novel techniques that lead to dramatic improvements in test generation and software testing. They now are mature enough to be assembled in a book – even with executable code. https://www.fuzzingbook.org/
- Recent Papers Related To Fuzzing https://wcventure.github.io/FuzzingPaper/
- Fuzz your open source software in OSS-Fuzz | Documentation for OSS-Fuzz (google.github.io)
- Tools with active research/development are AFLplusplus/LibAFL (github.com) and google/centipede (github.com)

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.
BOSCH

Thank you!

More:
Automated security testing to provide more protection from the start

Automated software testing by Bosch

https://www.bosch.com/stories/automated-security-testing/

Dr.-Ing.
Christopher Huth

Corporate Sector Research and Advance Engineering
Security, Privacy & Safety

christopher.huth@bosch.com

© Robert Bosch GmbH 2022. Alle Rechte vorbehalten, auch bzgl. jeder Verfügung, Verwertung, Reproduktion, Bearbeitung, Weitergabe sowie für den Fall von Schutzrechtsanmeldungen.