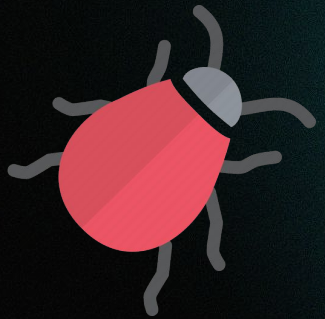
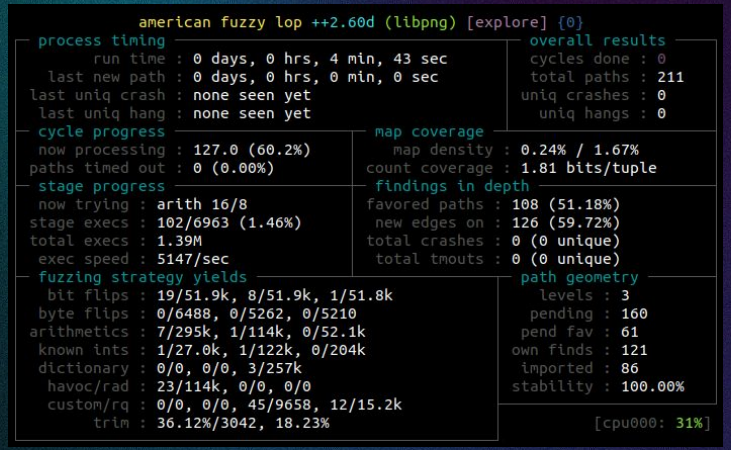
# Introduction To Fuzzing



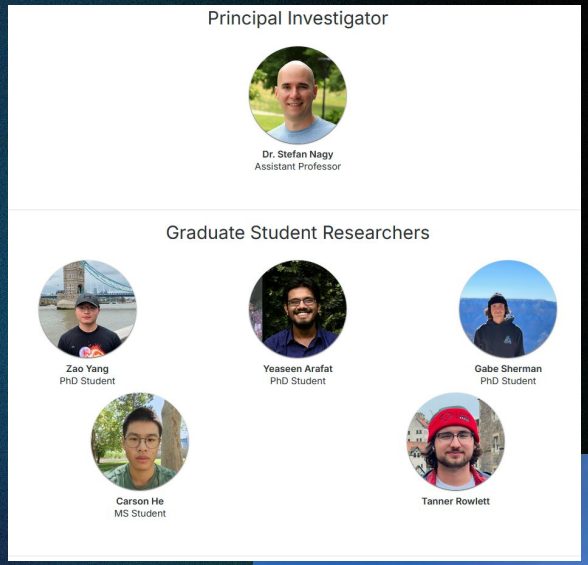



2/11/2026
Gabriel Sherman

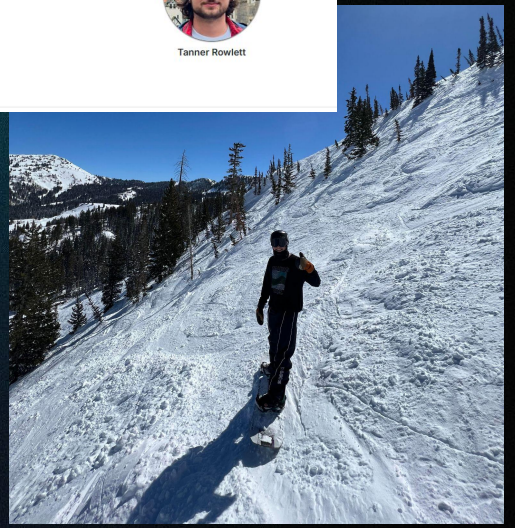
Slides adapted from Stefan Nagy

# About Me

- What I do:
  - Ph.D. student at the University of Utah
  - Performing research in cybersecurity
  - Uncovered over 50 bugs in software
  - Talk to me about the program!




- Other things about me:
  - Love being outside
  - Did my undergrad here as well
  - Have a weiner dog



Check out our lab: https://futures.cs.utah.edu/

# Software Is Buggy!

## This can have disastrous effects ...

**Ballista Botnet Exploits Unpatched TP-Link Vulnerability, Infects Over 6,000 Devices**

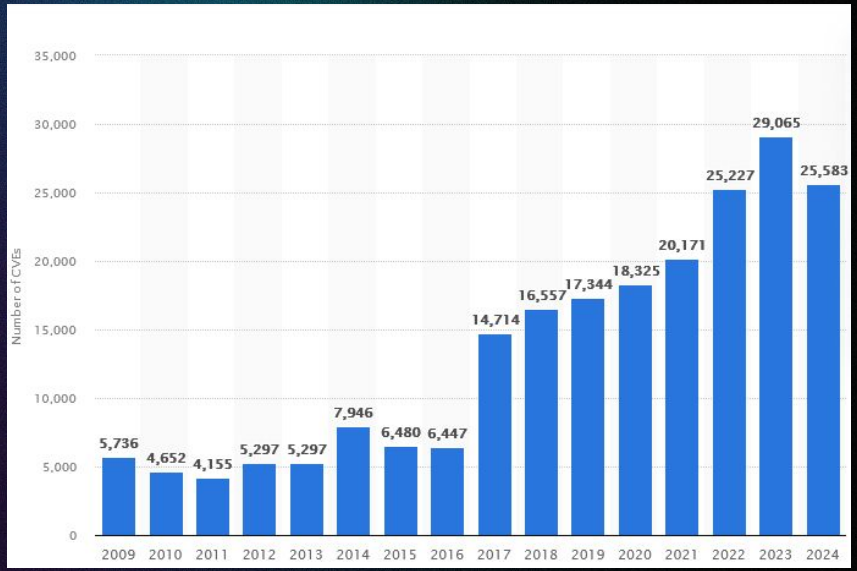
Mar 11, 2025 Ravie Lakshmanan

**The 'most serious' security breach ever is unfolding right now. Here's what you need to know.**

Dec. 20, 2021 at 10:00 pm | *Updated Dec. 20, 2021 at 10:25 pm*

**Heartbleed Bug: Tech firms urge password reset**

9 April 2014



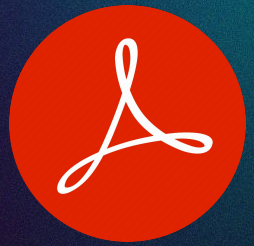

https://www.statista.com/statistics/500755/worldwide-common-vulnerabilities-and-exposures/

# Software Uses Inputs

Audio

Video

Photos



And more...

When these inputs are not properly handled, bugs occur!

# How do we find bugs?

## Manual Analysis

- Codebases are huge!
- Massive amounts of effort
- Still might miss something

## Static Analysis

- Analyze program without running it
- Quickly becomes computationally infeasible
- False positives and false negatives

## Dynamic Analysis

- Analyze program by running it
- What you see is what you get
- Minimal overhead

# Fuzzing

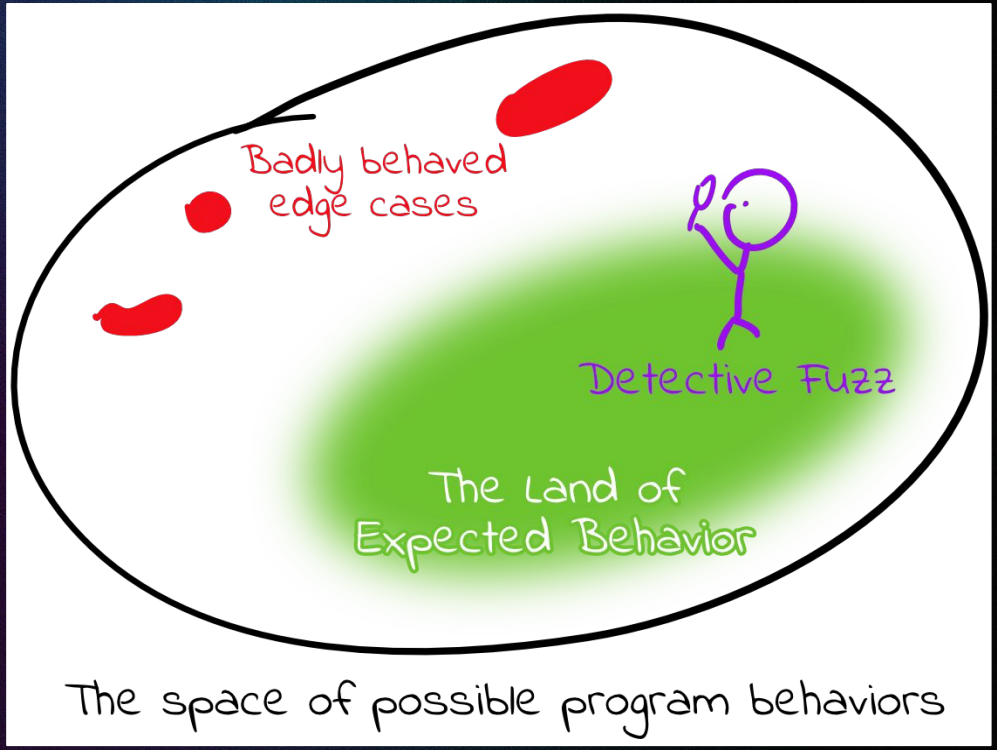
- A dynamic analysis technique that aims to execute a program as many times as possible with random inputs.
- Widely used across all aspects parts of the software ecosystem.
- Google's OSS-Fuzz project has found over 10,000 vulnerabilities and 36,000 bugs through fuzzing



# Fuzzing

The General Idea:

- Quickly generate a ton of random inputs
- Execute the program on each input. Does the program:
  - Crash?
  - Hang?
  - Do Nothing?



# What can you fuzz?

Everything

# What can you fuzz?

JANUS: Detecting Rendering Bugs in Web Browsers via Visual Delta Consistency

Chijin Zhou†, Quan Zhang†, Bingzhou Qian‡, and Yu Jiang†✉
†BNRist, Tsinghua University, Beijing, China
‡National University of Defense Technology, Changsha, China

FUZZILLI: Fuzzing for JavaScript JIT Compiler Vulnerabilities

Samuel Groß
Google
saleo@google.com

Simon Koch
TU Braunschweig
simon.koch@tu-braunschweig.de

Lukas Bernhard
Ruhr University Bochum
lukas.bernhard@rub.de

Thorsten Holz
CISPA Helmholtz Center for Information Security
holz@cispa.de

Martin Johns
TU Braunschweig
m.johns@tu-braunschweig.de

# What can you fuzz?

**WhiteFox: White-Box Compiler Fuzzing Empowered by Large Language Models**


CHENYUAN YANG, University of Illinois at Urbana-Champaign, USA
YINLIN DENG, University of Illinois at Urbana-Champaign, USA
RUNYU LU, Huazhong University of Science and Technology, China
JIAYI YAO, Chinese University of Hong Kong, China
JIAWEI LIU, University of Illinois at Urbana-Champaign, USA
REYHANEH JABBARVAND, University of Illinois at Urbana-Champaign, USA
LINGMING ZHANG, University of Illinois at Urbana-Champaign, USA


**Finding and Understanding Bugs in C Compilers**


Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

# What can you fuzz?



**Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing**

Tobias Scharnowski[1], Nils Bars[1], Moritz Schloegel[1], Eric Gustafson[2], Marius Muench[3], Giovanni Vigna[2,4], Christopher Kruegel[2], Thorsten Holz[1] and Ali Abbasi[1]

[1]*Ruhr-Universität Bochum,* [2]*UC Santa Barbara,* [3]*Vrije Universiteit Amsterdam,* [4]*VMware*

**SNIPUZZ: Black-box Fuzzing of IoT Firmware via Message Snippet Inference**

Xiaotao Feng
Swinburne University of Technology
Australia

Ruoxi Sun
The University of Adelaide
Australia

Xiaogang Zhu*
Swinburne University of Technology
Australia

Minhui Xue
The University of Adelaide
Australia

Sheng Wen*
Swinburne University of Technology
Australia

Dongxi Liu
CSIRO Data61
Australia

Surya Nepal
CSIRO Data61
Australia

Yang Xiang
Swinburne University of Technology
Australia

# What can you fuzz?


OpenSSL

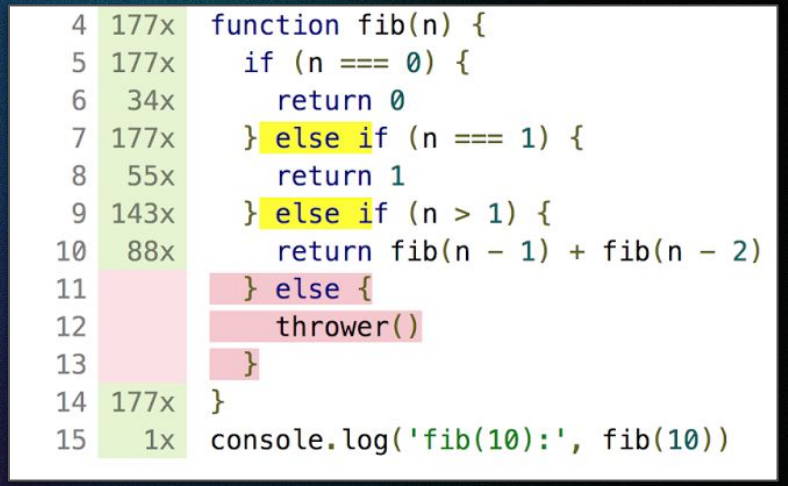
# Questions?

# AFLplusplus

- *Coverage guided*, *grey-box* fuzzer
- Most widely used fuzzing engine out there (5.5k github stars)
- High portability and adaptability
- Learn how to use this when learning fuzzing!
- Some useful links:
  - https://github.com/AFLplusplus/AFLplusplus
  - https://aflplus.plus/

# Coverage-guided Fuzzing

- Addresses the challenge of determining if an input is interesting or meaningful
- Keep and further mutate inputs that reach new coverage in the program under test
- No new coverage means the input was likely uninteresting!



```
 4  177x  function fib(n) {
 5  177x    if (n === 0) {
 6   34x      return 0
 7  177x    } else if (n === 1) {
 8   55x      return 1
 9  143x    } else if (n > 1) {
10   88x      return fib(n - 1) + fib(n - 2)
11           } else {
12             thrower()
13           }
14  177x  }
15    1x  console.log('fib(10):', fib(10))
```

# A Simple Example

bgatradf

New coverage! –
Mutate this input

Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
}
```

# A Simple Example

bagtfd

No new coverage

Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
}
```

# A Simple Example

aertstbwr

New coverage! –
Mutate this input

Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
}
```

# A Simple Example

abwrertst

New coverage! –
Mutate this input

Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
}
```

# A Simple Example

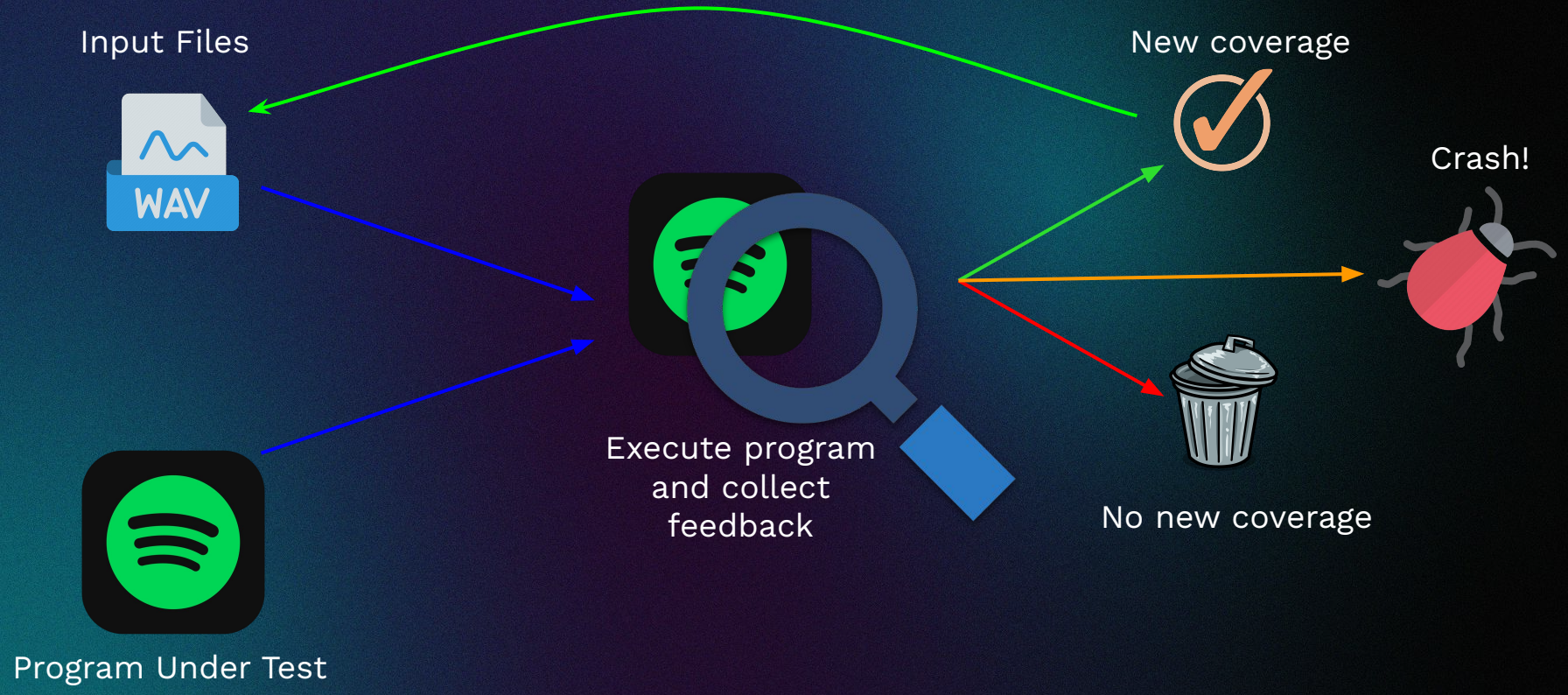
acwrertst

Crash!

Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
}
```

# The Overall Process



# Instrumentation

- *Instrument* the program to report when regions of code are reached.
- Open Source?
  - Instrument at compilation time with afl-clang-fast or afl-clang-fast++
- Closed Source?
  - Instrument on the fly using afl-qemu

Uninstrumented Program

```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
```

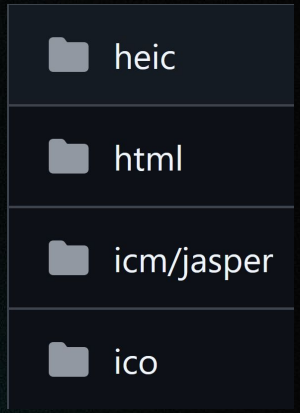
Instrumented Program

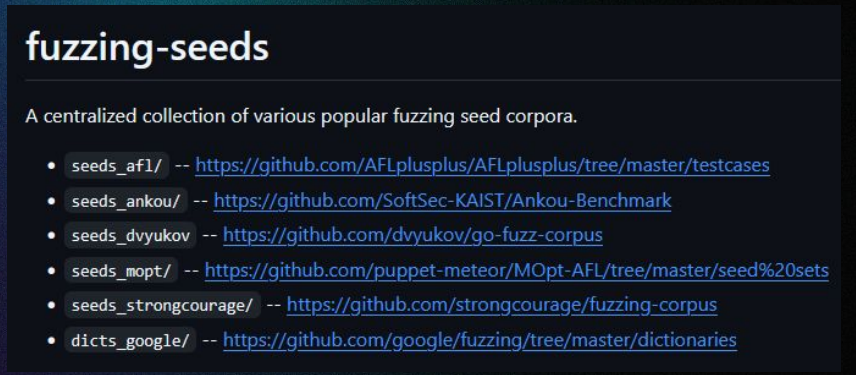
```
if(input[0] == ‘a’){
        if(input[1] == ‘b’){
                return 2;
        }
        if(input[1] == ‘c’){
                Crash!
        }
```

<Hey AFL, I’m being executed!>

# Get a headstart

- Fuzzers generate inputs and mutate them. Give it a headstart by providing *seeds*.
- Fuzzing seeds are *examples* of inputs the program may expect. This lets the fuzzer operate on and mutate these rather than starting from scratch.
- Seeds are *program dependent*.
- Some examples
  - Adobe Acrobat: Different PDF files
  - Spotify: Different audio files
  - IMessage: .db files




https://github.com/FuturesLab/fuzzing-seeds/

# Questions?

# Library Fuzzing

Libraries are used *everywhere* in software. Bugs and vulnerabilities that affect them not only affect the library itself, but software that depends on it!

**For example...**

XZ-Utils Backdoor: Adversary placed a backdoor in the liblzma library. Granted them remote access to linux systems.

**Linux could have been brought down by backdoor found in widely used utility**

The malicious code modifies functions within a data compression library that is a foundational part of several Linux distributions

OpenSSL Heartbleed Vulnerability: Improper input validation allowed attackers to leak sensitive information (secret keys, passwords, etc). 500,000 servers were immediately vulnerable.

# Fuzzing Harnesses

- Libraries depend on other programs to call their functionality (like the fopen function in the standard C library)
- Fuzzing harnesses are lightweight programs that consume fuzzer input and pass it into the library.
- The library is like a toolbox, while the harness is actually using the tools.
- I wrote a blog about how to get started with harnessing: https://gabe-sherman.github.io/2025-09-17-beginning_harnessing/

**LibUCL**

**Fuzzing Harness**

```c
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include "ucl.h"

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
        // If size is 0 we need a null-terminated string.
        // We dont null-terminate the string and by the design
        // of the API passing 0 as size with non null-terminated string
        // gives undefined behavior.
        if(size==0){
                return 0;
        }
        struct ucl_parser *parser;
        parser = ucl_parser_new(0);

        ucl_parser_add_string(parser, (char *)data, size);

        if (ucl_parser_get_error(parser) != NULL) {
                return 0;
        }

        ucl_parser_free (parser);
        return 0;
}
```

# Library Fuzzing Overview

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

#include <libical/ical.h>

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size)
{
    char *ical_string = (char*)malloc(size + 1);
    memcpy(ical_string, data, size);
    ical_string[size] = '\0';

    icalcomponent *component = icalparser_parse_string(ical_string);
    icalcomponent_free(component);

    free(ical_string);

    return 0;
}
```

**Harness**

**Inputs**

**Feed inputs to the library in a *meaningful way***

**Fuzzing Engine**

**Coverage Information**

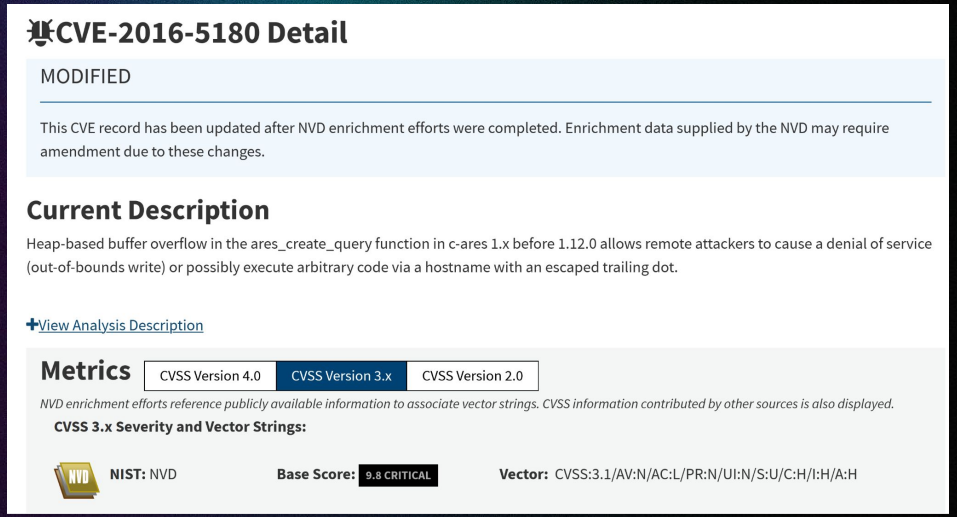
**Library**

# Workshop

https://github.com/gabe-sherman/cybersec_fuzzing_demo

You need a Linux environment, or use WSL(Windows subsystem for linux)

1. Fuzz the linux “file” command.
2. Fuzz the LibUCL library using a pre-written harness.
3. Find a CVE in c-ares!



**CVE-2016-5180 Detail**

MODIFIED

This CVE record has been updated after NVD enrichment efforts were completed. Enrichment data supplied by the NVD may require amendment due to these changes.

**Current Description**

Heap-based buffer overflow in the ares_create_query function in c-ares 1.x before 1.12.0 allows remote attackers to cause a denial of service (out-of-bounds write) or possibly execute arbitrary code via a hostname with an escaped trailing dot.

+View Analysis Description

**Metrics** CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

*NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.*

**CVSS 3.x Severity and Vector Strings:**

**NIST:** NVD **Base Score:** 9.8 CRITICAL **Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

# Use the Internet!

There are a ton of blog posts out there describing the different aspects of fuzzing and harnessing. These are some of the best resources for learning.

https://bushido-sec.com/index.php/2025/01/03/fuzzing-harness-guide/

https://google.github.io/clusterfuzz/setting-up-fuzzing/heartbleed-example/

https://www.mayhem.security/blog/firmware-fuzzing-101

https://www.fuzzingbook.org/

https://trustfoundry.net/2020/01/22/introduction-to-triaging-fuzzer-generated-crashes/

# Join Research!

Network Security

Cryptography

Mobile Security

Fuzzing

And more ... (AI security, systems security, exploits, cyber-physical systems, etc.)