

The background of the image is a vibrant sunset or sunrise over a savanna landscape. Silhouettes of acacia trees are scattered across the left side. In the center, a large sun is partially obscured by the branches of one tree. To the right, a herd of elephants is walking across the horizon. Several birds, possibly vultures, are flying in the orange and red sky.

# In The Wild Introduction

## Sudo Heap Buffer Overflow

**CVE-2021-3156**



- 본 발표에서 설명하는 모든 개념은 Linux 운영 체제를 기반으로 한다.
- 또한 liveoverflow의 pwnedit 연구를 기반으로 한 공부 발표이다.

# About Me :)

# About Me :)

- **What is your name?**

# About Me :)

- **What is your name?**
  - **Kim Dae Yeong**

# About Me :)



- **What is your name?**
  - **Kim Dae Yeong**

# **Table of Contents...**

# Table of Contents

# Table of Contents

- **What is Sudo?**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**
- **What is Fuzzing?**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**
- **What is Fuzzing?**
- **What is CVE-2021-3156**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**
- **What is Fuzzing?**
- **What is CVE-2021-3156**
  - **Find vulnerabilities with fuzzing**

# Table of Contents

- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**
- **What is Fuzzing?**
- **What is CVE-2021-3156**
  - **Find vulnerabilities with fuzzing**
  - **Look at the Crash Testcase**

# Table of Contents

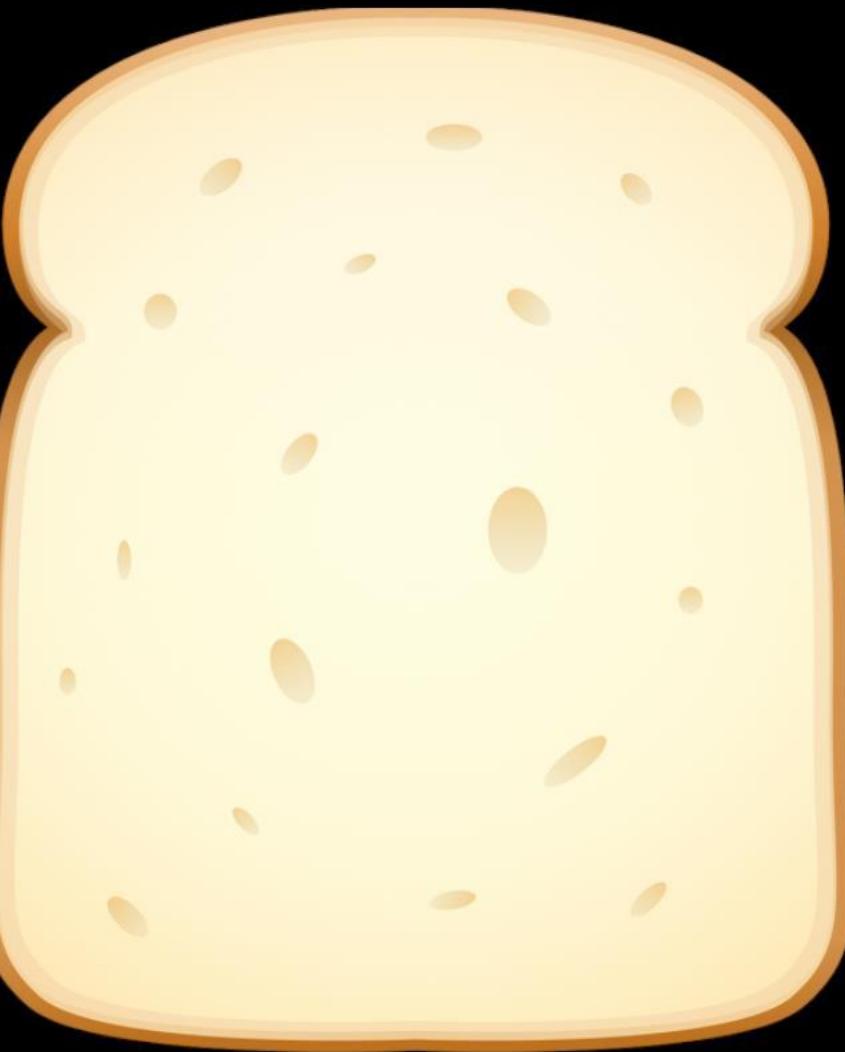
- **What is Sudo?**
- **What is Heap Memory?**
- **What is BoF?**
- **What is Fuzzing?**
- **What is CVE-2021-3156**
  - **Find vulnerabilities with fuzzing**
  - **Look at the Crash Testcase**
  - **Minimizing AFL Testcase**

# **What is Sudo?**

# **What is Sudo?**

# What is Sudo?

- Unix 계열 OS 유틸리티



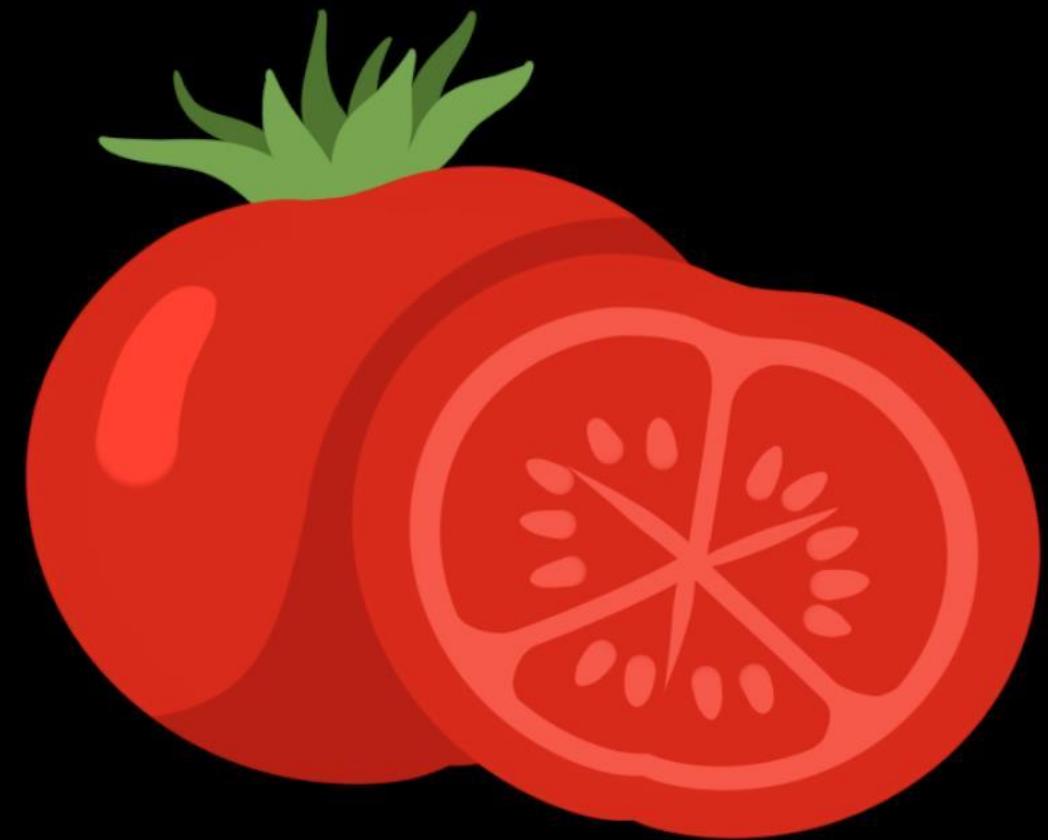
# What is Sudo?

- Unix 계열 OS 유틸리티
- GNU 코어 유틸리티는 아니다



# What is Sudo?

- Unix 계열 OS 유틸리티
- GNU 코어 유틸리티는 아니다
- 사용자는 프로그램을 실행하기 위해 루트 권한을 빌려온다.



# What is Sudo?

- Unix 계열 OS 유틸리티
- GNU 코어 유틸리티는 아니다
- 사용자는 프로그램을 실행하기 위해 루트 권한을 빌려온다.
- su와 달리 루트의 비밀번호가 필요한다.



# **What is Heap?**

# **What is Heap Memory?**

# What is Heap Memory?

- 힙 데이터 구조와 힙 메모리는 관련이 없다  
**(Pool)**



# What is Heap Memory?

- 힙 데이터 구조와 힙 메모리는 관련이 없다  
(Pool)
- 동적으로 할당된 메모리를 관리하는데 사용  
되는 메모리 영역



# What is Heap Memory?

- 힙 데이터 구조와 힙 메모리는 관련이 없다  
(Pool)
- 동적으로 할당된 메모리를 관리하는데 사용  
되는 메모리 영역
- 스택과 달리 낮은 주소에서 높은 주소로 확장  
된다.



# **What is BoF?**

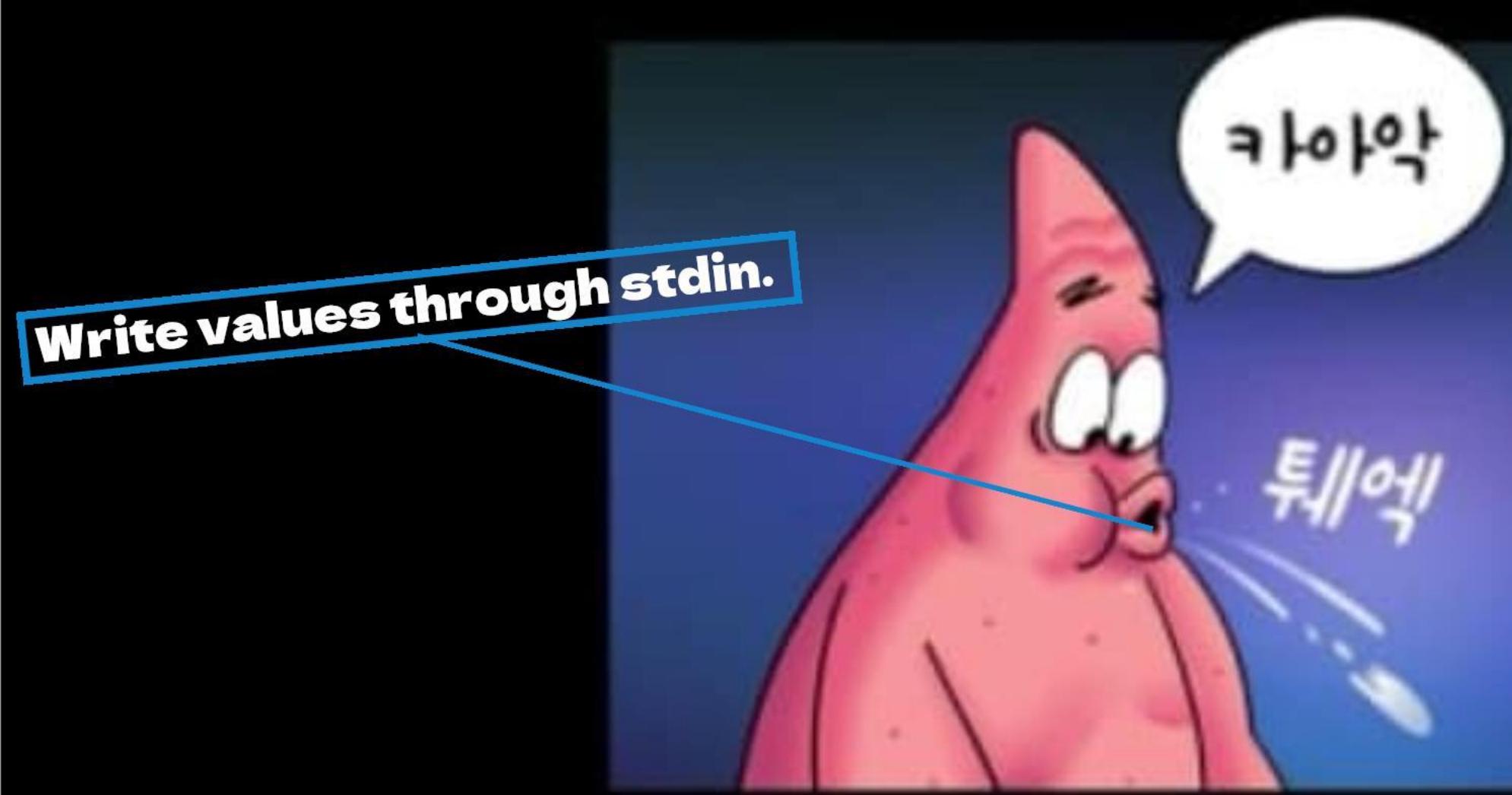
# **What is BoF?**

# What is BoF?

데이터가 한 곳에서 다른 곳으로 전송되는 동안 일시적으로 그 데이터를 보관하는 메모리의 공간이다.  
(쓰기와 읽기 모두 버퍼를 사용한다.)

# What is BoF?

데이터가 한 곳에서 다른 곳으로 전송되는 동안 일시적으로 그 데이터를 보관하는 메모리의 공간이다.  
(쓰기와 읽기 모두 버퍼를 사용한다.)



# What is BoF?

데이터가 한 곳에서 다른 곳으로 전송되는 동안 일시적으로 그 데이터를 보관하는 메모리의 공간이다.  
(쓰기와 읽기 모두 버퍼를 사용한다.)



# What is BoF?

버퍼는 일반적으로 배열을 의미하지만, 모든 배열이 버퍼는 아니다

# What is BoF?

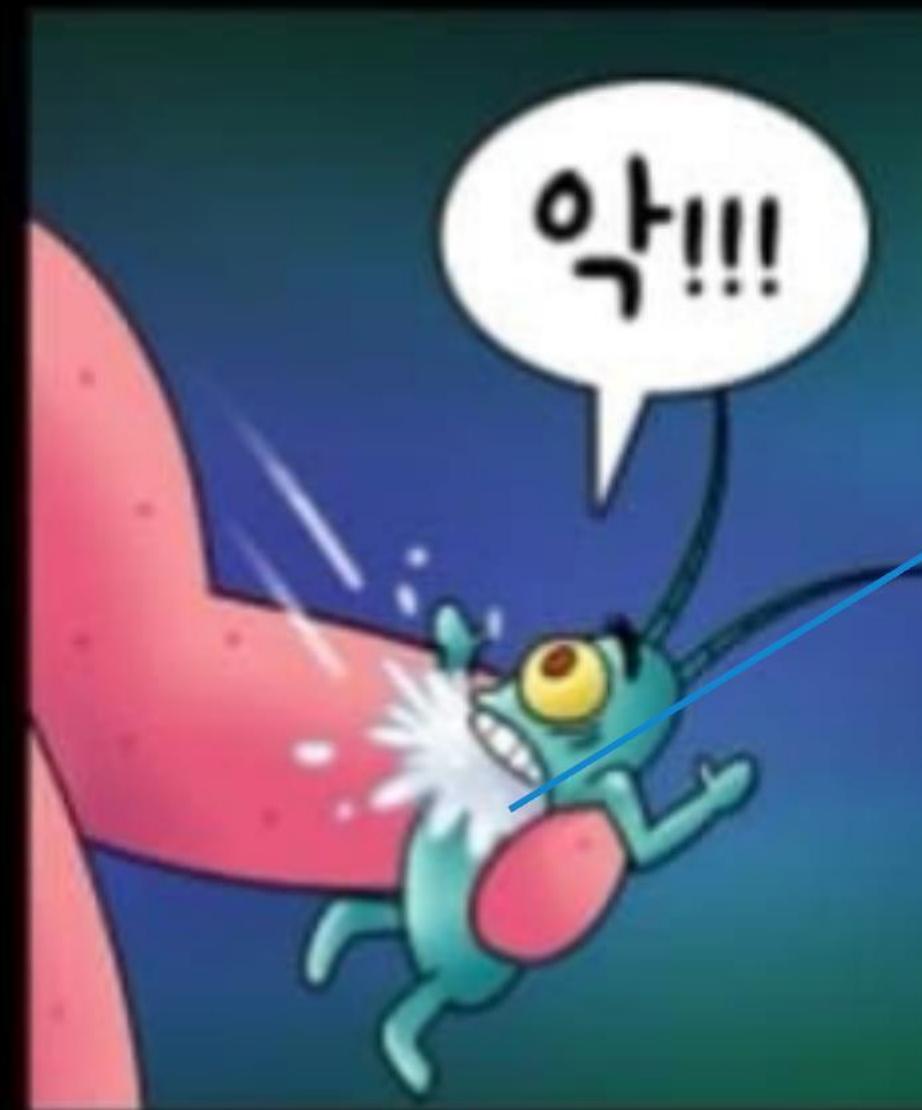
버퍼는 일반적으로 배열을 의미하지만, 모든 배열이 버퍼는 아니다

```
#include <stdio.h>
int main(){
    char buffer[10];
    int len;
    len = read(0, buffer, 9);
    buffer[len] = '\x00';
    write(1, buffer, 9);
    return 0;
}
```

# What is BoF?

버퍼는 일반적으로 배열을 의미하지만, 모든 배열이 버퍼는 아니다

```
#include <stdio.h>
int main(){
    char buffer[10];
    int len;
    len = read(0, buffer, 9);
    buffer[len] = '\x00';
    write(1, buffer, 9);
    return 0;
}
```



Buffer

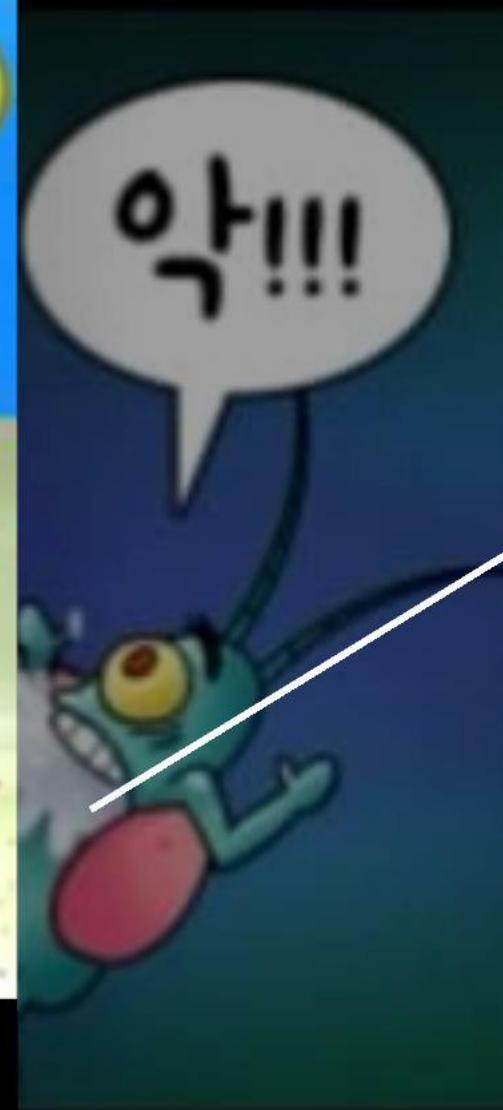
# What is BoF?

Buffer usually refers to an array, But not all arrays are buffers.

```
#include <stdio.h>
int main(){
    char buffer[10];
    int len;
    len = read(0, buffer, 10);
    buffer[len] = '\x00';
    write(1, buffer, 9);
    return 0;
}
```



NO!!!



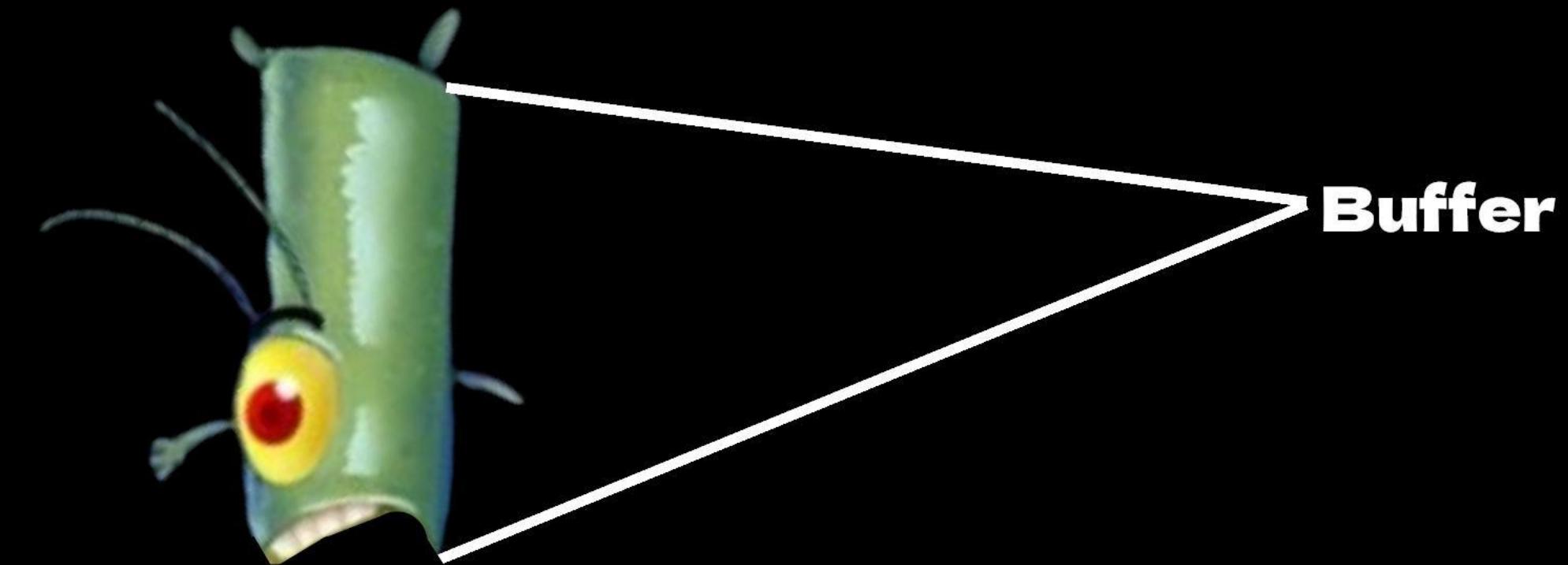
Buffer

# What is BoF?

할당된 메모리 공간 뷔퍼의 크기를 초과하여 입력한다.

# What is BoF?

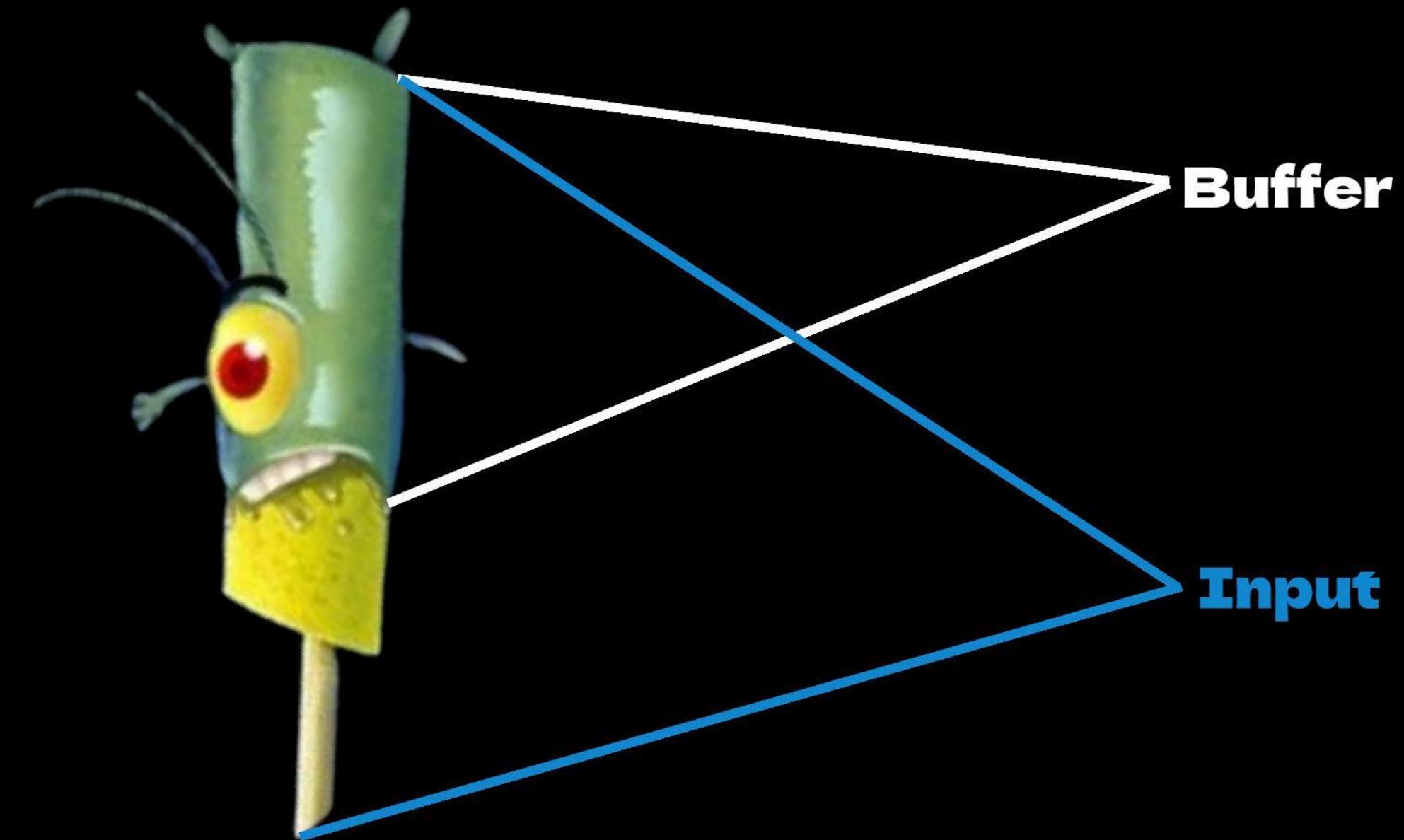
할당된 메모리 공간 버퍼의 크기를 초과하여 입력한다.



**Buffer**

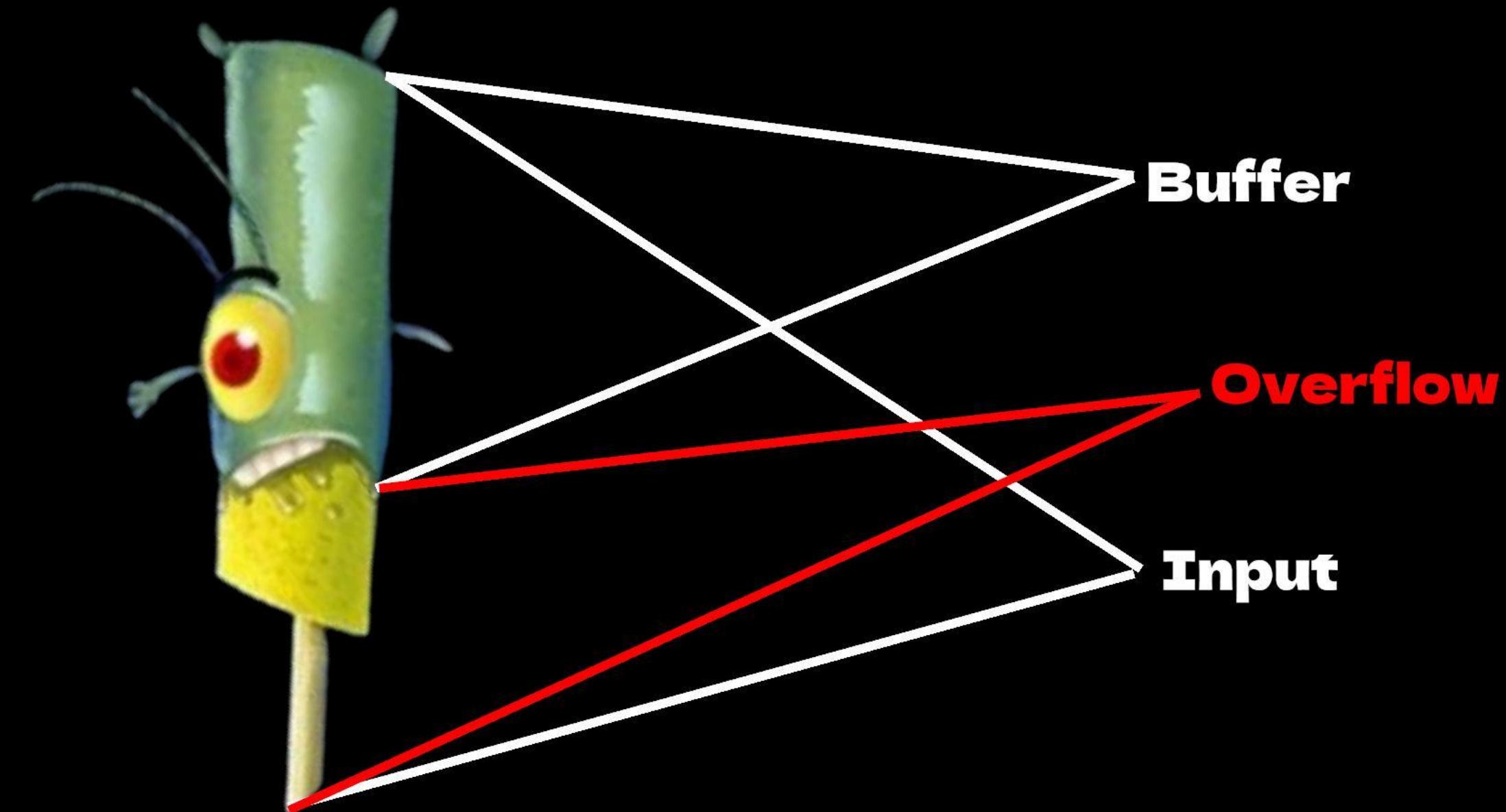
# What is BoF?

할당된 메모리 공간 버퍼의 크기를 초과하여 입력한다.



# What is BoF?

할당된 메모리 공간 버퍼의 크기를 초과하여 입력한다.



# What is BoF?

버퍼 오버플로우는 버퍼와 관련된 범위를 벗어난 읽기/쓰기의 한 형태이다

# What is BoF?

버퍼 오버플로우는 버퍼와 관련된 범위를 벗어난 읽기/쓰기의 한 형태이다

```
#include <stdio.h>
int main(){
    char buffer[10];
    read(0, buffer, 200);
    return 0;
}
```

## BoF Write

# What is BoF?

버퍼 오버플로우는 버퍼와 관련된 범위를 벗어난 읽기/쓰기의 한 형태이다

```
#include <stdio.h>
int main(){
    char buffer[10];
    read(0, buffer, 200);
    return 0;
}
```

**BoF Write**



# What is BoF?

버퍼 오버플로우는 버퍼와 관련된 범위를 벗어난 읽기/쓰기의 한 형태이다

```
#include <stdio.h>
int main(){
    char leak[16] = "Leak value";
    char buffer[10];
    read(0, buffer, 10);
    printf("%s", buffer);
    return 0;
}
```

BoF Read

# What is BoF?

버퍼 오버플로우는 버퍼와 관련된 범위를 벗어난 읽기/쓰기의 한 형태이다



A terminal window titled 'z3rodae0@DESKTOP-F8QRQJU:~\$' displays a C program. The code includes a leak of memory at address 0, reading 10 bytes into a buffer, and printing the buffer's contents. The terminal shows the output of the exploit.

```
#include <stdio.h>
int main(){
    char leak[16] = "Leak value";
    char buffer[10];
    read(0, buffer, 10);
    printf("%s", buffer);
    return 0;
}
```

```
#include <stdio.h>
int main(){
    char leak[16] = "Leak value";
    char buffer[10];
    read(0, buffer, 10);
    printf("%s", buffer);
    return 0;
}
```

**BoF Read**

# What is FUZZing?



# What is Fuzzing?

# What is Fuzzing?

소프트웨어 취약점을 테스트하는 기술 중 하나입니다.

# What is Fuzzing?

소프트웨어 취약점을 테스트하는 기술 중 하나입니다.

- 1989년 위스콘신-매디슨 대학의 바튼 밀러(Barton Miller) 교수와 그의 학생들이 개발한 기술이다.

# What is Fuzzing?

소프트웨어 취약점을 테스트하는 기술 중 하나입니다.

- 1989년 위스콘신-매디슨 대학의 바튼 밀러(Barton Miller) 교수와 그의 학생들이 개발한 기술이다.
- 브라우저, 시스템 커널, 대부분의 API 및 기타 애플리케이션을 포함하여 입력이 있는 거의 모든 소프트웨어는 퍼징의 대상이 될 수 있다.

# What is Fuzzing?

소프트웨어 취약점을 테스트하는 기술 중 하나입니다.

- 1989년 위스콘신-매디슨 대학의 바튼 밀러(Barton Miller) 교수와 그의 학생들이 개발한 기술이다.
- 브라우저, 시스템 커널, 대부분의 API 및 기타 애플리케이션을 포함하여 입력이 있는 거의 모든 소프트웨어는 퍼징의 대상이 될 수 있다.
- 퍼징에는 여러 가지 방법이 있다.(예: Differential Testing)

# **What is Fuzzing?**

**Generation-based VS Mutation-based**

# What is Fuzzing?

**Generation-based VS Mutation-based**



# What is Fuzzing?

**Generation-based VS Mutation-based**



- 대상 프로토콜이나 프로그램을 알아야 한다.

# What is Fuzzing?

## Generation-based VS Mutation-based



- 대상 프로토콜이나 프로그램을 알아야 한다.
- 자신만의 테스트 케이스를 디자인해야 한다.

# What is Fuzzing?

## Generation-based VS Mutation-based



- 대상 프로토콜이나 프로그램을 알아야 한다.
- 자신만의 테스트 케이스를 디자인해야 한다.
- **Mutation-based**보다 퍼징의 정확도가 높다.

# What is Fuzzing?

**Generation-based VS Mutation-based**



# What is Fuzzing?

## Generation-based VS Mutation-based

- 대상 프로토콜이나 프로그램의 구조를 잘 알 필요가 없다.



# What is Fuzzing?

## Generation-based VS Mutation-based

- 대상 프로토콜이나 프로그램의 구조를 잘 알 필요가 없다.
- 주어진 입력 시드를 지속적으로 변경 (Mutation)하여 새로운 시드를 생성한다.



# What is Fuzzing?

## Generation-based VS Mutation-based

- 대상 프로토콜이나 프로그램의 구조를 잘 알 필요가 없다.
- 주어진 입력 시드를 지속적으로 변경 (Mutation)하여 새로운 시드를 생성한다.
- Dumb 퍼징이라고도 불린다.



# **What is Fuzzing?**

## **Code Coverage**

# What is Fuzzing?

## Code Coverage

- 함수 내에 특정 코드를 삽입하여 프로그램 실행 흐름을 계측한다.

# What is Fuzzing?

## Code Coverage

- 함수 내에 특정 코드를 삽입하여 프로그램 실행 흐름을 계측한다.
- 코드 커버리지를 계산하려면 **Instrumentation**이 필요하다.

# What is Fuzzing?

## Code Coverage

- 함수 내에 특정 코드를 삽입하여 프로그램 실행 흐름을 계측한다.
- 코드 커버리지를 계산하려면 **Instrumentation**이 필요하다.
- **Instrumentation** 코드가 프로그램의 분기문에 삽입된 후 실행 여부에 따라 코드커버리지를 계산한다.

# What is Fuzzing?

## Code Coverage

- 함수 내에 특정 코드를 삽입하여 프로그램 실행 흐름을 계측한다.
- 코드 커버리지를 계산하려면 Instrumentation이 필요하다.
- Instrumentation 코드가 프로그램의 분기문에 삽입된 후 실행 여부에 따라 코드커버리지를 계산한다.
- 코드 커버리지의 경우 함수, 조건문, 분기문 등 실행 흐름이 분기되는 코드에 Instrumentation 계측이 삽입되어 계산된다.

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
void func_2(){ printf("%s\n", __func__); }
int main(){
    int a;
    scanf("%d", &a);
    a ? func_1() : func_2();
    return 0;
}
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
void func_2(){ printf("%s\n", __func__); }
int main(){
    int a;
    scanf("%d", &a);
    a ? func_1():func_2();
    return 0;
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    __afl_maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        __afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        __afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
void func_2(){ printf("%s\n", __func__); }
int main(){
    int a;
    scanf("%d", &a);
    a ? func_1() : func_2();
    return 0;
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    _afl_maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
void func_2(){ printf("%s\n", __func__); }
int main(){
    int a;
    scanf("%d", &a);
    a ? func_1(): func_2();
    return 0;
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    _afl_maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
void func_2(){ printf("%s\n", __func__); }
int main(){
    int a;
    scanf("%d", &a);
    a ?func_1():func_2();
    return 0;
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    _afl_maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
int __fastcall func_1(__int64 a1, __int64 a2, __int64 a3)
{
    __alet a,
    scanf("%d", &a);
    a ? func_1() : func_2();
    return 0;
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    __alet maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        __alet maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        __alet maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# What is Fuzzing?

## Code Coverage(afl-compiler)

```
#include <stdio.h>
void func_1(){ printf("%s\n", __func__); }
int __fastcall func_1(__int64 a1, __int64 a2, __int64 a3)
{
    __afl_maybe_log(a1, a2, a3, 65507LL);
    return puts("func_1");
}
int __fastcall func_2(__int64 a1, __int64 a2, __int64 a3)
{
    __afl_maybe_log(a1, a2, a3, 36369LL);
    return puts("func_2");
}
```

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     __int64 v3; // rcx
4     __int64 v4; // rcx
5     __int64 v6; // [rsp+0h] [rbp-B0h]
6     __int64 v7; // [rsp+8h] [rbp-A8h]
7     __int64 v9; // [rsp+20h] [rbp-90h]
8     int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF
9
10    v9 = v3;
11    __afl_maybe_log(argc, argv, envp, 23617LL);
12    __isoc99_scanf(&unk_402004, v10, envp, v9);
13    v6 = (unsigned int)v10[0];
14    if ( v10[0] )
15    {
16        __afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
17        func_1((__int64)&unk_402004, (__int64)v10, v6);
18    }
19    else
20    {
21        v7 = v4;
22        __afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
23        func_2(&unk_402004, v10, v6, v7);
24    }
25    return 0;
26 }
```

# **What is Fuzzing?**

## **American Fuzzy Lop Plusplus**

# What is Fuzzing?

## American Fuzzy Lop Plusplus



# What is Fuzzing?

## American Fuzzy Lop Plusplus

- Coverage-based 퍼저



# What is Fuzzing?

## American Fuzzy Lop Plusplus

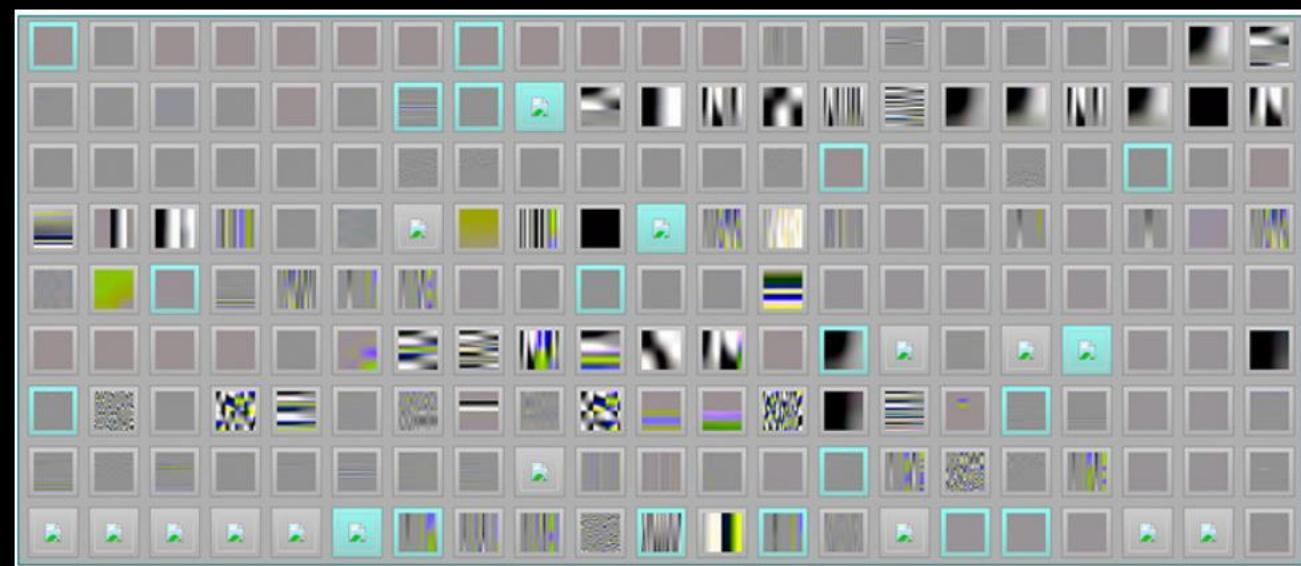
- Coverage-based 퍼져
- 컴파일 타입과 런타임에 계측 코드를 삽입하여  
코드 커버리지를 측정



# What is Fuzzing?

## American Fuzzy Lop Plusplus

- Coverage-based 퍼져
- 컴파일 타임과 런타임에 계측 코드를 삽입하여  
코드 커버리지를 측정
- 테스트 케이스 생성 및 변형(Mutation)

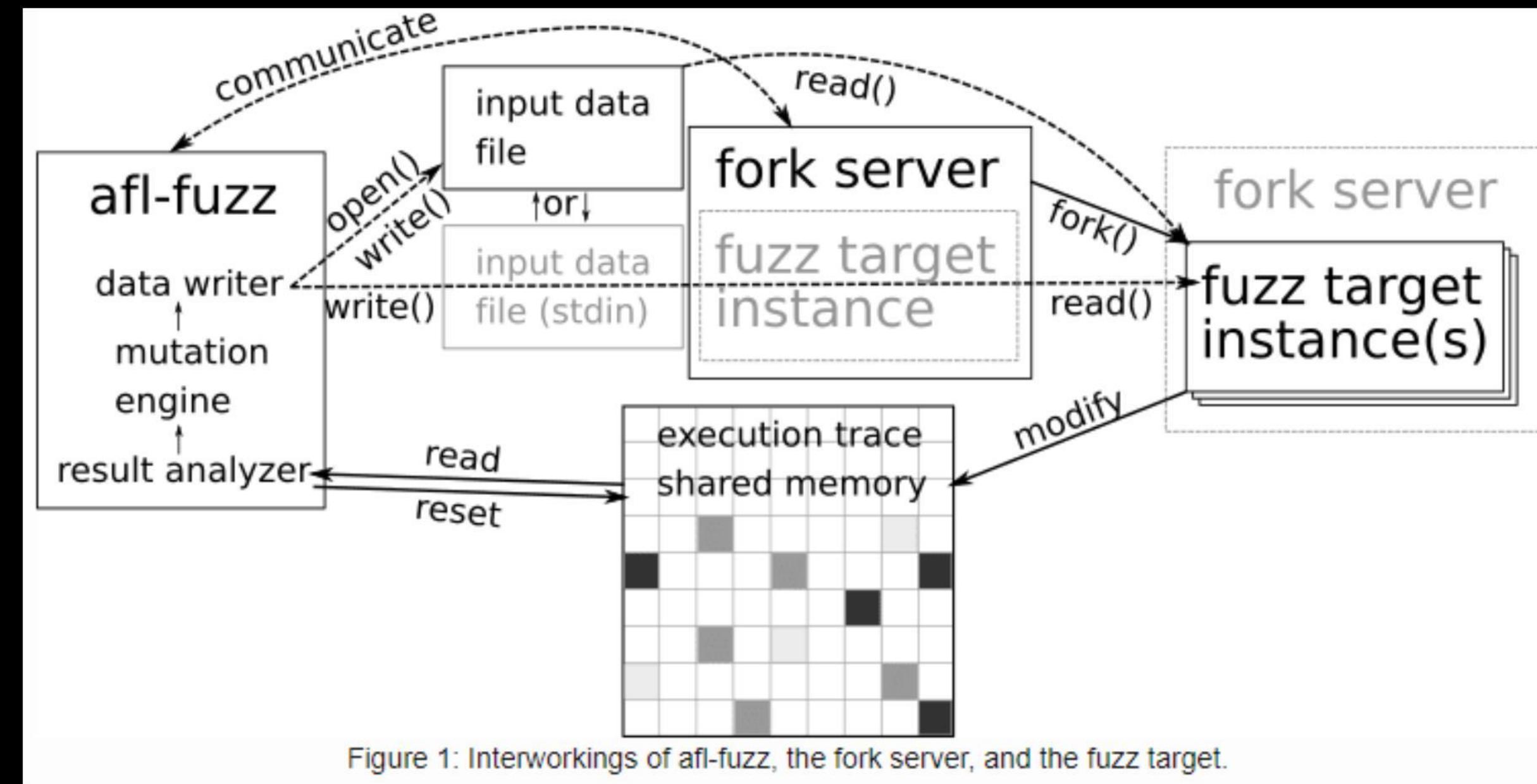


# **What is Fuzzing?**

## **AFL Structure**

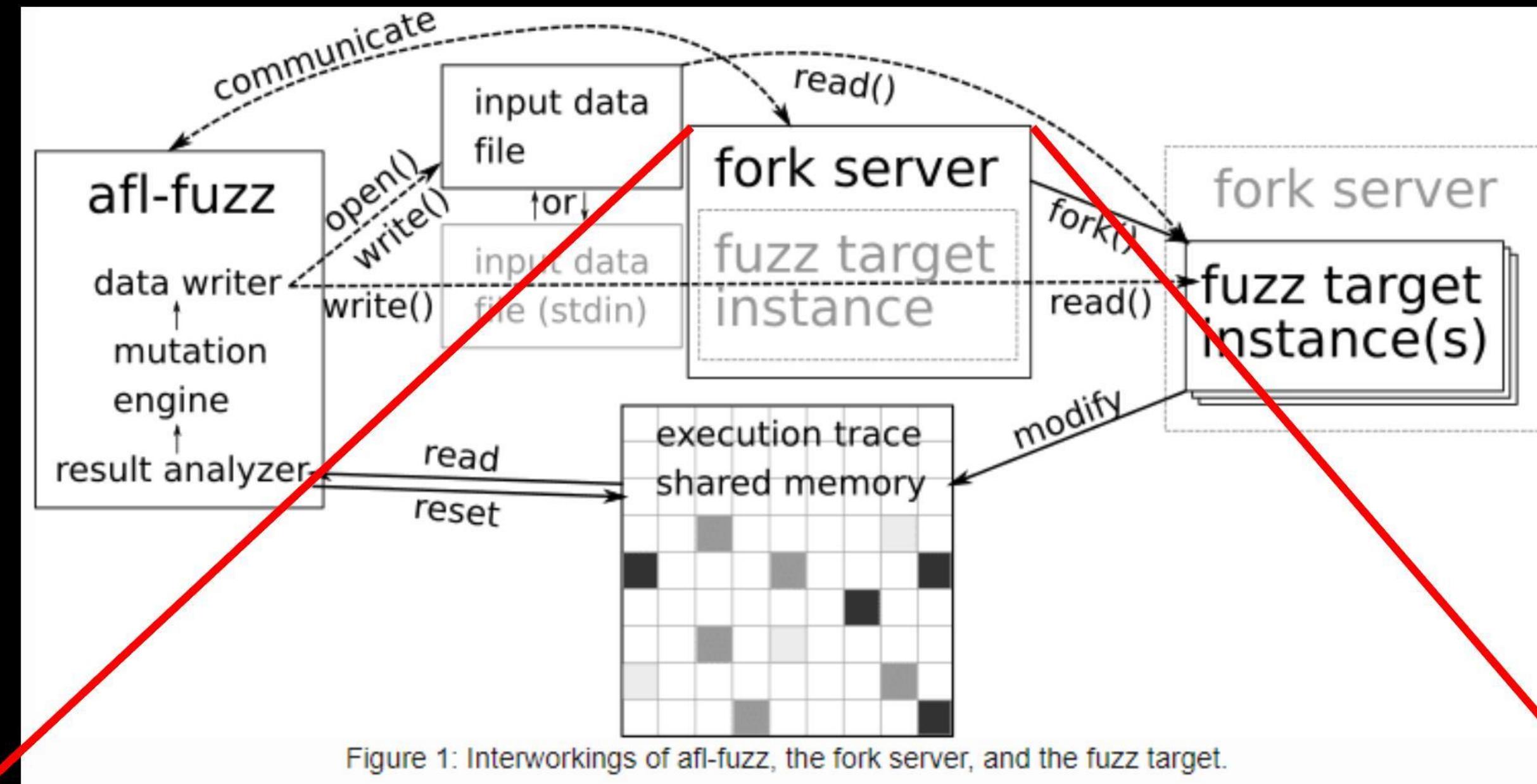
# What is Fuzzing?

## AFL Structure



# What is Fuzzing?

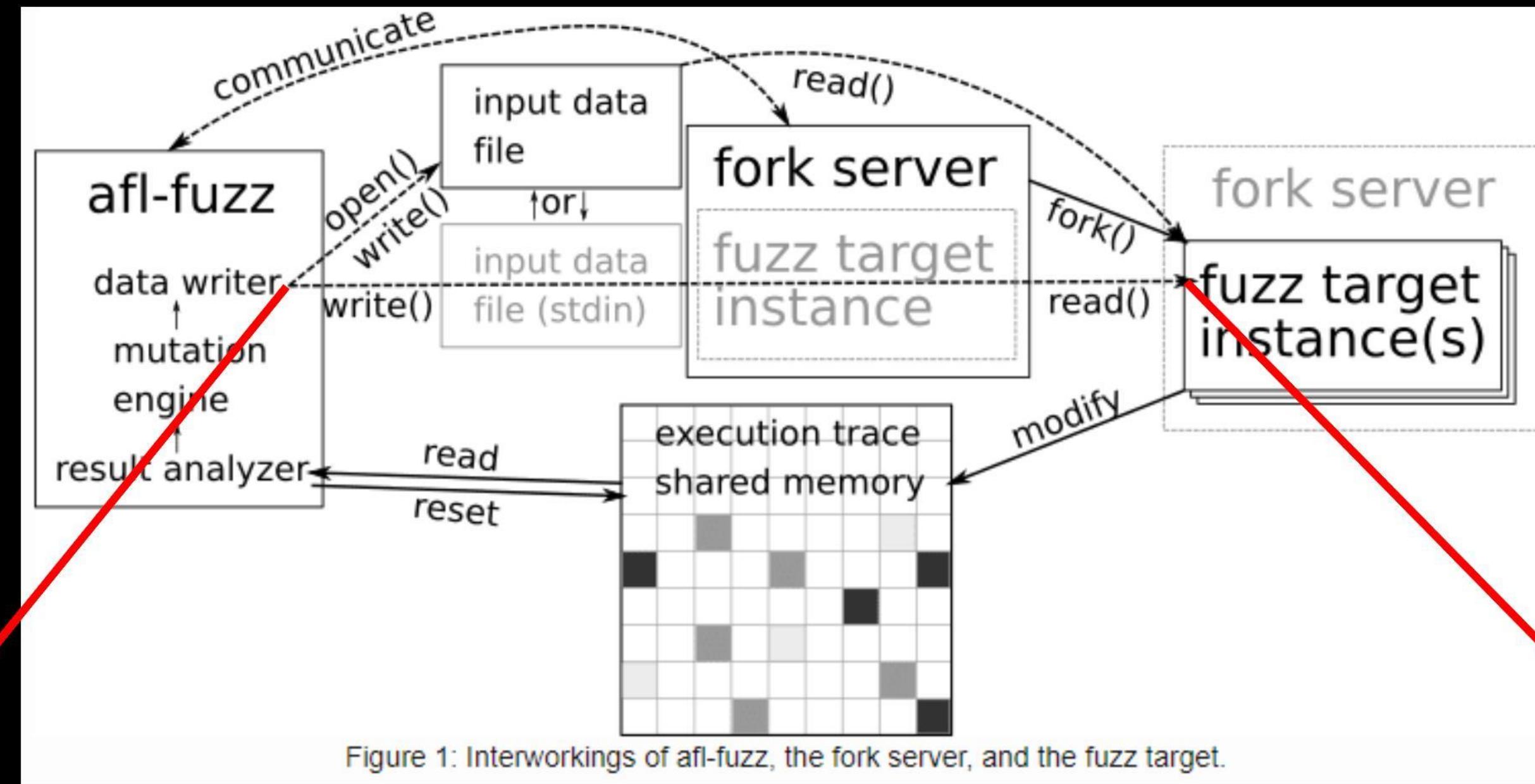
## AFL Structure



Create the program you want to fuzz with

# What is Fuzzing?

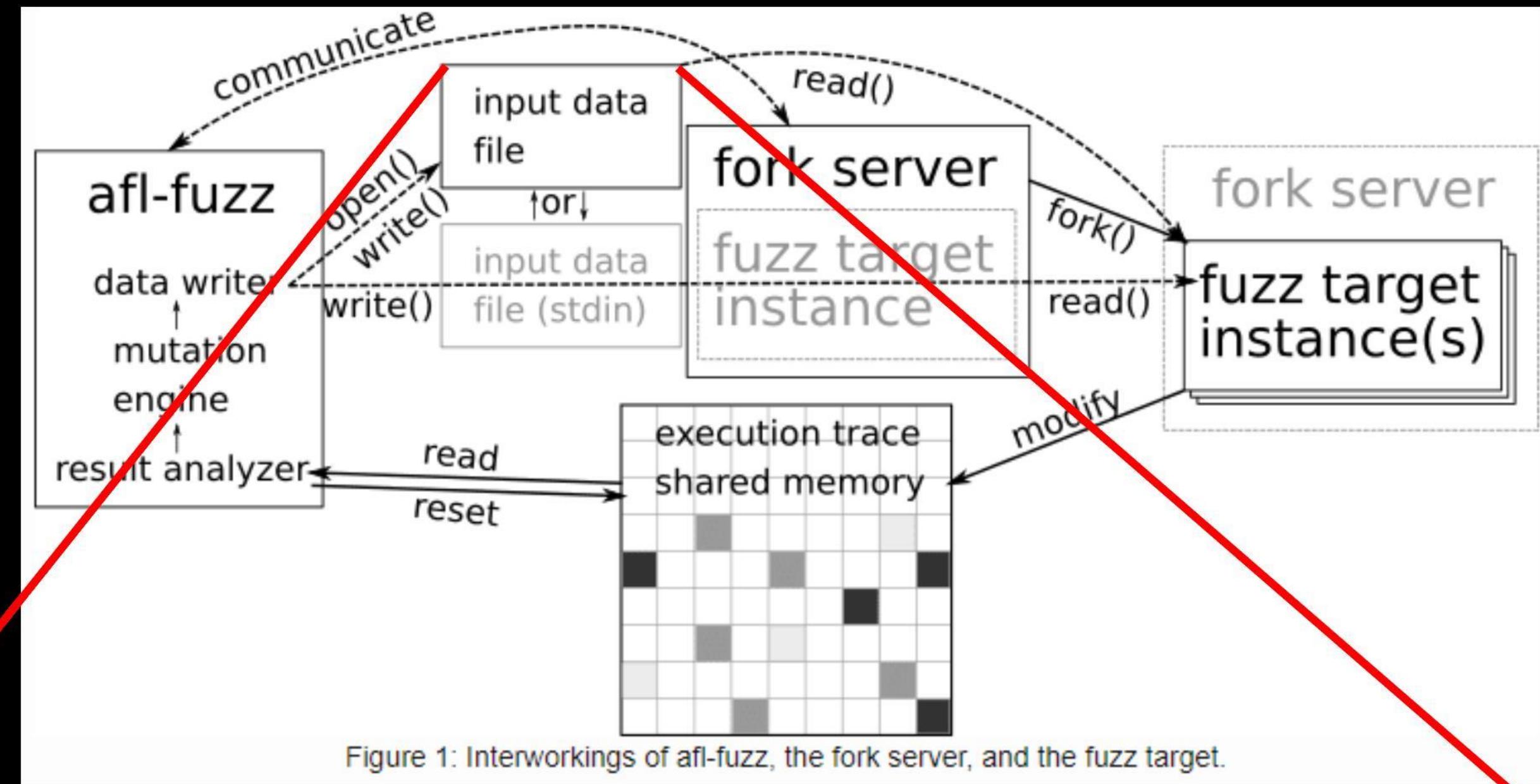
## AFL Structure



**Generated program and pipe communication**

# What is Fuzzing?

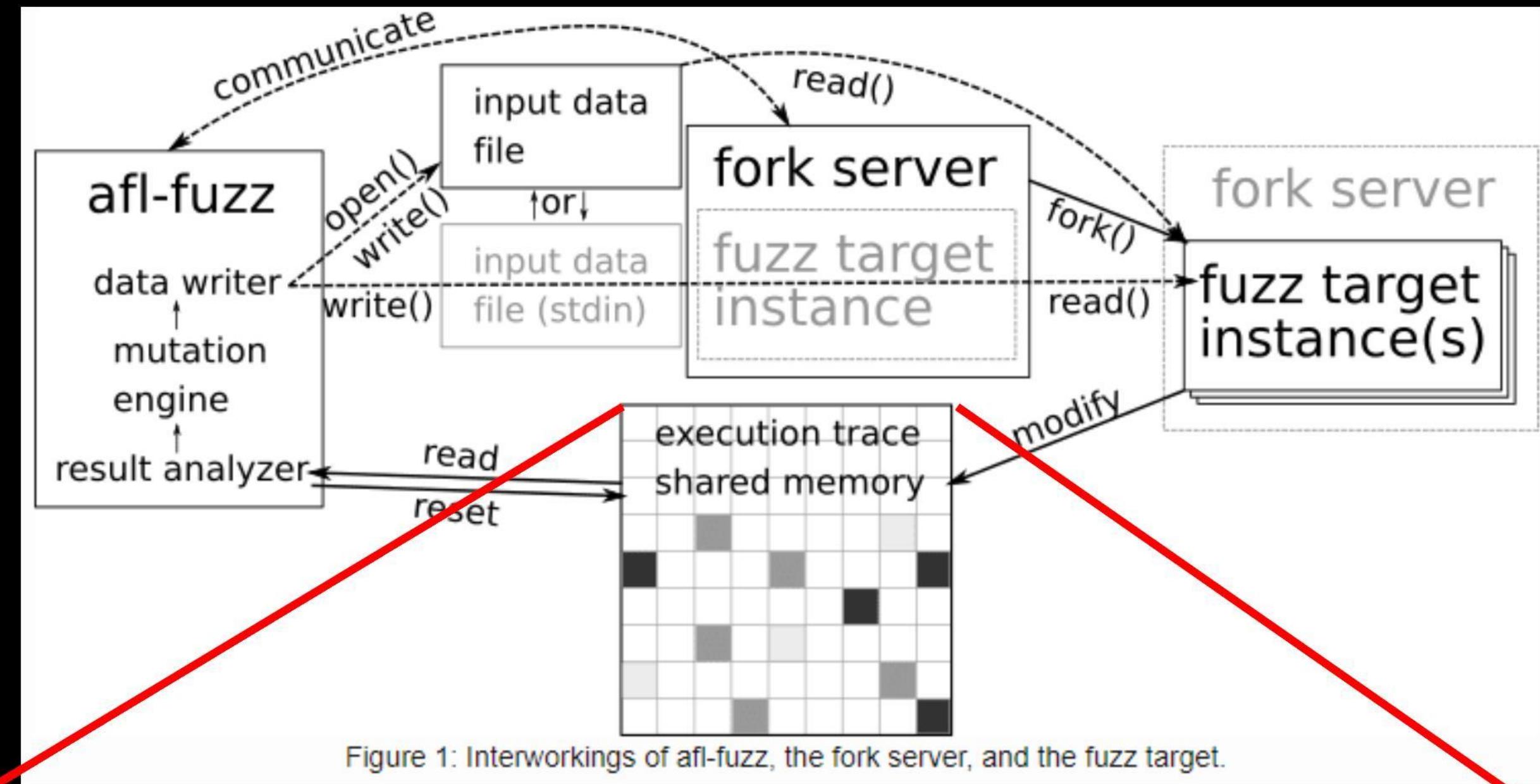
## AFL Structure



**Passes the value or file used as standard input.**

# What is Fuzzing?

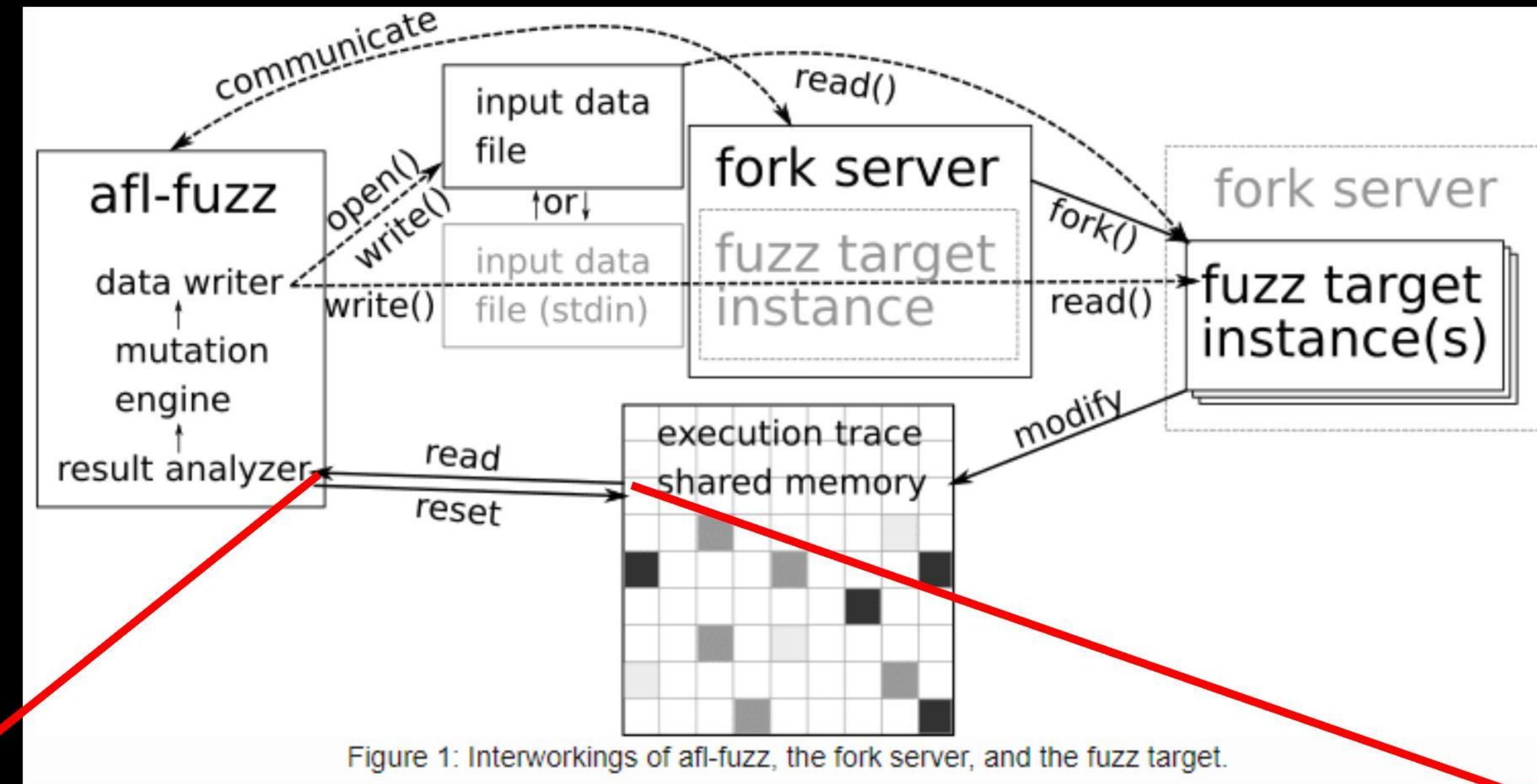
## AFL Structure



**Write execution results to shared memory**

# What is Fuzzing?

## AFL Structure



**afl-fuzz reads the shared memory, mutates it, creates a new value, and passes it on.**

**Now that's the end of the background knowledge. Start the main game**

A cartoon illustration of a green, blob-like character with a single large eye and a wide mouth, standing on a pink surface. Two white speech bubbles are positioned above its head. The bubble on the left contains the text "What is" and the bubble on the right contains the text "CVE-2021-3156".

**What is**

**CVE-2021-3156**

# **What is CVE-2021-3156**

# What is CVE-2021-3156

Himanshu Kathpal's Blog

## CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)



Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.

January 26, 2021 - 12 min read

Like 415

Last updated on: December 23, 2022

**Update Feb 3, 2021:** It has been reported that macOS, AIX, and Solaris are also vulnerable to CVE-2021-3156, and that others may also still be vulnerable. Qualys has not independently verified the exploit.

**Original Post:** The Qualys Research Team has discovered a heap overflow vulnerability in sudo, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default sudo configuration by exploiting this vulnerability.

Sudo is a powerful utility that's included in most if not all Unix- and Linux-based OSes. It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced in July 2011 (commit 8255ed69) and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration.

Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit and obtain full root privileges on Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), and Fedora 22 (Sudo 1.9.2). Other operating systems and distributions are also

# What is CVE-2021-3156

Himanshu Kathpal's Blog

## CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)



Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.  
January 26, 2021 - 12 min read

415

Last updated on: December 23, 2022

**Update Feb 3, 2021:** It has been reported that macOS, AIX, and Solaris are also vulnerable to CVE-2021-3156, and that others may also still be vulnerable. Qualys has not independently verified the exploit.

**Original Post:** The Qualys Research Team has discovered a heap overflow vulnerability in sudo, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default sudo configuration by exploiting this vulnerability.

Sudo is a powerful utility that's included in most if not all Unix- and Linux-based OSes. It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced in July 2011 (commit 8255ed69) and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration.

Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit and obtain full root privileges on Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), and Fedora 22 (Sudo 1.9.2). Other operating systems and distributions are also

Vulnerability Name

# What is CVE-2021-3156

Himanshu Kathpal's Blog

## CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)



Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.  
January 26, 2021 - 12 min read

415

Last updated on: December 23, 2022

**Update Feb 3, 2021:** It has been reported that macOS, AIX, and Solaris are also vulnerable to CVE-2021-3156, and that others may also still be vulnerable. Qualys has not independently verified the exploit.

**Original Post:** The Qualys Research Team has discovered a heap overflow vulnerability in sudo, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default sudo configuration by exploiting this vulnerability.

Sudo is a powerful utility that's included in most if not all Unix- and Linux-based OSes. It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced in July 2011 (commit 8255ed69) and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration.

Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit and obtain full root privileges on Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), and Fedora 22 (Sudo 1.9.2). Other operating systems and distributions are also

Vulnerability Name

# What is CVE-2021-3156

Himanshu Kathpal's Blog

CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)



Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.  
January 26, 2021 - 12 min read

415

Last updated on: December 23, 2022

Update Feb 3, 2021: It has been reported that AIX and Solaris are also vulnerable to CVE-2021-3156, and that others may also still be vulnerable. Qualys security researchers have independently verified the exploit.

Original Post: The Qualys Research Team has discovered a heap overflow vulnerability in sudo, a near-ubiquitous utility available on major Unix-like operating systems. Any unprivileged user can gain root privileges on a vulnerable host using a default sudo configuration by exploiting this vulnerability.

Sudo is a powerful utility that's included in most if not all Unix- and Linux-based OSes. It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced in July 2011 (commit 8255ed69) and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration.

Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit and obtain full root privileges on Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), and Fedora 32 (Sudo 1.9.2). Other operating systems and distributions are also

Vulnerability Name

This vulnerability was introduced in 2011, and the commit hash is 8255ed69.

# What is CVE-2021-3156

[sudo/plugins/sudoers/sudoers.c \(8255ed69 commit\)](#)

```
864 -         char *to, **from;
865 -         size_t size, n;
866 -
867 -         /* Alloc and build up user_args. */
868 -         for (size = 0, from = NewArgv + 1; *from; from++)
869 -             size += strlen(*from) + 1;
870 -         user_args = emalloc(size);
871 -         for (to = user_args, from = NewArgv + 1; *from; from++) {
872 -             n = strlcpy(to, *from, size - (to - user_args));
873 -             if (n >= size - (to - user_args))
874 -                 errorx(1, _("internal error, set_cmnd() overflow"));
875 -             to += n;
876 -             *to++ = ' ';
864 +         char *to, *from, **av;
865 +         size_t size, n;
866 +
867 +         /* Alloc and build up user_args. */
868 +         for (size = 0, av = NewArgv + 1; *av; av++)
869 +             size += strlen(*av) + 1;
870 +         user_args = emalloc(size);
871 +         if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
872 +             /*
873 +              * When running a command via a shell, the sudo front-end
874 +              * escapes potential meta chars. We unescape non-spaces
875 +              * for sudoers matching and logging purposes.
876 +             */
877 +             for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
878 +                 while (*from) {
879 +                     if (from[0] == '\\' && !isspace((unsigned char)from[1]))
880 +                         from++;
881 +                     *to++ = *from++;
882 +                 }
883 +                 *to++ = ' ';
884 +             }
885 +             *--to = '\0';
886 +         } else {
887 +             for (to = user_args, av = NewArgv + 1; *av; av++) {
888 +                 n = strlcpy(to, *av, size - (to - user_args));
889 +                 if (n >= size - (to - user_args))
890 +                     errorx(1, _("internal error, set_cmnd() overflow"));
891 +                 to += n;
892 +                 *to++ = ' ';
893 +             }
894 +             *--to = '\0';
```

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859         /*
860          * When running a command via a shell, the sudo front-end
861          * escapes potential meta chars. We unescape non-spaces
862          * for sudoers matching and logging purposes.
863         */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859         /*
860          * When running a command via a shell, the sudo front-end
861          * escapes potential meta chars. We unescape non-spaces
862          * for sudoers matching and logging purposes.
863         */
864         for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865             while (*from) {
866                 if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                     from++;
868                 *to++ = *from++;
869             }
870             *to++ = ' ';
871         }
872         *--to = '\0';
873     } else {
874         for (to = user_args, av = NewArgv + 1; *av; av++) {
875             n = strlcpy(to, *av, size - (to - user_args));
876             if (n >= size - (to - user_args)) {
877                 sudo_warnx(U_("internal error, %s overflow"), __func__);
878                 debug_return_int(-1);
879             }
880             to += n;
881             *to++ = ' ';
882         }
883         *--to = '\0';
884     }
```

명령줄 인자의 문자열 길이를 더한다

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

명령줄 인자의 문자열 길이를 더한다

size 만큼 heap 메모리 공간을 할당 받는다

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851  /* Alloc and build up user_args. */
852  for (size = 0, av = NewArgv + 1; *av; av++)
853  size += strlen(*av) + 1;
854  if (size == 0 || (user_args = malloc(size)) == NULL) {
855      sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856      debug_return_int(-1);
857  }
858  if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859      /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863      */
864  for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865      while (*from) {
866          if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867              from++;
868          *to++ = *from++;
869      }
870      *to++ = ' ';
871  }
872  *--to = '\0';
873 } else {
874  for (to = user_args, av = NewArgv + 1; *av; av++) {
875      n = strlcpy(to, *av, size - (to - user_args));
876      if (n >= size - (to - user_args)) {
877          sudo_warnx(U_("internal error, %s overflow"), __func__);
878          debug_return_int(-1);
879      }
880      to += n;
881      *to++ = ' ';
882  }
883  *--to = '\0';
884 }
```

명령줄 인자의 문자열 길이를 더한다

size 만큼 heap 메모리 공간을 할당 받는다

문자열을 문자 단위로 heap 메모리에 복사한다

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

명령줄 인자의 문자열 길이를 더한다

size 만큼 heap 메모리 공간을 할당 받는다

문자열을 문자 단위로 heap 메모리에 복사한다

'\' 문자가 존재하면, from 포인터를 증가시킨다

# What is CVE-2021-3156

[sudo/plugins/sudoers/s](#)

```
851     /* Alloc and build up user args */
852     for (size = 0, av = NewArgv(); *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args == 0))
855         sudo_warnx(U_( "%s: %s"), __func__, strerror(errno));
856     debug_return_int(-1);
857 }
858 if (ISSET(sudo_mode, MODE_SUDO))
859 /*
860  * When running a command via sudo, we
861  * escapes potential meta characters
862  * for sudoers matching and
863  */
864 for (to = user_args, av = NewArgv(); *av; av++)
865     while (*from) {
866         if (from[0] == '\\\' &&
867             from++,
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv(); *av; av++)
875         n = strlcpy(to, *av, size);
876     if (n >= size - (to - user_args))
877         sudo_warnx(U_( "internal error"));
878     debug_return_int(-1);
879 }
880     to += n;
881     *to++ = ' ';
882 }
883     *--to = '\0';
884 }
```



hs of Argv.

cate the heap  
length.

character by  
heap.

ring, the from  
es once more.

View in detail

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859         /*
860          * When running a command via a shell, the sudo front-end
861          * escapes potential meta chars. We unescape non-spaces
862          * for sudoers matching and logging purposes.
863         */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for [to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859         /*
860          * When running a command via a shell, the sudo front-end
861          * escapes potential meta chars. We unescape non-spaces
862          * for sudoers matching and logging purposes.
863         */
864     for [to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

문자열을 한 문자씩 동적으로 할당된 공간에 복사한다.

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859         /*
860          * When running a command via a shell, the sudo front-end
861          * escapes potential meta chars. We unescape non-spaces
862          * for sudoers matching and logging purposes.
863         */
864         for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865             while (*from) {
866                 if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                     from++;
868                 *to++ = *from++;
869             }
870             *to++ = ' ';
871         }
872         *--to = '\0';
873     } else {
874         for (to = user_args, av = NewArgv + 1; *av; av++) {
875             n = strlcpy(to, *av, size - (to - user_args));
876             if (n >= size - (to - user_args)) {
877                 sudo_warnx(U_("internal error, %s overflow"), __func__);
878                 debug_return_int(-1);
879             }
880             to += n;
881             *to++ = ' ';
882         }
883         *--to = '\0';
884     }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. we unescape non-spaces
862      * for sudoers matching and logging purposes.
863      */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

**-s \x00AAAAA\lx00z3rodaeo**

# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\\' && !isspace((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869             }
870             *to++ = ' ';
871         }
872         *--to = '\\0';
873     } else {
874         for (to = user_args, av = NewArgv + 1; *av; av++) {
875             n = strlcpy(to, *av, size - (to - user_args));
876             if (n >= size - (to - user_args)) {
877                 sudo_warnx(U_("internal error, %s overflow"), __func__);
878                 debug_return_int(-1);
879             }
880             to += n;
881             *to++ = ' ';
882         }
883         *--to = '\\0';
884     }
```

명령줄 인수에 '\\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO



from

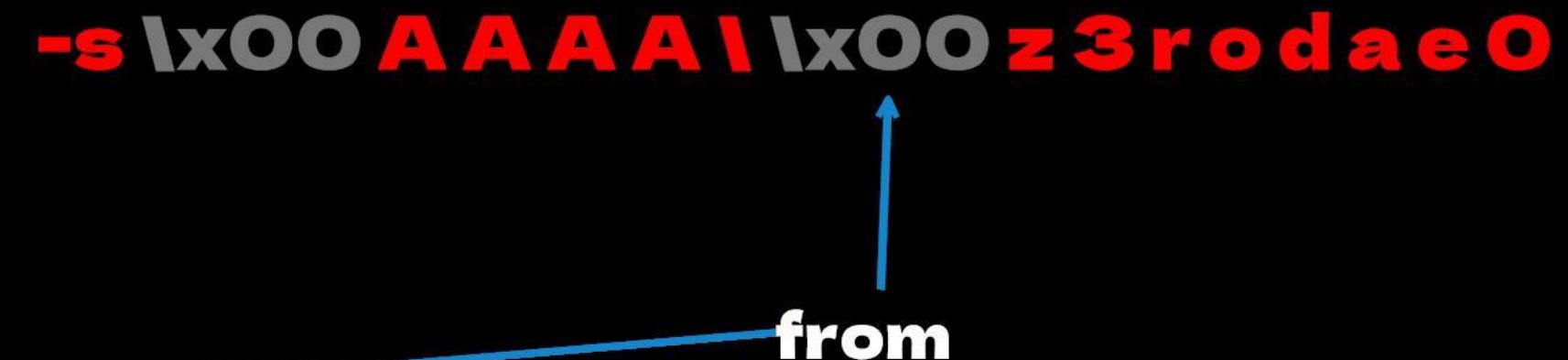
# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1])) {
867                 from++;
868                 *to++ = *from++;
869             }
870             *to++ = ' ';
871         }
872         *--to = '\0';
873     } else {
874         for (to = user_args, av = NewArgv + 1; *av; av++) {
875             n = strlcpy(to, *av, size - (to - user_args));
876             if (n >= size - (to - user_args)) {
877                 sudo_warnx(U_("internal error, %s overflow"), __func__);
878                 debug_return_int(-1);
879             }
880             to += n;
881             *to++ = ' ';
882         }
883         *--to = '\0';
884     }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO



# What is CVE-2021-3156

## [sudo/plugins/sudoers/sudoers.c](#)

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !isspace((unsigned char)from[1])) {
867                 from++;
868                 *to++ = *from++;
869             }
870             *to++ = ' ';
871         }
872         *--to = '\0';
873     } else {
874         for (to = user_args, av = NewArgv + 1; *av; av++) {
875             n = strlcpy(to, *av, size - (to - user_args));
876             if (n >= size - (to - user_args)) {
877                 sudo_warnx(U_("internal error, %s overflow"), __func__);
878                 debug_return_int(-1);
879             }
880             to += n;
881             *to++ = ' ';
882         }
883         *--to = '\0';
884     }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO

from

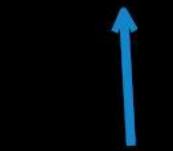
# What is CVE-2021-3156

## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !is_hex((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO



from

Write out of bounds string because it is not null

# What is CVE-2021-3156

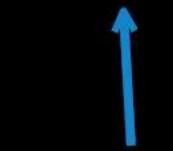
## sudo/plugins/sudoers/sudoers.c

```
851     /* Alloc and build up user_args. */
852     for (size = 0, av = NewArgv + 1; *av; av++)
853         size += strlen(*av) + 1;
854     if (size == 0 || (user_args = malloc(size)) == NULL) {
855         sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856         debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859     /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863     */
864     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865         while (*from) {
866             if (from[0] == '\\\' && !is_hex((unsigned char)from[1]))
867                 from++;
868             *to++ = *from++;
869         }
870         *to++ = ' ';
871     }
872     *--to = '\0';
873 } else {
874     for (to = user_args, av = NewArgv + 1; *av; av++) {
875         n = strlcpy(to, *av, size - (to - user_args));
876         if (n >= size - (to - user_args)) {
877             sudo_warnx(U_("internal error, %s overflow"), __func__);
878             debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884 }
```

**Write out of bounds string because it is not null**

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO



**from**

```
root:/pwd/sudo-1.8.31p2/src# ./sudoedit -s aaaaaaaaa\\
from = aaaaaaaaa\
*from = a
from = aaaaaaaaa\
*from = a
from = aaaaaaaaa\
*from = a
from = aaaaaaa\
*from = a
from = aaaa\
*from = a
from = aaa\
*from = a
from = aa\
*from = a
from = a\
*from = a
from = \
*from = \
```

# What is CVE-2021-3156

[sudo/plugins/sudoers/sudoers.c](#)

```
851  /* Alloc and build up user_args */
852  for (size = 0, av = NewArgv + 1;
853      size += strlen(*av) + 1;
854      if (size == 0 || (user_args =
855          sudo_warnx(U_( "%s: %s"), __func__,
856          debug_return_int(-1));
857      }
858      if (ISSET(sudo_mode, MODE_SHELL))
859      /*
860       * When running a command via
861       * escapes potential meta char
862       * for sudoers matching and lo
863       */
864      for (to = user_args, av = NewA
865          while (*from) {
866              if (from[0] == '\\\' && !is
867                  from++;
868                  *to++ = *from++;
869              }
870              *to++ = ' ';
871          }
872          *--to = '\0';
873      } else {
874          for (to = user_args, av = NewA
875              n = strlcpy(to, *av, size);
876              if (n >= size - (to - user_
877                  sudo_warnx(U_( "internal er
878                  debug_return_int(-1);
879              }
880              to += n;
881              *to++ = ' ';
882          }
883          *--to = '\0';
884      }
```



으면 from 포인터가 증가된

ax00z3rodadeO

↑  
from

dit -s aaaaaaaa\\

## vulnerability

```
*from = a
from = a\
*from = a
from = \
*from = \
```

# What is CVE-2021-3156

[sudo/plugins/sudoers/sudoers.c](#)

```
851  /* Alloc and build up user_args. */
852  for (size = 0, av = NewArgv + 1; *av; av++)
853      size += strlen(*av) + 1;
854  if (size == 0 || (user_args = malloc(size)) == NULL) {
855      sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
856      debug_return_int(-1);
857  }
858  if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
859      /*
860      * When running a command via a shell, the sudo front-end
861      * escapes potential meta chars. We unescape non-spaces
862      * for sudoers matching and logging purposes.
863      */
864  for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
865      while (*from) {
866          if (from[0] == '\\\' && !is_hex((unsigned char)from[1]))
867              from++;
868          *to++ = *from++;
869          }
870          *to++ = ' ';
871      }
872      *--to = '\0';
873  } else {
874      for (to = user_args, av = NewArgv + 1; *av; av++) {
875          n = strlcpy(to, *av, size - (to - user_args));
876          if (n >= size - (to - user_args)) {
877              sudo_warnx(U_("warning: %s overflow (%d)\n"), __func__, n);
878              debug_return_int(1);
879          }
880          to += n;
881          *to++ = ' ';
882      }
883      *--to = '\0';
884  }
```

취약한 코드를 실행시키려면 어떻게 해야 할까?

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

-s \x00AAAAA\ \x00z3rodaeO

from

```
root:/pwd/sudo-1.8.31p2/src# ./sudoedit -s aaaaaaaa\
from = aaaaaaaaa\
*from = a
from = aaaaaaaaa\
*from = a
from = aaaaaaaaa\
*from = a
from = aaaaaaa\
*from = a
from = aaaa\
*from = a
from = aaa\
*from = a
from = a\
*from = a
from = \
*from = \
```

# What is CVE-2021-3156

[sudo/plugins/sudoers/sudoers.c](#)

```
819     if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
820         // printf("debug... sudo_mode = %d & %d\n", sudo_mode, (MODE_RUN | MODE_EDIT | MODE_CHECK));
821         if (ISSET(sudo_mode, MODE_RUN | MODE_CHECK)) {
822
823             /* Alloc and build up user_args. */
824             for (size = 0, av = NewArgv + 1; *av; av++)
825                 size += strlen(*av) + 1;
826             if (size == 0 || (user_args = malloc(size)) == NULL) {
827                 sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
828                 debug_return_int(-1);
829             }
830             if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
831                 /*
832                  * When running a command via a shell, the sudo front-end
833                  * escapes potential meta chars. We unescape non-spaces
834                  * for sudoers matching and logging purposes.
835                 */
836                 for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
837                     while (*from) {
838                         if (from[0] == '\\\' && !isspace((unsigned char)from[1])) {
839
840                             from++;
841                             *to++ = *from++;
842                         }
843                         *to++ = ' ';
844                     }
845                     *--to = '\0';
846                 } else {
```

[sudo/src/parse\\_args.c](#)

```
case 'e':
    if (mode && mode != MODE_EDIT)
        usage_excl();
    mode = MODE_EDIT;
    sudo_settings[ARG_SUDOEDIT].value = "true";
    valid_flags = MODE_NONINTERACTIVE;
    break;

or

case 'l':
    if (mode) {
        if (mode == MODE_LIST)
            SET(flags, MODE_LONG_LIST);
        else
            usage_excl();
    }
    mode = MODE_LIST;
    valid_flags = MODE_NONINTERACTIVE|MODE_LONG_LIST;
    break;

if ((flags & valid_flags) != flags)
    usage();
```

# What is CVE-2021-3156

[sudo/plugins/sudoers/sudoers.c](#)

```
819     if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
820         // printf("debug... sudo_mode = %d & %d\n", sudo_mode, (MODE_RUN | MODE_EDIT | MODE_CHECK));
821         if (ISSET(sudo_mode, MODE_RUN | MODE_CHECK)) {
822
823             /* Alloc and build up user_args. */
824             for (size = 0, av = NewArgv + 1; *av; av++)
825                 size += strlen(*av) + 1;
826             if (size == 0 || (user_args = malloc(size)) == NULL) {
827                 sudo_warnx(U_("%s: %s"), __func__, U_("unable to allocate memory"));
828                 debug_return_int(-1);
829             }
830             if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
831
832                 * When running a command via a shell, the sudo front-end
833                 * escapes potential meta chars. We unescape non-spaces
834                 * for sudoers matching and logging purposes.
835
836                 for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
837                     while (*from) {
838                         if (from[0] == '\\\' && !isspace((unsigned char)from[1]))
839
840                             from++;
841                         *to++ = *from++;
842                     }
843                     *to++ = ' ';
844                 }
845                 *--to = '\0';
846             } else {
```

[sudo/src/parse\\_args.c](#)

```
/* First, check to see if we were invoked as "sudoedit". */
proglen = strlen(progname);
if (proglen > 4 && strcmp(progname + proglen - 4, "edit") == 0) {
    progname = "sudoedit";
    mode = MODE_EDIT;
    sudo_settings[ARG_SUDOEDIT].value = "true";
}
...
case 's':
    sudo_settings[ARG_USER_SHELL].value = "true";
    SET(flags, MODE_SHELL);
    break;
```

**sudoedit -s == sudo -e -s**

**Find vulnerabilities with  
fuzzing**

# Find vulnerabilities with fuzzing

# Find vulnerabilities with fuzzing

SUDO는 공격 표면이 ARGV이다.

명령줄 인수 ARGV 퍼징

# Find vulnerabilities with fuzzing

SUDO는 공격 표면이 ARGVOID이다.

명령줄 인수 AI



how?

# Find vulnerabilities with fuzzing

## AFL/experimental/argv\_fuzzing/argv-fuzz-inl.h

```
1  /*
2   Copyright 2015 Google LLC All rights reserved.
3
4   Licensed under the Apache License, Version 2.0 (the "License");
5   you may not use this file except in compliance with the License.
6   You may obtain a copy of the License at:
7
8     http://www.apache.org/licenses/LICENSE-2.0
9
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15 */
16
17
18  /*
19   american fuzzy lop - sample argv fuzzing wrapper
20  -----
21
22  Written by Michal Zalewski <lcamtuf@google.com>
23
24  This file shows a simple way to fuzz command-line parameters with stock
25  afl-fuzz. To use, add:
26
27  #include "/path/to/argv-fuzz-inl.h"
28
29  ...to the file containing main(), ideally placing it after all the
30  standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
31  main().
32
```

# Find vulnerabilities with fuzzing

## AFL/experimental/argv\_fuzzing/argv-fuzz-inl.h

```
1  /*
2   * Copyright 2015 Google LLC All rights reserved.
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at:
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15 */
16
17 /*
18 * american fuzzy lop - sample argv fuzzing wrapper
19 -----
20
21 Written by Michal Zalewski <lcamtuf@google.com>
22
23 This file shows a simple way to fuzz command-line parameters with stock
24 afl-fuzz. To use, add:
25
26 #include "/path/to/argv-fuzz-inl.h"
27
28 ...to the file containing main(), ideally placing it after all the
29 standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
30 main().
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
918
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
```

# Find vulnerabilities with fuzzing

## AFL/experimental/argv\_fuzzing/argv-fuzz-inl.h

```
1  /*
2   * Copyright 2015 Google LLC All rights reserved.
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at:
7   *
8   * http://www.apache.org/licenses/LICENSE-2.0
9   *
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15 */
16
17 /**
18 * american fuzzy lop - sample argv fuzzing wrapper
19 -----
20
21 Written by Michal Zalewski <lcamtuf@google.com>
22
23 This file shows a simple way to fuzz command-line parameters with stock
24 afl-fuzz. To use, add:
25
26 #include "/path/to/argv-fuzz-inl.h"
27
28 ...to the file containing main(), ideally placing it after all the
29 standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
30 main().
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
#include "argv-fuzz-inl.h"
int main(int argc, char *argv[]){
    AFL_INIT_ARGV();
    return 0;
}
```

# Find vulnerabilities with fuzzing

## AFL/experimental/argv\_fuzzing/argv-fuzz-inl.h

```
1  /*
2   * Copyright 2015 Google LLC All rights reserved.
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at:
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15 */
16
17 /**
18 * american fuzzy lop - sample argv fuzzing wrapper
19 -----
20
21 Written by Michal Zalewski <lcamtuf@google.com>
22
23 This file shows a simple way to fuzz command-line parameters with stock
24 afl-fuzz. To use, add:
25
26 #include "/path/to/argv-fuzz-inl.h"
27
28 ...to the file containing main(), ideally placing it after all the
29 standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
30 main().
31
32 
```

```
#include "argv-fuzz-inl.h"
int main(int argc, char *argv[]){
    AFL_INIT_ARGV();
    return 0;
}
```

# Find vulnerabilities with fuzzing

## AFL/experimental/argv\_fuzzing/argv-fuzz-inl.h

```
1  /*
2   * Copyright 2015 Google LLC All rights reserved.
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at:
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15 */
16
17 /**
18 * american fuzzy lop - sample argv fuzzing wrapper
19 -----
20
21 Written by Michal Zalewski <lcamtuf@google.com>
22
23 This file shows a simple way to fuzz command-line parameters with stock
24 afl-fuzz. To use, add:
25
26 #include "/path/to/argv-fuzz-inl.h"
27
28 ...to the file containing main(), ideally placing it after all the
29 standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
30 main().
31
32 
```

```
#include "argv-fuzz-inl.h"
int main(int argc, char *argv[]){
    AFL_INIT_ARGV();
    return 0;
}

#define AFL_INIT_ARGV() \
do { \
    argv = afl_init_argv(&argc); \
} while (0)
```

# Find vulnerabilities with fuzzing

## sudo/src/sudo.c

```
135 int
136 ~main(int argc, char *argv[], char *envp[])
137 {
138     int nargc, ok, status = 0;
139     char **nargv, **env_add;
140     char **user_info, **command_info, **argv_out, **user_env_out;
141     struct sudo_settings *settings;
142     struct plugin_container *plugin, *next;
143     sigset_t mask;
144     debug_decl_vars(main, SUDO_DEBUG_MAIN)
145
146     initprogname(argc > 0 ? argv[0] : "sudo");
147
148     /* Crank resource limits to unlimited. */
149     unlimit_sudo();
150
151     /* Make sure fds 0-2 are open and do OS-specific initialization. */
152     fix_fds();
153     os_init(argc, argv, envp);
154
155     setlocale(LC_ALL, "");
156     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
157     textdomain(PACKAGE_NAME);
158
159     (void) tzset();
160
161     /* Must be done before we do any password lookups */
162 ~#if defined(HAVE_GETPRPWNAME) && defined(HAVE_SET_AUTH_PARAMETERS)
163     (void) set_auth_parameters(argc, argv);
164 # ifdef HAVE_INITPRIVS
165     initprivs();
166 # endif
167 #endif /* HAVE_GETPRPWNAME && HAVE_SET_AUTH_PARAMETERS */
168
169     /* Initialize the debug subsystem. */
170     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
171         exit(EXIT_FAILURE);
172     sudo_debug_instance = sudo_debug_register(getprogname(),
173         NULL, NULL, sudo_conf_debug_files(getprogname()));
174     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
175         exit(EXIT_FAILURE);
176
177     /* Make sure we are setuid root. */
178     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
135 int
136 ~main(int argc, char *argv[], char *envp[])
137 {
138     int nargc, ok, status = 0;
139     char **nargv, **env_add;
140     char **user_info, **command_info, **argv_out, **user_env_out;
141     struct sudo_settings *settings;
142     struct plugin_container *plugin, *next;
143     sigset_t mask;
144     debug_decl_vars(main, SUDO_DEBUG_MAIN)
145
146     initprogname(argc > 0 ? argv[0] : "sudo");
147
148     /* Crank resource limits to unlimited. */
149     unlimit_sudo();
150
151     /* Make sure fds 0-2 are open and do OS-specific initialization. */
152     fix_fds();
153     os_init(argc, argv, envp);
154
155     setlocale(LC_ALL, "");
156     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
157     textdomain(PACKAGE_NAME);
158
159     (void) tzset();
160
161     /* Must be done before we do any password lookups */
162 ~#if defined(HAVE_GETPRPWNAME) && defined(HAVE_SET_AUTH_PARAMETERS)
163     (void) set_auth_parameters(argc, argv);
164 # ifdef HAVE_INITPRIVS
165     initprivs();
166 # endif
167 #endif /* HAVE_GETPRPWNAME && HAVE_SET_AUTH_PARAMETERS */
168
169     /* Initialize the debug subsystem. */
170     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
171         exit(EXIT_FAILURE);
172     sudo_debug_instance = sudo_debug_register(getprogname(),
173         NULL, NULL, sudo_conf_debug_files(getprogname()));
174     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
175         exit(EXIT_FAILURE);
176
177     /* Make sure we are setuid root. */
178     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

#include "argv-fuzz-inl.h"

AFL\_INIT\_ARGV();

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
135 int main(int argc, char *argv[], char *envp[])
136 {
137     int nargc, ok, status = 0;
138     char **nargv, **env_add;
139     char **user_info, **command_info, **argv_out, **user_env_out;
140     struct sudo_settings *settings;
141     struct plugin_container *plugin, *next;
142     sigset_t mask;
143     debug_decl_vars(main, SUDO_DEBUG_MAIN);
144
145     initprogname(argc > 0 ? argv[0] : "sudo");
146
147     /* Crank resource limits to unlimited. */
148     unlimit_sudo();
149
150     /* Make sure fds 0-2 are open and do OS-specific initialization. */
151     fix_fds();
152     os_init(argc, argv, envp);
153
154     setlocale(LC_ALL, "");
155     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
156     textdomain(PACKAGE_NAME);
157
158     (void) tzset();
159
160     /* Must be done before we do any password lookups */
161 #if defined(HAVE_GETPRPWNAME) && defined(HAVE_SET_AUTH_PARAMETERS)
162     (void) set_auth_parameters(argc, argv);
163 # ifdef HAVE_INITPRIVS
164     initprivs();
165 # endif
166 #endif /* HAVE_GETPRPWNAME && HAVE_SET_AUTH_PARAMETERS */
167
168     /* Initialize the debug subsystem. */
169     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
170         exit(EXIT_FAILURE);
171     sudo_debug_instance = sudo_debug_register(getprogname(),
172         NULL, NULL, sudo_conf_debug_files(getprogname()));
173     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
174         exit(EXIT_FAILURE);
175
176     /* Make sure we are setuid root. */
177     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

#include "argv-fuzz-inl.h"

AFL\_INIT\_ARGV();

# Find vulnerabilities with fuzzing

**sudo/src/****sudo.c**

```
155 int main(int argc, char *argv[], char *envp[])
156 {
157     int nargc, nenv_status = 0;
158     char **nargv, **envp_end;
159     char **user_info, **command_info, **argv_out, **user_env_end;
160     struct sudo_settings *settings;
161     struct plugin_container *plugin, *next;
162     sigset_t mask;
163     debug_decl_vars(main, SUDO_DEBUG_MAIN);
164
165     initprogname(argc > 0 ? argv[0] : "sudo");
166
167     /* Crank resource limits to unlimited. */
168     unlimit_sudo();
169
170     /* Make sure fds 0-2 are open and do OS-specific initialization. */
171     fix_fds();
172     os_init(argc, argv, envp);
173
174     setlocale(LC_ALL, "");
175     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
176     textdomain(PACKAGE_NAME);
177
178     (void) tzset();
179
180     /* Must be done before we do any password lookups */
181 #if defined(HAVE_GETPRPWNAME) && defined(HAVE_SET_AUTH_PARAMETERS)
182     (void) set_auth_parameters(argc, argv);
183 # ifdef HAVE_INITPRIVS
184     initprivs();
185 # endif
186 #endif /* HAVE_GETPRPWNAME && HAVE_SET_AUTH_PARAMETERS */
187
188     /* Initialize the debug subsystem. */
189     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
190         exit(EXIT_FAILURE);
191     sudo_debug_instance = sudo_debug_register(getprogname(),
192                                              NULL, NULL,
193                                              sudo_conf_debug_files(getprogname()));
194     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
195         exit(EXIT_FAILURE);
196
197     /* Make sure we are setuid root. */
198     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

#include "argv-fuzz-inl.h"

AFL\_INIT\_ARGV();

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
135 int main(int argc, char *argv[], char *envp[])
136 {
137     int nargc, ok, status = 0;
138     char **nargv, **env_add;
139     char **user_info, **command_info, **argv_out, **user_env_out;
140     struct sudo_settings *settings;
141     struct plugin_container *plugin, *next;
142     sigset_t mask;
143     debug_decl_vars(main, SUDO_DEBUG_MAIN)
144
145     initprogname(argc > 0 ? argv[0] : "sudo");
146
147     /* Crank resource limits to unlimited. */
148     unlimit_sudo();
149
150     /* Make sure fds 0-2 are open and do OS-specific initialization. */
151     fix_fds();
152     os_init(argc, argv, envp);
153
154     setlocale(LC_ALL, "");
155     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
156     textdomain(PACKAGE_NAME);
157
158     (void) tzset();
159
160     /* Must be done before we do any password lookups */
161 #if defined(HAVE_GETPRPWNAME) && defined(HAVE_SET_AUTH_PARAMETERS)
162     (void) set_auth_parameters(argc, argv);
163 # ifdef HAVE_INITPRIVS
164     initprivs();
165 # endif
166 #endif /* HAVE_GETPRPWNAME && HAVE_SET_AUTH_PARAMETERS */
167
168     /* Initialize the debug subsystem. */
169     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
170         exit(EXIT_FAILURE);
171     sudo_debug_instance = sudo_debug_register(getprogname(),
172         NULL, NULL, sudo_conf_debug_files(getprogname()));
173     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
174         exit(EXIT_FAILURE);
175
176     /* Make sure we are setuid root. */
177     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

#include "argv-fuzz-inl.h"

AFL\_INIT\_ARGV();

```
134 #include "argv-fuzz-inl.h"
135
136 int
137 main(int argc, char *argv[], char *envp[])
138 {
139     ARGV_INIT_ARGV();
140     int nargc, ok, status = 0;
141     char **nargv, **env_add;
142     char **user_info, **command_info, **argv_out, **user_env_out;
143     struct sudo_settings *settings;
144     struct plugin_container *plugin, *next;
145     sigset_t mask;
146     debug_decl_vars(main, SUDO_DEBUG_MAIN)
```

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
135 int main(int argc, char *argv[], char *envp[])
136 {
137     int nargc, nstatus = 0;
138     char **nargv, **envp_addr;
139     char **user_info, **command_info, **argv_out;
140     struct sudo_settings *settings;
141     struct plugin_container *plugin, *next;
142     sigset_t mask;
143     debug_decl_vars(main, SUDO_DEBUG_MAIN)
144
145     initprogname(argc > 0 ? argv[0] : "sudo");
146
147     /* Crank resource limits to unlimited. */
148     unlimit_sudo();
149
150     /* Make sure fds 0-2 are open and do OS-specific stuff. */
151     fix_fds();
152     os_init(argc, argv, envp);
153
154     setlocale(LC_ALL, "");
155     bindtextdomain(PACKAGE_NAME, LOCALEDIR);
156     textdomain(PACKAGE_NAME);
157
158     (void) tzset();
159
160     /* Must be done before we do any password lookup. */
161 #if defined(HAVE_GETPRPNAM) && defined(HAVE_SETPRPNAM)
162     (void) set_auth_parameters(argc, argv);
163 #endif
164 #ifdef HAVE_INITPRIVS
165     initprivs();
166 #endif
167 #endif /* HAVE_GETPRPNAM && HAVE_SET_AUTH_PARAMS */
168
169     /* Initialize the debug subsystem. */
170     if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) != -1)
171         exit(EXIT_FAILURE);
172     sudo_debug_instance = sudo_debug_register(getprogname(),
173 NULL, NULL, sudo_conf_debug_files(getprogname()));
174     if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
175         exit(EXIT_FAILURE);
176
177     /* Make sure we are setuid root. */
178     sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```



-fuzz-inl.h"

/();

\*envp[])

info, \*\*argv\_out, \*\*user\_env\_out;

);

.ln, \*next;

SUG\_MAIN)

**This allows argv fuzzing.**

# Find vulnerabilities with fuzzing

## **sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {
58
59     static char  in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int   rc   = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67 ✓ while (*ptr) {
68
69     ret[rc] = ptr;
70
71     /* insert '\0' at the end of ret[rc] on first space-sym */
72     while (*ptr && !isspace(*ptr)) ptr++;
73     *ptr = '\0';
74     ptr++;
75
76     /* skip more space-syms */
77     while (*ptr && isspace(*ptr)) ptr++;
78
79     rc++;
80 }
81
82 *argc = rc;
83
84 return ret;
85
86 }
```

# Find vulnerabilities with fuzzing

## **sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {
58
59     static char  in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int   rc   = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67 ✓ while (*ptr) {
68
69     ret[rc] = ptr;
70
71     /* insert '\0' at the end of ret[rc] on first space-sym */
72     while (*ptr && !isspace(*ptr)) ptr++;
73     *ptr = '\0';
74     ptr++;
75
76     /* skip more space-syms */
77     while (*ptr && isspace(*ptr)) ptr++;
78
79     rc++;
80 }
81
82 *argc = rc;
83
84 return ret;
85
86 }
```

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {  
58     static char  in_buf[MAX_CMDLINE_LEN];  
59     static char* ret[MAX_CMDLINE_PAR];  
60  
61     char* ptr = in_buf;  
62     int    rc   = 1; /* start after argv[0] */  
63  
64     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);  
65  
66     while (*ptr) {  
67         ret[rc] = ptr;  
68  
69         /* insert '\0' at the end of ret[rc] on first space-sym */  
70         while (*ptr && !isspace(*ptr)) ptr++;  
71         *ptr = '\0';  
72         ptr++;  
73  
74         /* skip more space-syms */  
75         while (*ptr && isspace(*ptr)) ptr++;  
76  
77         rc++;  
78     }  
79  
80     *argc = rc;  
81  
82     return ret;  
83  
84 }
```

int rc = 0;

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {
58
59     static char  in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int   rc   = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && !isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

**int rc = 0;**

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {  
58     static char  in_buf[MAX_CMDLINE_LEN];  
59     static char* ret[MAX_CMDLINE_PAR];  
60  
61     char* ptr = in_buf;  
62     int    rc   = 1; /* start after argv[0] */  
63  
64     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);  
65  
66     while (*ptr) {  
67  
68         ret[rc] = ptr;  
69  
70         /* insert '\0' at the end of ret[rc] on first space-sym */  
71         while (*ptr && !isspace(*ptr)) ptr++;  
72         *ptr = '\0';  
73         ptr++;  
74  
75         /* skip more space-syms */  
76         while (*ptr && isspace(*ptr)) ptr++;  
77  
78         rc++;  
79     }  
80  
81     *argc = rc;  
82  
83     return ret;  
84  
85  
86 }
```

int rc = 0;

62	char* ptr = in_buf;
63	int rc = 0; /* include argv[0] */

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
57 ✓ static char** afl_init_argv(int* argc) {  
58     static char  in_buf[MAX_CMDLINE_LEN];  
59     static char* ret[MAX_CMDLINE_PAR];  
60  
61     char* ptr = in_buf;  
62     int   rc   = 1; /* start after argv[0] */  
63  
64     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);  
65  
66     while (*ptr) {  
67         ret[rc] = ptr;  
68         /* insert '\0' at end of string */  
69         while (*ptr && !isspace(*ptr)) ptr++;  
70         *ptr = '\0';  
71         ptr++;  
72  
73         /* skip more space-syms */  
74         while (*ptr && isspace(*ptr)) ptr++;  
75  
76         rc++;  
77     }  
78  
79     *argc = rc;  
80  
81     return ret;  
82  
83  
84 }  
85  
86 }
```

int rc = 0;

**This way, even the program name can be fuzzed.**

```
62  
char  ptr = in_buf;  
int   rc   = 0; /* include argv[0] */
```

# Find vulnerabilities with fuzzing

## **sudo/lib/util/progname.c**

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85         progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

**sudo/lib/util/progname.c**

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (_progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76     if ((progname = strrchr(name, '/')) != NULL) //{
77         progname++;
78     } else {
79         progname = name;
80     }
81
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85         progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

**Delete unnecessary code to always get the program name from argv[0].**

# Find vulnerabilities with fuzzing

## sudo/lib/util/progname.c

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

## sudo/lib/util/progname.c

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

## sudo/lib/util/progname.c

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

## sudo/lib/util/progname.c

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

**sudo/lib/util/progname.c**

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

**sudo/lib/util/progname.c**

```
64 static const char *progname = "";
65
66 void
67 initprogname(const char *name)
68 {
69 # ifdef HAVE__PROGNAME
70     extern const char *__progname;
71
72     if (__progname != NULL && *__progname != '\0')
73         progname = __progname;
74     else
75 # endif
76
77     if ((progname = strrchr(name, '/')) != NULL) //{
78         progname++;
79     } else {
80         progname = name;
81     }
82
83     /* Check for libtool prefix and strip it if present. */
84     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
85     progname[3] != '\0')
86         progname += 3;
87 }
88
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

```
39 static const char *progname = "";
40
41 void
42 initprogname(const char *name)
43 {
44     if ((progname = strrchr(name, '/')) != NULL) //{
45         progname++;
46     } else {
47         progname = name;
48     }
49
50     /* Check for libtool prefix and strip it if present. */
51     if (progname[0] == 'l' && progname[1] == 't' && progname[2] == '-' &&
52     progname[3] != '\0')
53         progname += 3;
54 }
55
56 const char *
57 sudo_getprogname(void)
58 {
59     return progname;
60 }
```

# Find vulnerabilities with fuzzing

`sudo/lib/util/progname.c`



**Successfully deleted unnecessary code**

```
89 const char *
90 sudo_getprogname(void)
91 {
92     return progname;
93 }
94 #endif /* !HAVE_GETPROGNAME */
```

# Find vulnerabilities with fuzzing

CPU 리소스를 해결하기 위해 exec 계열의 함수 호출 코드를 제거한다.

# Find vulnerabilities with fuzzing

## CPU 리소스를 해결하기 위해 exec 계열의 함수 호출 코드를 제거한다.

File Name	Code Line Num
visudo.c	234, 235, 871
aix_auth.c	212
tgetpass.c	347
check_noexec.c	95, 194
logging.c	764, 766

# Find vulnerabilities with fuzzing

사용자 권한으로 접근할 수 있는 sudo의 취약점을 찾아야 한다.

# Find vulnerabilities with fuzzing

사용자 권한으로 접근할 수 있는 sudo의 취약점을 찾아야 한다.

- 사용자 권한이 있는 계정을 사용한 퍼징 또는 실행

# Find vulnerabilities with fuzzing

사용자 권한으로 접근할 수 있는 sudo의 취약점을 찾아야 한다.

- 사용자 권한이 있는 계정을 사용한 퍼징 또는 실행
- 루트 권한이 있는 계정을 사용한 퍼징 또는 실행

# Find vulnerabilities with fuzzing

## **sudo/src/****sudo.c**

```
509     /* XXX - bound check number of entries */
510     user_info = reallocarray(NULL, 32, sizeof(char *));
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY, O_RDWR);
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```

# Find vulnerabilities with fuzzing

**sudo/src/****sudo.c**

```
509     /* XXX - bound c
510     user_info = real
511     if (user_info ==
512         goto oom;
513
514     ud->pid = getpid()
515     ud->ppid = getpp
516     ud->pgid = getpg
517     ud->tcpgid = -1;
518     fd = open(_PATH_
519     if (fd != -1) {
520         ud->tcpgid = tcg
521         close(fd);
522     }
523     ud->sid = getsid
524
525     ud->uid = getuid
526     ud->euid = geteu
527     ud->gid = getgid
528     ud->egid = getegid();
```



## Hard Coding

# Find vulnerabilities with fuzzing

## sudo/src/sudo.c

```
509     /* XXX - bound check number of entries */
510     user_info = reallocarray(NULL, 32, sizeof(char *));
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY, O_RDWR);
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```

```
uid=1000(z3rodae0) gid=1000(z3rodae0)
```

# Find vulnerabilities with fuzzing

## sudo/src/sudo.c

```
509     /* XXX - bound check number of entries */
510     user_info = reallocarray(NULL, 32, sizeof(char *));
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY, O_RDWR);
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```

```
uid=1000(z3rodae0) gid=1000(z3rodae0)
```

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
509     /* XXX - bound check number of entries */
510     user_info = reallocarray(NULL, 32, sizeof(char *));
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY, O_RDWR);
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```

uid=1000(z3rodae0) gid=1000(z3rodae0)

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
509     /* XXX - bound check number of entries */
510     user_info = reallocarray(NULL, 32, sizeof(char *));
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY, O_RDWR);
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid(); //getuid();
526     ud->euid = geteuid(); //geteuid();
527     ud->gid = getgid(); //getgid();
528     ud->egid = getegid(); //getegid();
```

uid=1000(z3rodae0) gid=1000(z3rodae0)

```
525     ud->uid = 1000; //getuid();
526     ud->euid = geteuid(); //geteuid();
527     ud->gid = 1000; //getgid();
528     ud->egid = getegid(); //getegid();
```

# Find vulnerabilities with fuzzing

**sudo/src/sudo.c**

```
509     /* XXX - bound check */
510     user_info = realloc(user_info, size);
511     if (user_info == NULL)
512         goto oom;
513
514     ud->pid = getpid();
515     ud->ppid = getppid();
516     ud->pgid = getpgid(0);
517     ud->tcpgid = -1;
518     fd = open(_PATH_TTY,
519     if (fd != -1) {
520         ud->tcpgid = tcgetpgrp(fd);
521         close(fd);
522     }
523     ud->sid = getsid(0);
524
525     ud->uid = getuid();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```



```
ae0) gid=1000(z3rodae0)
uid = 1000; //getuid();
euid = geteuid();
gid = 1000; //getgid();
egid = getegid();
```

**Fake User**

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && !isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && !isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINELEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(in
58
59     static char in_buf[MAX_CMDL
60     static char* ret[MAX_CMDLINE
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after
64
65     if (read(0, in_buf, MAX_CMDL
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end
72         while (*ptr && !isspace(*p
73             *ptr = '\0';
74             ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*pt
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```



**Out of Boundary**

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

## Bounds Checking Code

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && !isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0);
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && !isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

## Bounds Checking Code

```
if(rc >= MAX_CMDLINE_PAR) {
    break;
}
```

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0),
66
67         while (*ptr) {
68
69             ret[rc] = ptr;
70
71             /* insert '\0' at the end of ret[rc] on first space-sym */
72             while (*ptr && !isspace(*ptr)) ptr++;
73             *ptr = '\0';
74             ptr++;
75
76             /* skip more space-syms */
77             while (*ptr && isspace(*ptr)) ptr++;
78
79             rc++;
80         }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

## Bounds Checking Code

**if( $rc \geq MAX\_CMDLINE\_PAR$ ) {  
break;  
}**

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0),
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

## Bounds Checking Code

**if( $rc \geq MAX\_CMDLINE\_PAR$ ) {  
break;  
}**

# Find vulnerabilities with fuzzing

**sudo/src/argv-fuzz-inl.h**

```
54 #define MAX_CMDLINE_LEN 100000
55 #define MAX_CMDLINE_PAR 1000
56
57 static char** afl_init_argv(int* argc) {
58
59     static char in_buf[MAX_CMDLINE_LEN];
60     static char* ret[MAX_CMDLINE_PAR];
61
62     char* ptr = in_buf;
63     int rc = 1; /* start after argv[0] */
64
65     if (read(0, in_buf, MAX_CMDLINE_LEN - 2) < 0),
66
67     while (*ptr) {
68
69         ret[rc] = ptr;
70
71         /* insert '\0' at the end of ret[rc] on first space-sym */
72         while (*ptr && isspace(*ptr)) ptr++;
73         *ptr = '\0';
74         ptr++;
75
76         /* skip more space-syms */
77         while (*ptr && isspace(*ptr)) ptr++;
78
79         rc++;
80     }
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

## Bounds Checking Code

```
if(rc >= MAX_CMDLINE_PAR) {
    break;
}
```

```
71 while (*ptr) {
72
73     if(rc >= MAX_CMDLINE_PAR) {
74         break;
75     }
76
77     ret[rc] = ptr;
```

# Find vulnerabilities with fuzzing

sudo

```
54     #de
55     #de
56
57     stat
58
59     st
60     st
61     ch
62     in
63
64     i
65     t
66     w
67     wh
68
69
70     -
71
72
73
74
75
76     re
77
78     *rc
81
82     *argc = rc;
83
84     return ret;
85
86 }
```

Code

AR) {

) {



All preparations are complete. Start fuzzing

# Find vulnerabilities with fuzzing

A screenshot of a terminal window with a dark background. The title bar shows two tabs: "root@4bf514dd8211: /pwd/t" and "z3rodae0@DESKTOP-F8QRQJL". The main pane displays a root shell on a system running sudo version 1.8.31p2. The command entered is "root:/pwd/sudo-1.8.31p2#". The cursor is visible at the bottom left of the terminal window.

**few hours later...**

# Find vulnerabilities with fuzzing



# Find vulnerabilities with fuzzing

```
american fuzzy lop ++4.09c {default} (alf_sudo) [fast]
process timing
  run time : 0 days, 15 hrs, 16 min, 1 sec
  last new find : 0 days, 3 hrs, 51 min, 47 sec
  last saved crash : 0 days, 2 hrs, 50 min, 38 sec
  last saved hang : none seen yet
cycle progress
  now processing : 214.1108 (33.1%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : splice 8
  stage execs : 7/14 (50.00%)
  total execs : 46.8M
  exec speed : 725.5/sec
fuzzing strategy yields
  bit flips : disabled (default, enable with -D)
  byte flips : disabled (default, enable with -D)
  arithmetics : disabled (default, enable with -D)
  known ints : disabled (default, enable with -D)
  dictionary : n/a
  havoc/splice : 423/15.7M, 227/31.0M
  py/custom/rq : unused, unused, unused, unused
    trim/eff : 6.65%/135k, disabled
strategy: exploit state: in progress
overall results
  cycles done : 141
  corpus count : 647
  saved crashes : 6
  saved hangs : 0
map coverage
  map density : 6.98% / 18.70%
  count coverage : 1.87 bits/tuple
findings in depth
  favored items : 105 (16.23%)
  new edges on : 161 (24.88%)
  total crashes : 29 (6 saved)
  total tmouts : 23 (0 saved)
item geometry
  levels : 10
  pending : 0
  pend fav : 0
  own finds : 644
  imported : 0
  stability : 100.00%
[cpu000: 33%]
^C
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit.-hhhhh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 hh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s^.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhhhhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s^.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhhhhhhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s^.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhhhhhhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s|-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s^.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhhhhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhhhhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhnnnnhhhhhh
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-Sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhnnnnnnnnn
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Look at the Crash Testcase

## Crash file

```
root:/pwd# xxd crash
00000000: 7375 646f 6564 6974 002d 6868 6873 6868 sudoedit -hhhshh
00000010: 6868 2d42 002d 7300 0100 2d41 0011 6200 nh-B.-s...-A..b.
00000020: 2d73 002d 7375 646f 6564 6974 0068 6868 -s.-Sudoedit.hhh
00000030: 6868 6868 6868 6868 2d42 002d 7300 0100 hhhhhhhh-B.-s...
00000040: 2d41 0011 6200 2d73 002d 735e 002d 6868 -A..b.-s.-s.-hh
00000050: 6868 6868 6868 6868 6868 6868 6868 6868 hhhhhhhnnnnnnnnn
00000060: 6868 7c2d 425c 002d 5500 2d75 002d 6400 hh|-B\.-U.-u.-d.
00000070: 2d76 002d 002d 5300 2d73 002d 5400 2d55 -v.-.-S.-s.-T.-U
00000080: 002d 7500 2d6e 6e6e 6e6e 6e6e 6a6e 6e6e .-u.-nnnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200 nnnn.-C.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e -B-A.-nnnnnnnnnnn
000000b0: 6e6e 002d 6500 6200 2d42 002d 002d 5300 nn.-e.b.-B.-.-S.
000000c0: 2d73 646f b2b2 b2b2 b2b2 b2c3 7564 6f65 -sdo.....udoe
000000d0: 6469 7400 2d73 0064 5400 2d55 2d56 402d dit.-s.dT.-U-V@-
000000e0: 7600 2d73 002d 5454 002d 5500 2d75 002d v.-s.-TT.-U.-u.-
000000f0: 5600 2d76 e32d 002d 4200 6215 2d42 002d V.-v.-.-B.b.-B.-
00000100: 4300 2d45 002d 2d2d 2d2d 2d2d 2d2d 2d2d C.-E.-----
00000110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d73 -----s
00000120: 5e00 2d48 002d 5300 2d50 5050 002d 0000 ^.-H.-S.-PPP.-..
00000130: 03e8 2d00 0100 2d41 0011 6200 2d73 002d ..-.-A..b.-s.-
00000140: 7375 646f 6564 6974 001b 7300 6454 002d sudoedit..s.dT.-
00000150: 552d 562d 2d2d 2d2d 2d53 002d 7364 6fb2 U-V-----S.-sdo.
00000160: b2b2 b2b2 b2b2 b2b2 b2 .....
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
Block length = 32, remaining size = 361
Block length = 16, remaining size = 96
Block length = 8, remaining size = 80
Block length = 4, remaining size = 80
Block length = 2, remaining size = 72
Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
Block length = 4, remaining size = 69
Block length = 2, remaining size = 69
Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

File size reduced by : 80.89% (to 69 bytes)
Characters simplified : 504.35%
Number of execs done : 222
    Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
Block length = 32, remaining size = 361
Block length = 16, remaining size = 96
Block length = 8, remaining size = 80
Block length = 4, remaining size = 80
Block length = 2, remaining size = 72
Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
Block length = 4, remaining size = 69
Block length = 2, remaining size = 69
Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

File size reduced by : 80.89% (to 69 bytes)
Characters simplified : 504.35%
Number of execs done : 222
Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

```
z3rodae0@DESKTOP-F8QRQJU:~$ xxd out_crash
00000000: 3065 6469 7400 2d68 3000 2d73 0030 3030 0edit.-h0.-s.000
00000010: 3030 3030 3030 3030 5c00 3030 3030 3030 00000000\.000000
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
Block length = 32, remaining size = 361
Block length = 16, remaining size = 96
Block length = 8, remaining size = 80
Block length = 4, remaining size = 80
Block length = 2, remaining size = 72
Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
Block length = 4, remaining size = 69
Block length = 2, remaining size = 69
Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

    File size reduced by : 80.89% (to 69 bytes)
    Characters simplified : 504.35%
    Number of execs done : 222
        Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

```
z3rodae0@DESKTOP-F8QRQJU:~$ xxd out_crash
00000000: 3065 6469 7400 2d68 3000 2d73 0030 3030 0edit.-h0.-s.000
00000010: 3030 3030 3030 3030 5c00 3030 3030 3030 00000000\..000000
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
```

## **sudo/src/parse\_args.c**

```
/* First, check to see if we were invoked as "sudoedit". */
proglen = strlen(progname);
if (proglen > 4 && strcmp(progname + proglen - 4, "edit") == 0) {
    progname = "sudoedit";
    mode = MODE_EDIT;
    sudo_settings[ARG_SUDOEDIT].value = "true";
}
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
  Block length = 32, remaining size = 361
  Block length = 16, remaining size = 96
  Block length = 8, remaining size = 80
  Block length = 4, remaining size = 80
  Block length = 2, remaining size = 72
  Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
  Block length = 4, remaining size = 69
  Block length = 2, remaining size = 69
  Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

  File size reduced by : 80.89% (to 69 bytes)
  Characters simplified : 504.35%
  Number of execs done : 222
    Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

```
z3rodae0@DESKTOP-F8QRQJU:~$ xxd out_crash
00000000: 3065 6469 7400 2d68 3000 2d73 0030 3030 0edit.-h0.-s.000
00000010: 3030 3030 3030 3030 5c00 3030 3030 3030 00000000\..000000
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
```

## **sudo/src/parse\_args.c**

```
/* First, check to see if we were invoked as "sudoedit". */
proglen = strlen(progname);
if (proglen > 4 && strcmp(progname + proglen - 4, "edit") == 0) {
  progname = "sudoedit";
  mode = MODE_EDIT;
  sudo_settings[ARG_SUDOEDIT].value = "true";
}
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
  Block length = 32, remaining size = 361
  Block length = 16, remaining size = 96
  Block length = 8, remaining size = 80
  Block length = 4, remaining size = 80
  Block length = 2, remaining size = 72
  Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
  Block length = 4, remaining size = 69
  Block length = 2, remaining size = 69
  Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

  File size reduced by : 80.89% (to 69 bytes)
  Characters simplified : 504.35%
  Number of execs done : 222
    Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

```
z3rodae0@DESKTOP-F8QRQJU:~$ xxd out_crash
00000000: 3065 6469 7400 2d68 3000 2d73 0030 3030 0edit.-h0.-s.000
00000010: 3030 3030 3030 3030 5c00 3030 3030 3030 00000000\..000000
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
```

## **sudo/src/parse\_args.c**

```
/* First, check to see if we were invoked as "sudoedit". */
proglen = strlen(progname);
if (proglen > 4) && strcmp(progname + proglen - 4, "edit") == 0 {
  progname = "sudoedit";
  mode = MODE_EDIT;
  sudo_settings[ARG_SUDOEDIT].value = "true";
}
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 345 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
  Block length = 32, remaining size = 361
  Block length = 16, remaining size = 96
  Block length = 8, remaining size = 80
  Block length = 4, remaining size = 80
  Block length = 2, remaining size = 72
  Block length = 1, remaining size = 70
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 code points)...
[+] Symbol minimization finished, 1 symbol (1 byte) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
  Block length = 4, remaining size = 69
  Block length = 2, remaining size = 69
  Block length = 1, remaining size = 69
[+] Block removal complete, 0 bytes deleted.

  File size reduced by : 80.89% (to 69 bytes)
  Characters simplified : 504.35%
  Number of execs done : 222
    Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```

```
z3rodae0@DESKTOP-F8QRQJU:~$ xxd out_crash
00000000: 3065 6469 7400 2d68 3000 2d73 0030 3030 0edit.-h0.-s.000
00000010: 3030 3030 3030 3030 5c00 3030 3030 3030 00000000\..000000
00000020: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
00000030: 3030 3030 3030 3030 3030 3030 3030 3030 0000000000000000
```

## **sudo/src/parse\_args.c**

```
/* First, check to see if we were invoked as "sudoedit". */
proglen = strlen(progname);
if (proglen > 4 && strcmp(progname + proglen - 4, "edit") == 0) {
  progname = "sudoedit";
  mode = MODE_EDIT;
  sudo_settings[ARG_SUDOEDIT].value = "true";
}
```

# Minimizing AFL Testcase

**afl-tmin -i crash -o /tmp/out\_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo**

```
root:/pwd# afl-tmin -i crash -o /tmp/out_crash/2 -- /pwd/sudo-1.8.31p2/src/sudo
afl-tmin++4.09c by Michal Zalewski

[+] Read 361 bytes from 'crash'.
[*] Spinning up the fork server...
[+] All right - fork server is up.
[*] Target map size: 8379
[*] Performing dry run (mem limit = 0 MiB)
[+] Program exits with a signal, minimization disabled
[*] Stage #0: One-time block normalization
[+] Block normalization complete, 345 blocks
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
  Block length = 32, remaining size = 320 bytes
  Block length = 16, remaining size = 288 bytes
  Block length = 8, remaining size = 256 bytes
  Block length = 4, remaining size = 252 bytes
  Block length = 2, remaining size = 250 bytes
  Block length = 1, remaining size = 249 bytes
[+] Block removal complete, 292 bytes deleted.
[*] Stage #2: Minimizing symbols (11 components)
[+] Symbol minimization finished, 1 symbol removed
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 bytes deleted
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
  Block length = 4, remaining size = 69 bytes
  Block length = 2, remaining size = 67 bytes
  Block length = 1, remaining size = 66 bytes
[+] Block removal complete, 0 bytes deleted.

  File size reduced by : 80.89% (to 69 bytes)
  Characters simplified : 504.35%
  Number of execs done : 222
    Fruitless execs : path=117 crash=0 hang=0

[*] Writing output to '/tmp/out_crash/2'...
[+] We're done here. Have a nice day!
```



```
_crash
2d73 0030 3030 0edit.-h0.-s.000
3030 3030 3030 00000000\..000000
3030 3030 3030 0000000000000000
3030 3030 3030 0000000000000000
```

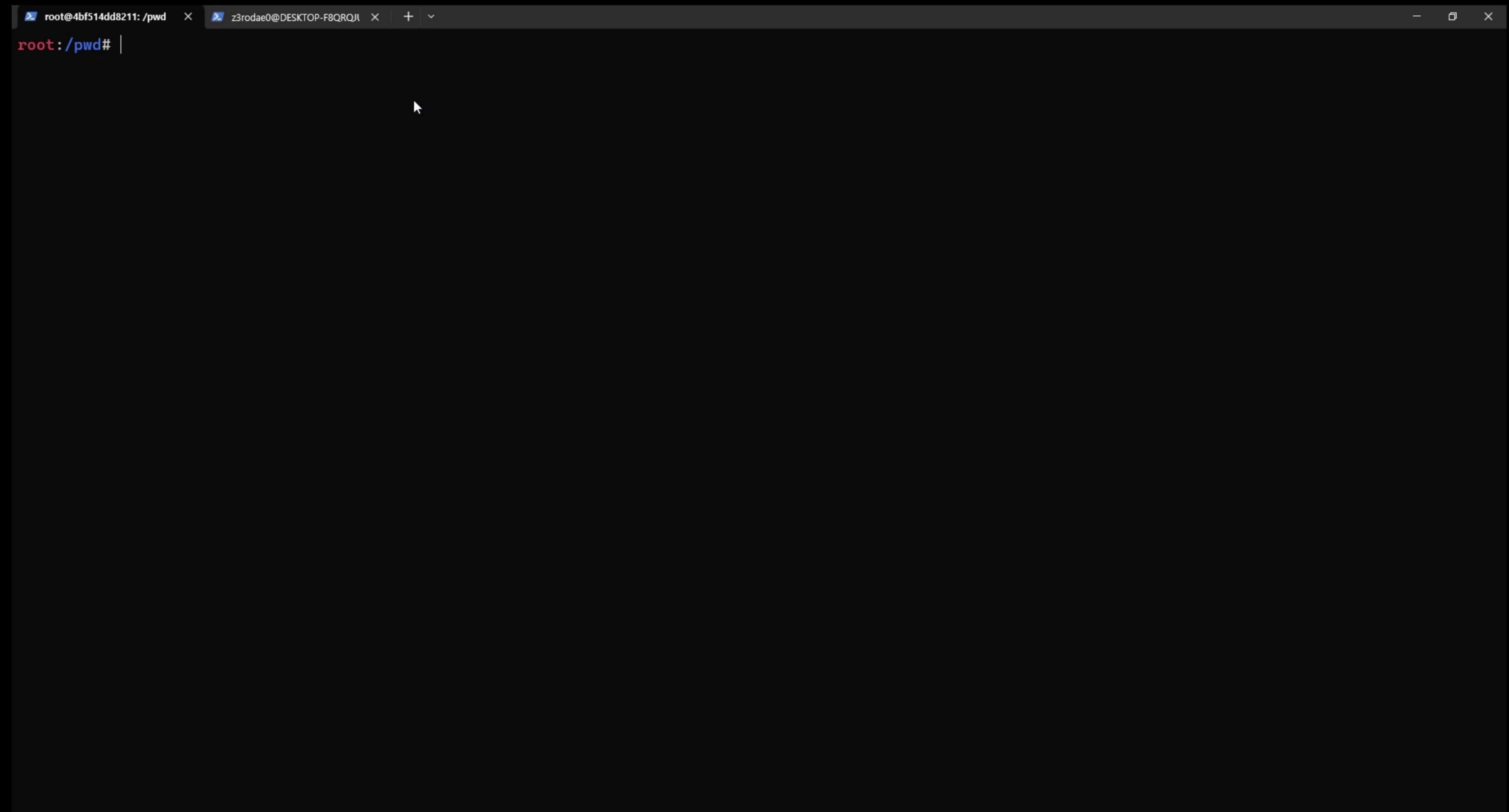
invoked as "sudoedit". \*/

me + proglen - 4, "edit") == 0 {

```
sudo_settings_t *me_sudoedit, .value = "true";
}
```

## correct crash

# Minimizing AFL Testcase



The image shows a screenshot of a terminal window with two tabs. The left tab is titled "root@4bf514dd8211: /pwd" and the right tab is titled "z3rodae0@DESKTOP-F8QRQJL". The "root" tab contains the command "root:/pwd# |" followed by a cursor. The "z3rodae0" tab is partially visible.

**QnA**