
In The Wild Introduction
Sudo Heap Buffer Overflow
CVE-2021-3156


WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING

- 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

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

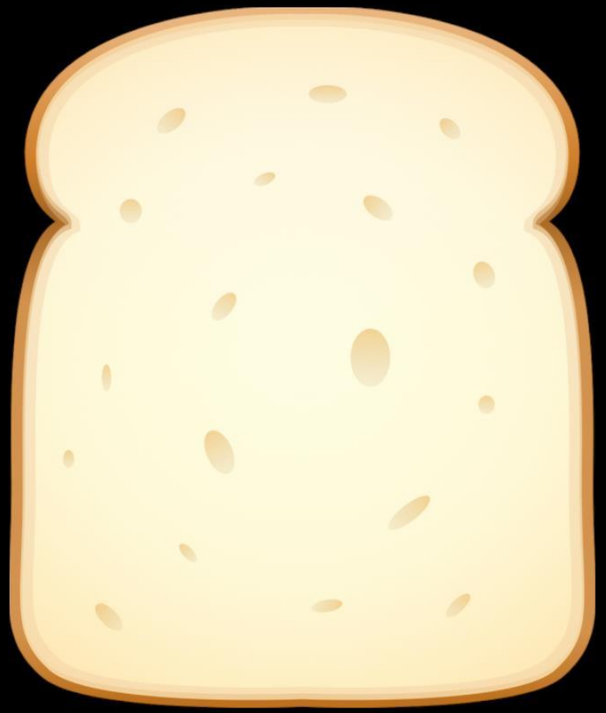
Table of Contents

What is Sudo?

What is Sudo?

What is Sudo?

- Unix 계열 OS 유틸리티

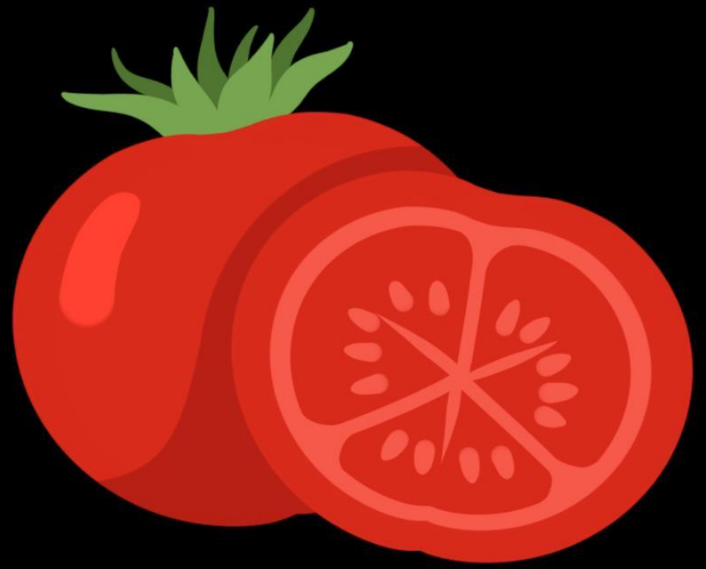


What is Sudo?

- 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 사용되는 메모리 영역

What is Heap Memory?

- 힙 데이터 구조와 힙 메모리는 관련이 없다 (Pool)
- 동적으로 할당된 메모리를 관리하는데 사용되는 메모리 영역
- 스택과 달리 낮은 주소에서 높은 주소로 확장된다.

What is BoF?

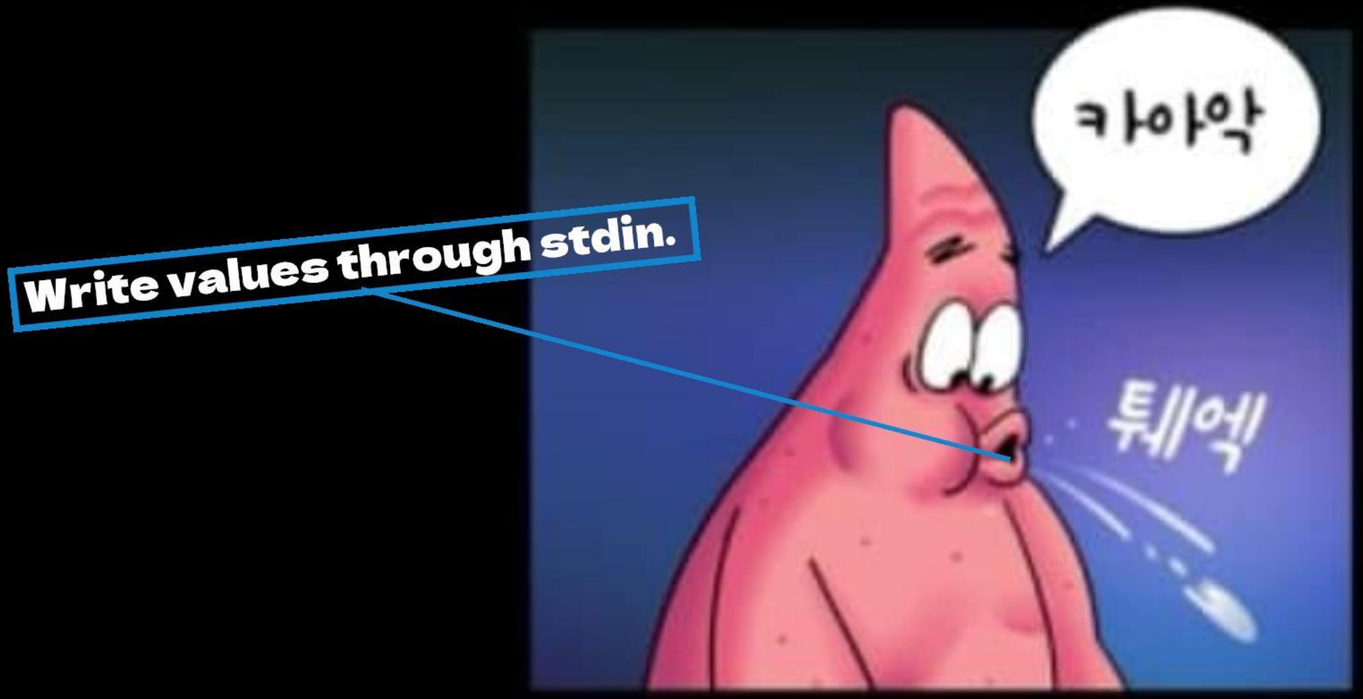
What is BoF?

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?

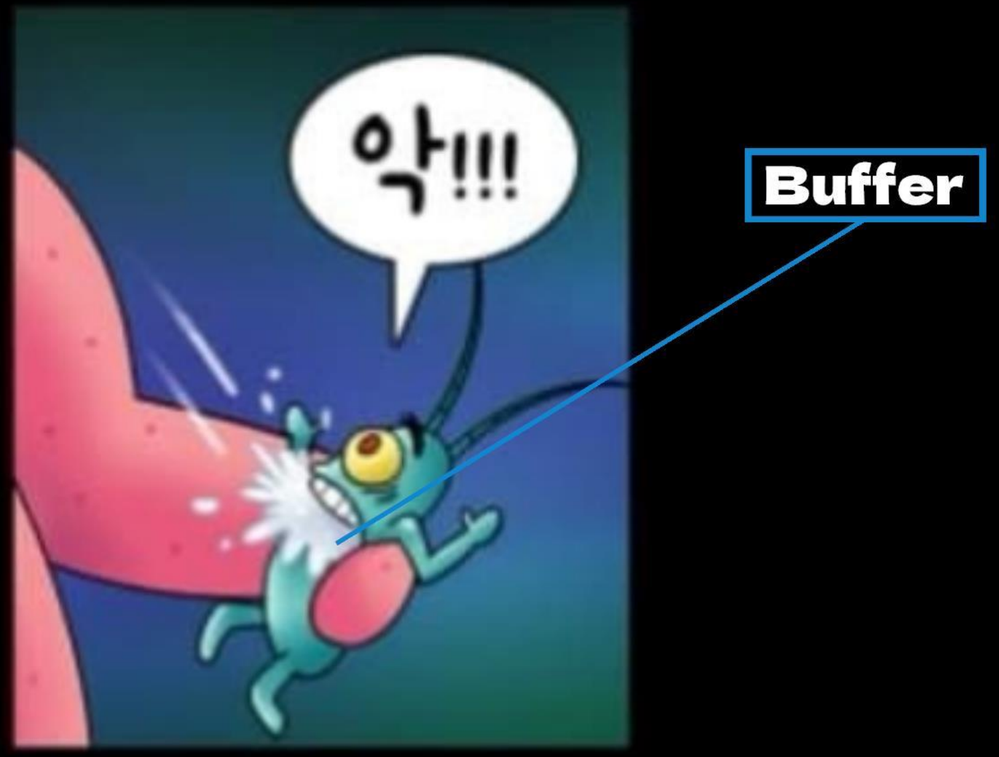
버퍼는 일반적으로 배열을 의미하지만, 모든 배열이 버퍼는 아니다

```c
#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

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

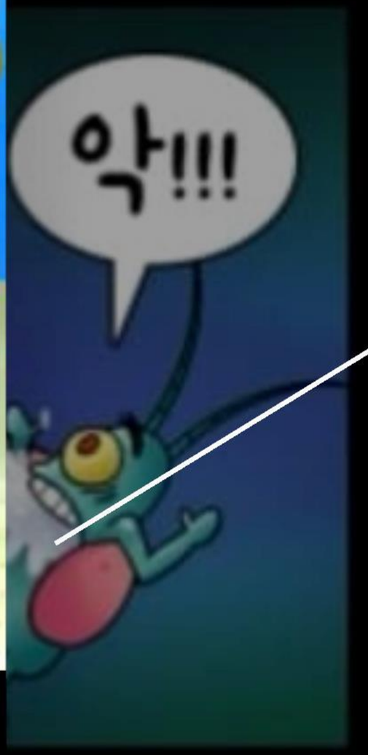


What is BoF?

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, buff
        buffer[len] = '\x0
        write(1, buffer, 9
        return 0;
}
```

NO!!!



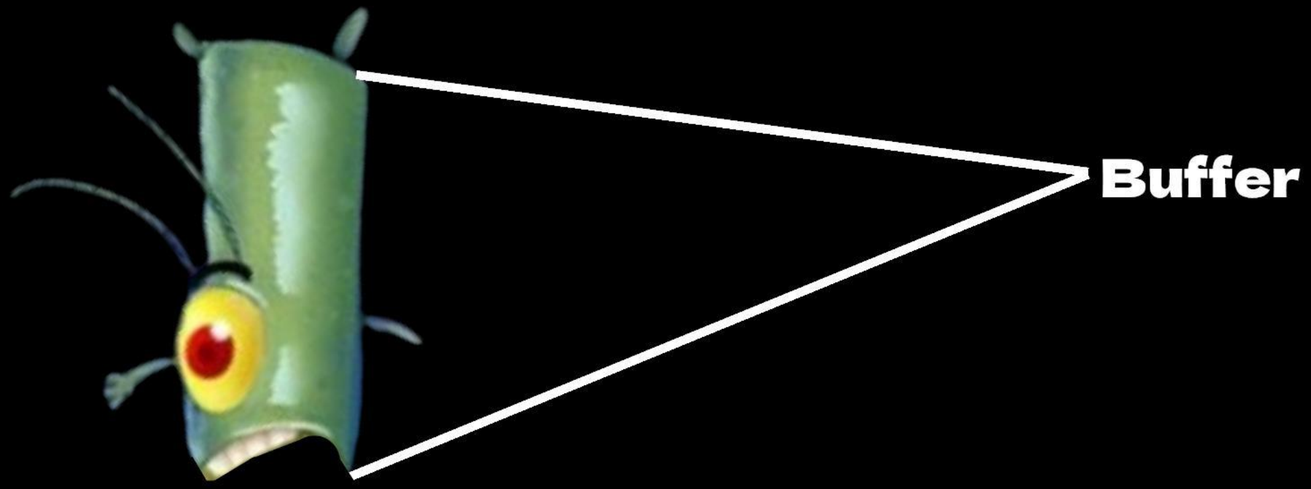
Buffer

What is BoF?

할당된 메모리 공간 버퍼의 크기를 초과하여 입력한다.

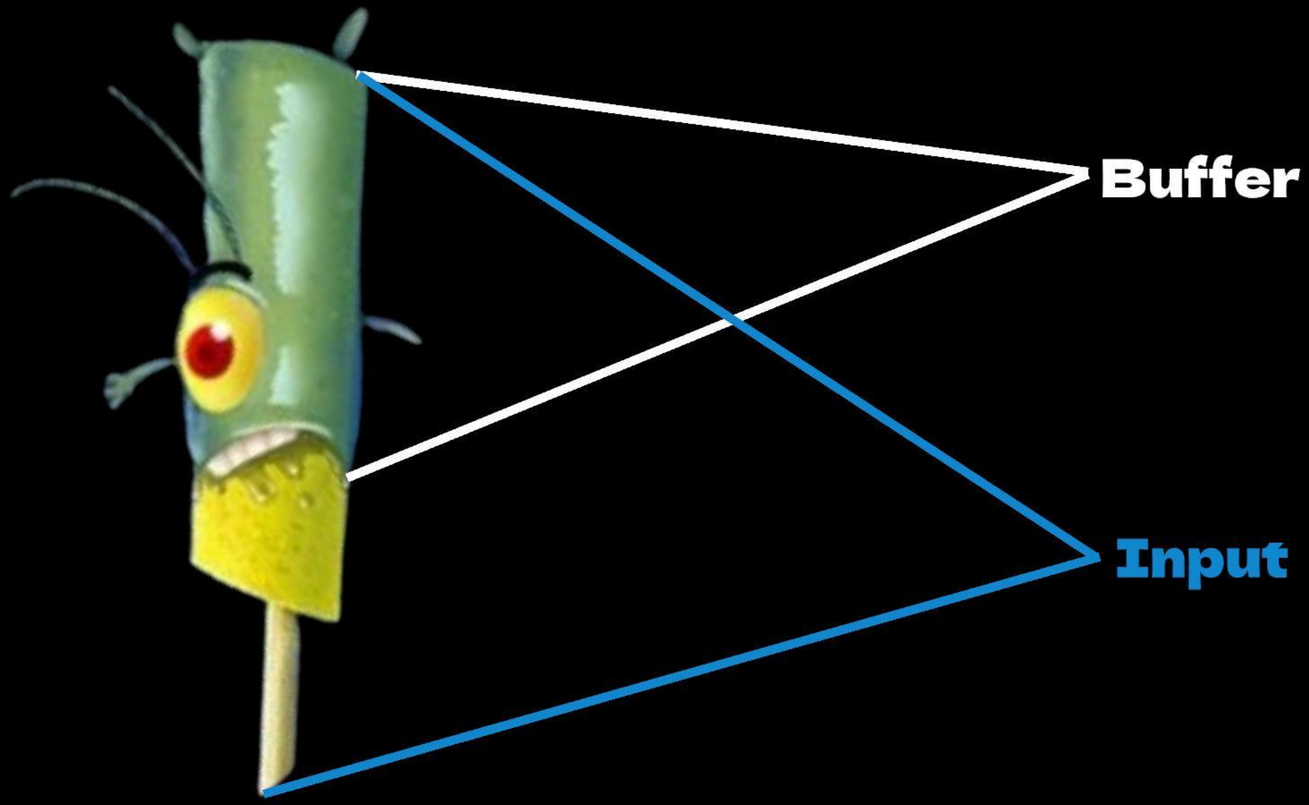
What is BoF?

할당된 메모리 공간 버퍼의 크기를 초과하여 입력한다.



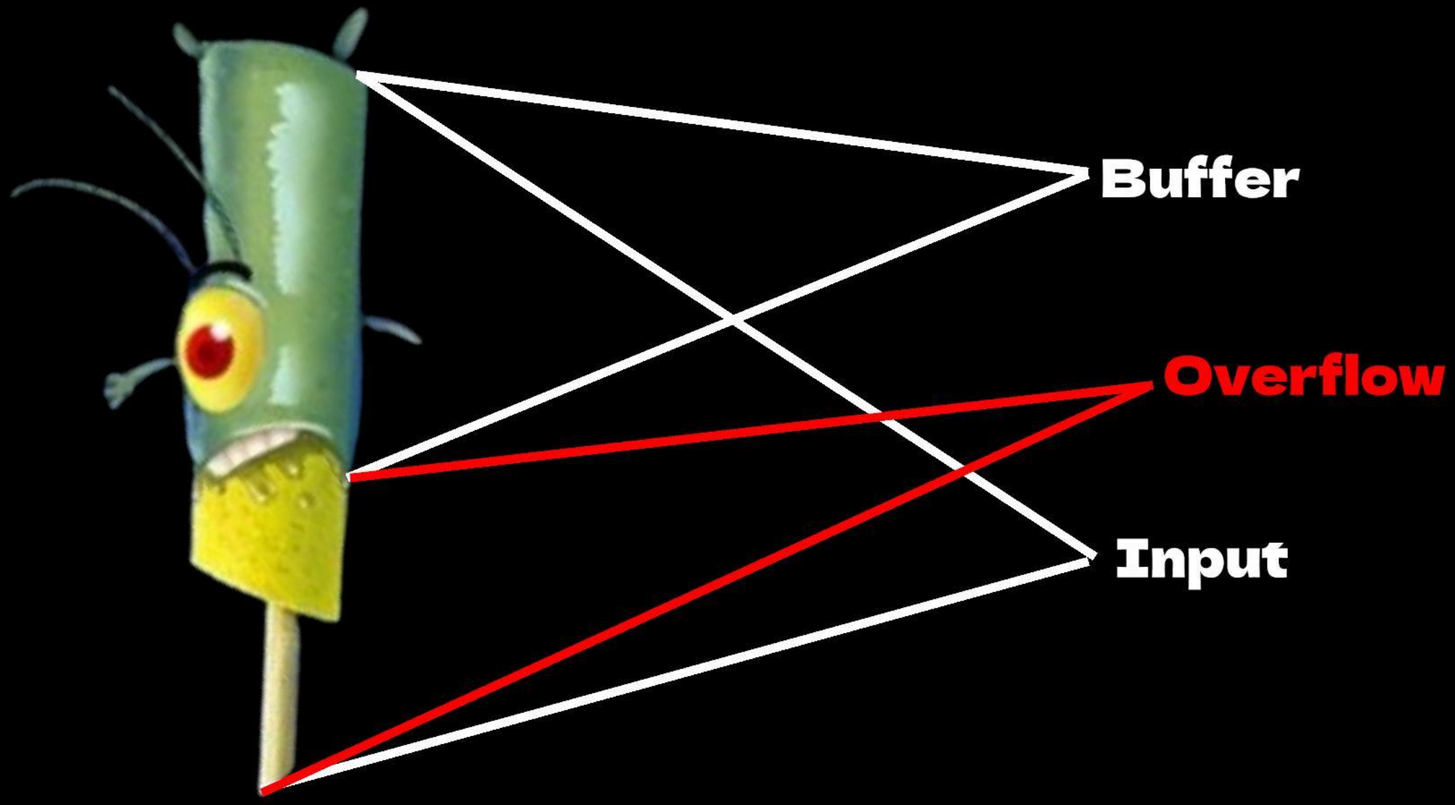
What is BoF?

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

```c
#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

```c
#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


```c
#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-based보다 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 Instrumentation이 필요하다.

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.

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)

```c
#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;
}
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

What is Fuzzing?

Code Coverage(afl-compiler)

```c
#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;
}
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

What is Fuzzing?

Code Coverage(afl-compiler)

```c
#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;
}
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

What is Fuzzing?

Code Coverage(afl-compiler)

```c
#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;
}
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

What is Fuzzing?

Code Coverage(afl-compiler)

```c
#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;
}
```

```c
int __fastcall func_1(__int64 a1, __int64 a2, __int64 a3)
{
  _afl_maybe_log(a1, a2, a3, 65507LL);
  return puts("func_1");
}
```

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

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");
}
```

```
}
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rcx
  __int64 v6; // [rsp+0h] [rbp-B0h]
  __int64 v7; // [rsp+8h] [rbp-A8h]
  __int64 v9; // [rsp+20h] [rbp-90h]
  int v10[3]; // [rsp+A4h] [rbp-Ch] BYREF

  v9 = v3;
  _afl_maybe_log(argc, argv, envp, 23617LL);
  __isoc99_scanf(&unk_402004, v10, envp, v9);
  v6 = (unsigned int)v10[0];
  if ( v10[0] )
  {
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 13635LL);
    func_1((__int64)&unk_402004, (__int64)v10, v6);
  }
  else
  {
    v7 = v4;
    _afl_maybe_log(&unk_402004, v10, (unsigned int)v10[0], 60836LL);
    func_2(&unk_402004, v10, v6, v7);
  }
  return 0;
}
```

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?

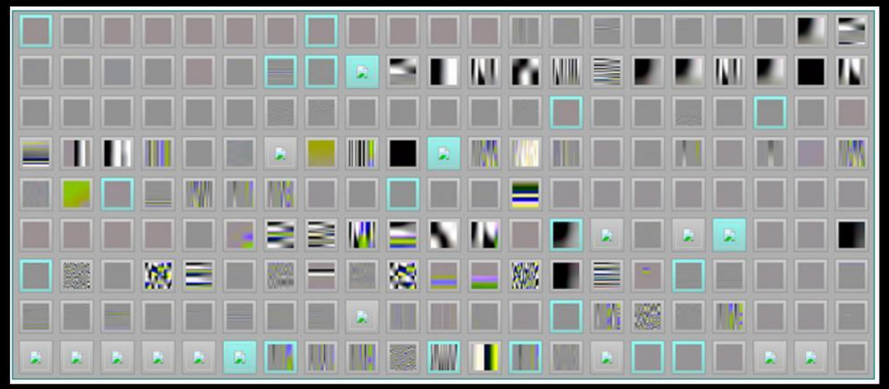
American Fuzzy Lop Plusplus

- Coverage-based 퍼저
- 컴파일 타임과 런타임에 계측 코드를 삽입하여 코드 커버리지를 측정

What is Fuzzing?

American Fuzzy Lop Plusplus

- Coverage-based 퍼저
- 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

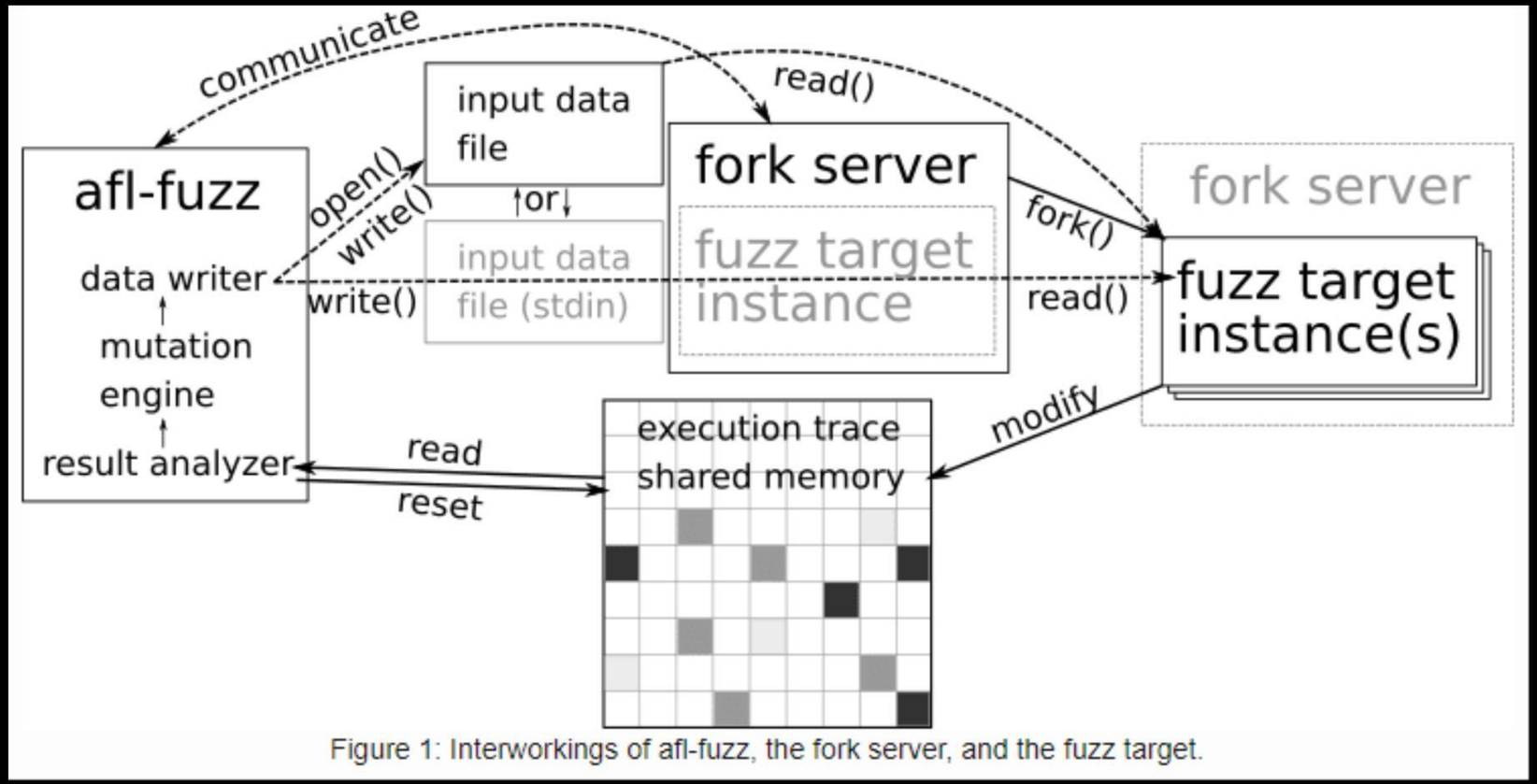


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

What is Fuzzing?

AFL Structure

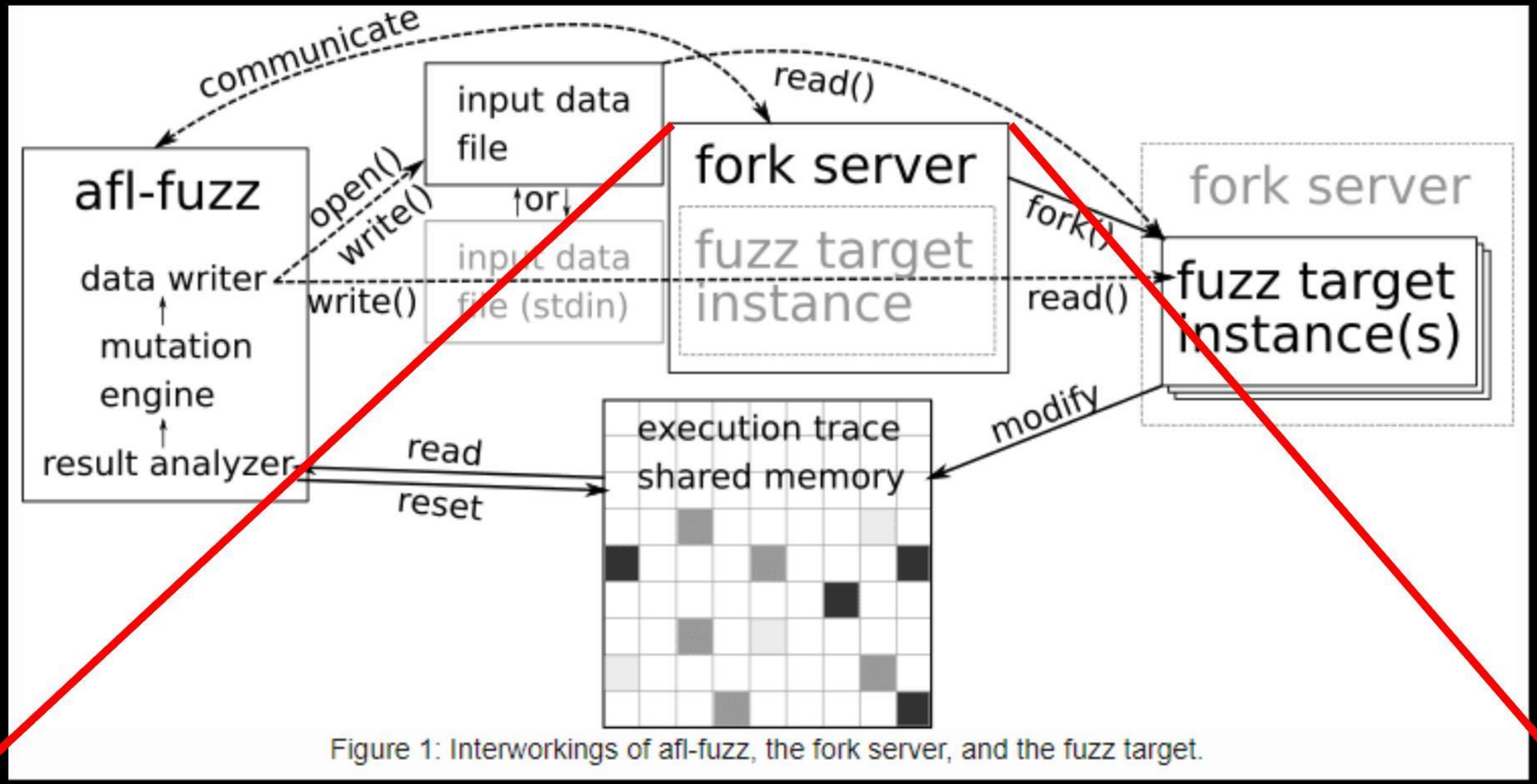


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

Create the program you want to fuzz with

What is Fuzzing?

AFL Structure

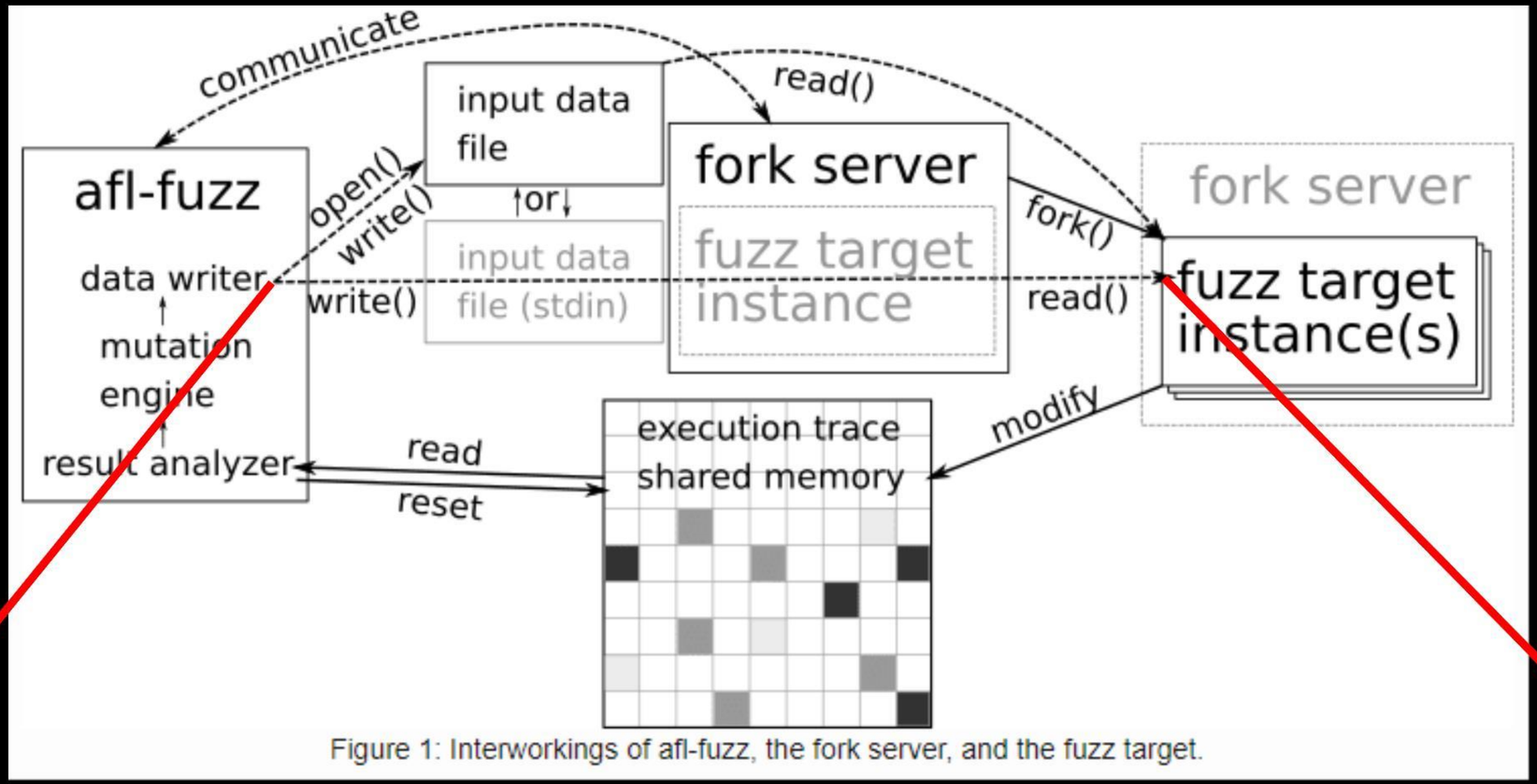


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

Generated program and pipe communication

What is Fuzzing?

AFL Structure

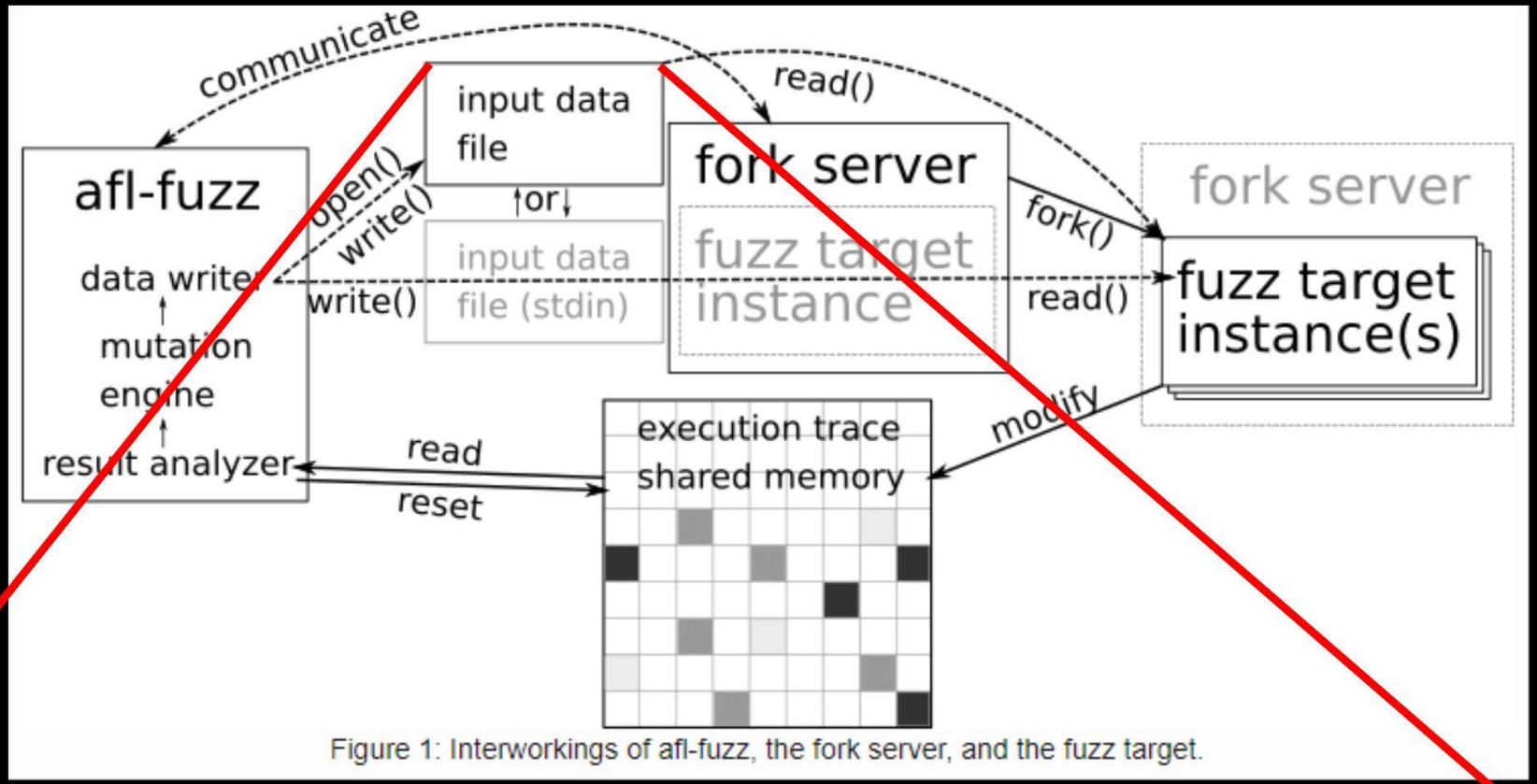


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

Passes the value or file used as standard input.

What is Fuzzing?

AFL Structure

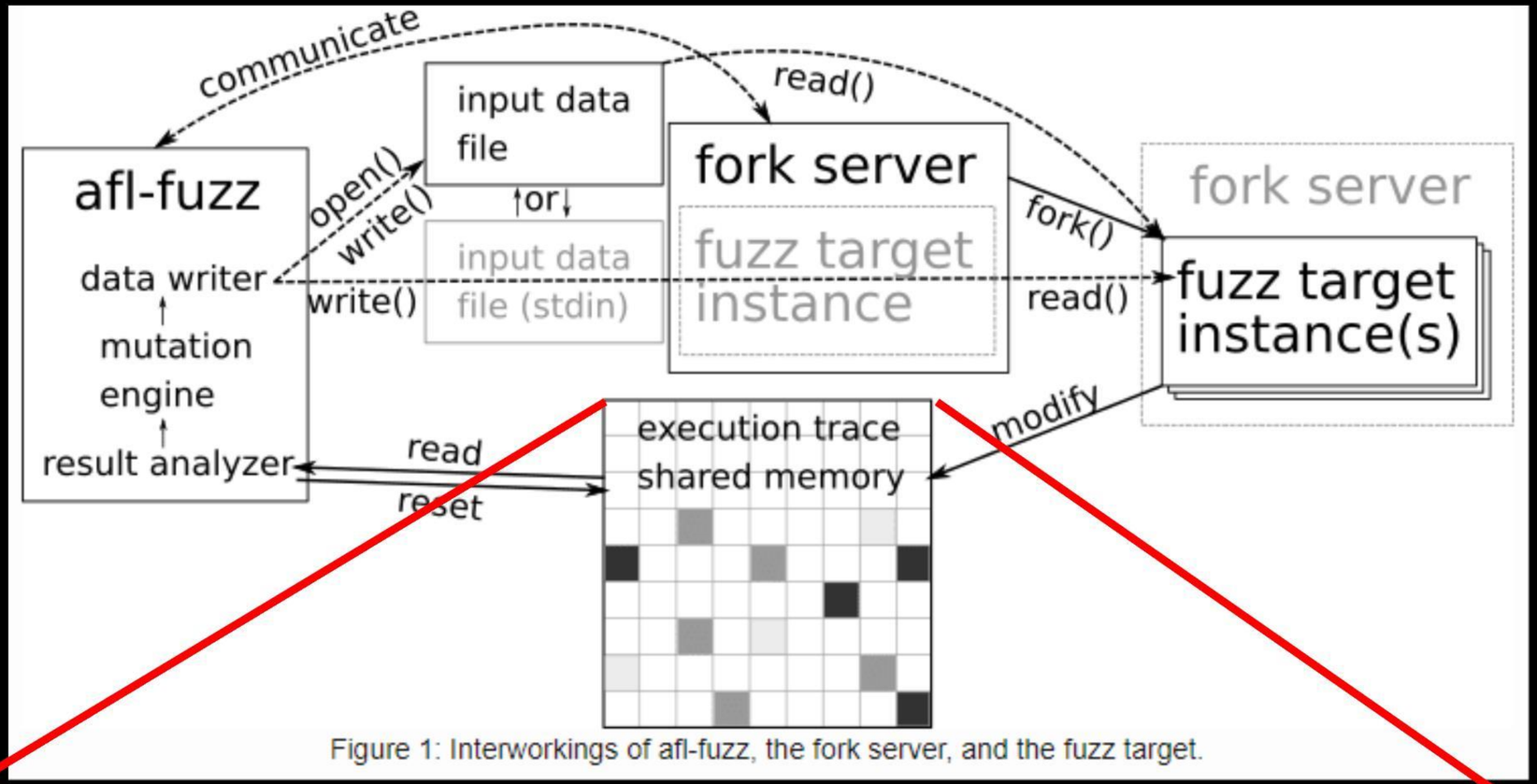


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

Write execution results to shared memory

What is Fuzzing?

AFL Structure

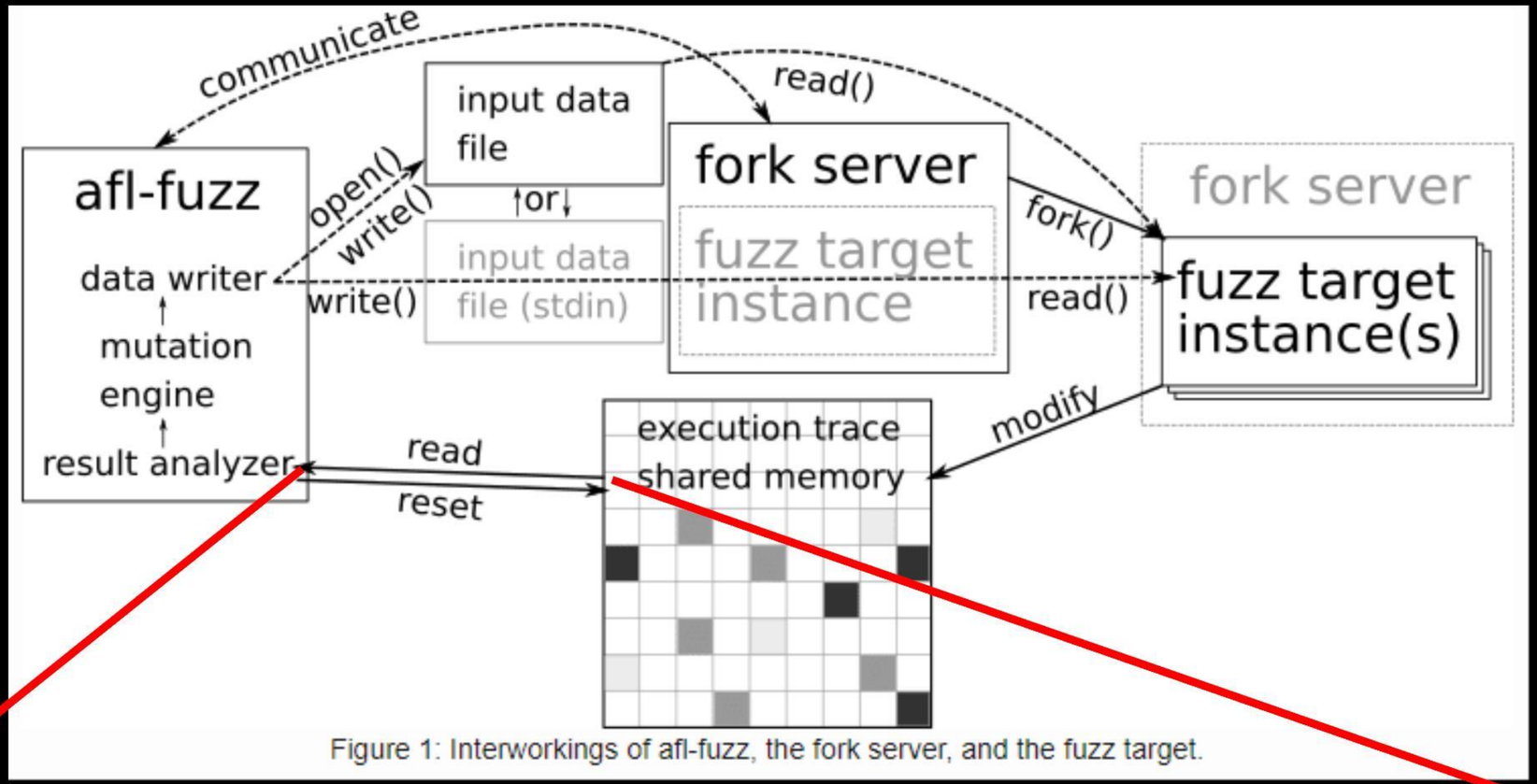


Figure 1: Interworkings of afl-fuzz, the fork server, and the fuzz target.

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


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

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 33 (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 33 (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 33 (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 these reports.

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 33 (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)

```
- char *to, **from;
  size_t size, n;

  /* Alloc and build up user_args. */
- for (size = 0, from = NewArgv + 1; *from; from++)
-     size += strlen(*from) + 1;
  user_args = emalloc(size);
- for (to = user_args, from = NewArgv + 1; *from; from++) {
-     n = strlcpy(to, *from, size - (to - user_args));
-     if (n >= size - (to - user_args))
-         errorx(1, _("internal error, set_cmnd() overflow"));
-     to += n;
-     *to++ = ' ';
```

```
+ char *to, *from, **av;
  size_t size, n;

  /* Alloc and build up user_args. */
+ for (size = 0, av = NewArgv + 1; *av; av++)
+     size += strlen(*av) + 1;
  user_args = emalloc(size);
+ if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
+     /*
+      * When running a command via a shell, the sudo front-end
+      * escapes potential meta chars.  We unescape non-spaces
+      * for sudoers matching and logging purposes.
+      */
+     for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
+         while (*from) {
+             if (from[0] == '\\' && !isspace((unsigned char)from[1]))
+                 from++;
+             *to++ = *from++;
+         }
+         *to++ = ' ';
+     }
+     *--to = '\0';
+ } else {
+     for (to = user_args, av = NewArgv + 1; *av; av++) {
+         n = strlcpy(to, *av, size - (to - user_args));
+         if (n >= size - (to - user_args))
+             errorx(1, _("internal error, set_cmnd() overflow"));
+         to += n;
+         *to++ = ' ';
+     }
+     *--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
852     for (size = 0, av = NewArgv
853     size += strlen(*av) + 1;
854     if (size == 0 || (user_args
855     sudo_warnx(U_("%s: %s"), __
856     debug_return_int(-1);
857     }
858     if (ISSET(sudo_mode, MODE_S
859     /*
860      * When running a command v
861      * escapes potential meta c
862      * for sudoers matching and
863      */
864     for (to = user_args, av = N
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 = N
875         n = strlcpy(to, *av, si
876         if (n >= size - (to - u
877         sudo_warnx(U_("interna
878         debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884     }
```

View in detail

…hs of Argv.

…cate the heap
…length.

…character by
… heap.

…ring, the from
…es once more.

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 \xOO A A A A \ \xOO z 3 r o d a e O

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 \xOO A A A A \ \xOO z 3 r o d a e O

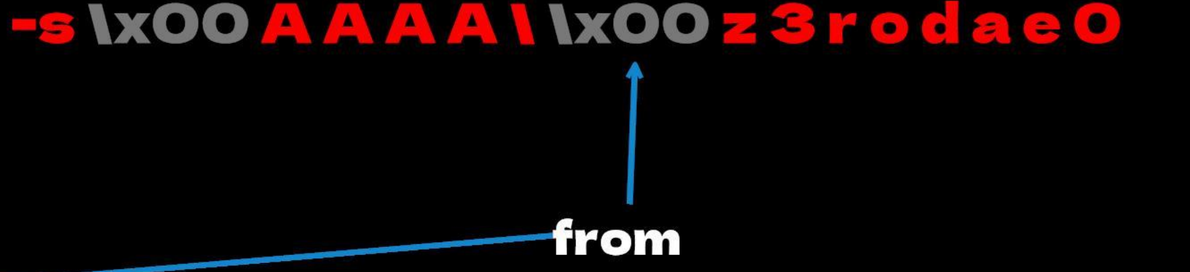
from

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.



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 \xOO A A A A \ \xOO z 3 r o d a e O

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] == '\\' && !isc((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 \xOO A A A A \ \xOO z 3 r o d a e O

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] == '\\' && !isalnum((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 \xOO A A A A \ \xOO z 3 r o d a e O

from

Write out of bounds string because it is not null

```
root:/pwd/sudo-1.8.31p2/src# ./sudoedit -s aaaaaaaa\\
from = aaaaaaaa\
*from = a
from = aaaaaaa\
*from = a
from = aaaaaa\
*from = a
from = aaaaa\
*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/su

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

vulnerability

으면 from 포인터가 증가된

xOOz3rodaeO

from

dit -s aaaaaaaa\\

```
*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] == '\\' && !isalpha((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_("%s: %s"), __func__, U_("argument too large"));
878         debug_return_int(-1);
879         }
880         to += n;
881         *to++ = ' ';
882     }
883     *--to = '\0';
884     }
```

명령줄 인수에 '\' 문자가 있으면 from 포인터가 증가된다.

```
-s \xOO A A A A \\xOO z 3 r o d a e O
```

from

Write out of bounds string because it is not null

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

취약한 코드를 실행시키려면 어떻게 해야할까?

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)) {
```

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

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

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

```
/*
  Copyright 2015 Google LLC All rights reserved.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at:

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
*/


/*
   american fuzzy lop - sample argv fuzzing wrapper
   ------------------------------------------------

   Written by Michal Zalewski <lcamtuf@google.com>

   This file shows a simple way to fuzz command-line parameters with stock
   afl-fuzz. To use, add:

   #include "/path/to/argv-fuzz-inl.h"

   ...to the file containing main(), ideally placing it after all the
   standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
   main().
```

Find vulnerabilities with fuzzing

AFL/experimental/argv_fuzzing/argv-fuzz-inl.h

```
/*
  Copyright 2015 Google LLC All rights reserved.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at:

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
*/


/*
   american fuzzy lop - sample argv fuzzing wrapper
   ------------------------------------------------

   Written by Michal Zalewski <lcamtuf@google.com>

   This file shows a simple way to fuzz command-line parameters with stock
   afl-fuzz. To use, add:

   #include "/path/to/argv-fuzz-inl.h"

   ...to the file containing main(), ideally placing it after all the
   standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
   main().
```

```
#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

```
/*
  Copyright 2015 Google LLC All rights reserved.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at:

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
*/


/*
   american fuzzy lop - sample argv fuzzing wrapper
   ------------------------------------------------

   Written by Michal Zalewski <lcamtuf@google.com>

   This file shows a simple way to fuzz command-line parameters with stock
   afl-fuzz. To use, add:

   #include "/path/to/argv-fuzz-inl.h"

   ...to the file containing main(), ideally placing it after all the
   standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
   main().
```

```
#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

```
/*
  Copyright 2015 Google LLC All rights reserved.

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at:

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
*/


/*
   american fuzzy lop - sample argv fuzzing wrapper
   ------------------------------------------------

   Written by Michal Zalewski <lcamtuf@google.com>

   This file shows a simple way to fuzz command-line parameters with stock
   afl-fuzz. To use, add:

   #include "/path/to/argv-fuzz-inl.h"

   ...to the file containing main(), ideally placing it after all the
   standard includes. Next, put AFL_INIT_ARGV(); near the very beginning of
   main().
```

```
#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_GETPRPWNAM) && defined(HAVE_SET_AUTH_PARAMETERS)
163      (void) set_auth_parameters(argc, argv);
164  # ifdef HAVE_INITPRIVS
165      initprivs();
166  # endif
167  #endif /* HAVE_GETPRPWNAM && 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_GETPRPWNAM) && defined(HAVE_SET_AUTH_PARAMETERS)
163      (void) set_auth_parameters(argc, argv);
164  # ifdef HAVE_INITPRIVS
165      initprivs();
166  # endif
167  #endif /* HAVE_GETPRPWNAM && 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
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_GETPRPWNAM) && defined(HAVE_SET_AUTH_PARAMETERS)
163      (void) set_auth_parameters(argc, argv);
164  # ifdef HAVE_INITPRIVS
165      initprivs();
166  # endif
167  #endif /* HAVE_GETPRPWNAM && 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

```
int
main(int argc, char *argv[], char *envp[])
{
    int narge, ok, status = 0;
    char **nargv, **env_add;
    char **user_info, **command_info, **argv_out, **user_env_out;
    struct sudo_settings *settings;
    struct plugin_container *plugin, *next;
    sigset_t mask;
    debug_decl_vars(main, SUDO_DEBUG_MAIN)

    initprogname(argc > 0 ? argv[0] : "sudo");

    /* Crank resource limits to unlimited. */
    unlimit_sudo();

    /* Make sure fds 0-2 are open and do OS-specific initialization. */
    fix_fds();
    os_init(argc, argv, envp);

    setlocale(LC_ALL, "");
    bindtextdomain(PACKAGE_NAME, LOCALEDIR);
    textdomain(PACKAGE_NAME);

    (void) tzset();

    /* Must be done before we do any password lookups */
#if defined(HAVE_GETPRPWNAM) && defined(HAVE_SET_AUTH_PARAMETERS)
    (void) set_auth_parameters(argc, argv);
# ifdef HAVE_INITPRIVS
    initprivs();
# endif
#endif /* HAVE_GETPRPWNAM && HAVE_SET_AUTH_PARAMETERS */

    /* Initialize the debug subsystem. */
    if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
    exit(EXIT_FAILURE);
    sudo_debug_instance = sudo_debug_register(getprogname(),
    NULL, NULL, sudo_conf_debug_files(getprogname()));
    if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
    exit(EXIT_FAILURE);

    /* Make sure we are setuid root. */
    sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

```
#include "argv-fuzz-inl.h"

AFL_INIT_ARGV();
```

Find vulnerabilities with fuzzing

sudo/src/sudo.c

```
int
main(int argc, char *argv[], char *envp[])
{
    int nargc, ok, status = 0;
    char **nargv, **env_add;
    char **user_info, **command_info, **argv_out, **user_env_out;
    struct sudo_settings *settings;
    struct plugin_container *plugin, *next;
    sigset_t mask;
    debug_decl_vars(main, SUDO_DEBUG_MAIN)

    initprogname(argc > 0 ? argv[0] : "sudo");

    /* Crank resource limits to unlimited. */
    unlimit_sudo();

    /* Make sure fds 0-2 are open and do OS-specific initialization. */
    fix_fds();
    os_init(argc, argv, envp);

    setlocale(LC_ALL, "");
    bindtextdomain(PACKAGE_NAME, LOCALEDIR);
    textdomain(PACKAGE_NAME);

    (void) tzset();

    /* Must be done before we do any password lookups */
#if defined(HAVE_GETPRPWNAM) && defined(HAVE_SET_AUTH_PARAMETERS)
    (void) set_auth_parameters(argc, argv);
# ifdef HAVE_INITPRIVS
    initprivs();
# endif
#endif /* HAVE_GETPRPWNAM && HAVE_SET_AUTH_PARAMETERS */

    /* Initialize the debug subsystem. */
    if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) == -1)
    exit(EXIT_FAILURE);
    sudo_debug_instance = sudo_debug_register(getprogname(),
    NULL, NULL, sudo_conf_debug_files(getprogname()));
    if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
    exit(EXIT_FAILURE);

    /* Make sure we are setuid root. */
    sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

```
#include "argv-fuzz-inl.h"

AFL_INIT_ARGV();
```

```
#include "argv-fuzz-inl.h"

int
main(int argc, char *argv[], char *envp[])
{
    ARGV_INIT_ARGV();
    int nargc, ok, status = 0;
    char **nargv, **env_add;
    char **user_info, **command_info, **argv_out, **user_env_out;
    struct sudo_settings *settings;
    struct plugin_container *plugin, *next;
    sigset_t mask;
    debug_decl_vars(main, SUDO_DEBUG_MAIN)
```

Find vulnerabilities with fuzzing

sudo/src/sudo.c

```
int
main(int argc, char *argv[], char *envp[])
{
    int nargc, status = 0;
    char **nargv, **env_add;
    char **user_info, **command_info, **argv_out
    struct sudo_settings *settings;
    struct plugin_container *plugin, *next;
    sigset_t mask;
    debug_decl_vars(main, SUDO_DEBUG_MAIN)

    initprogname(argc > 0 ? argv[0] : "sudo");

    /* Crank resource limits to unlimited. */
    unlimit_sudo();

    /* Make sure fds 0-2 are open and do OS-spe
    fix_fds();
    os_init(argc, argv, envp);

    setlocale(LC_ALL, "");
    bindtextdomain(PACKAGE_NAME, LOCALEDIR);
    textdomain(PACKAGE_NAME);

    (void) tzset();

    /* Must be done before we do any password l
#if defined(HAVE_GETPRPWNAM) && defined(HAVE_SE
    (void) set_auth_parameters(argc, argv);
# ifdef HAVE_INITPRIVS
    initprivs();
# endif
#endif /* HAVE_GETPRPWNAM && HAVE_SET_AUTH_PARA

    /* Initialize the debug subsystem. */
    if (sudo_conf_read(NULL, SUDO_CONF_DEBUG) =
    exit(EXIT_FAILURE);
    sudo_debug_instance = sudo_debug_register(getprogname(),
    NULL, NULL, sudo_conf_debug_files(getprogname()));
    if (sudo_debug_instance == SUDO_DEBUG_INSTANCE_ERROR)
    exit(EXIT_FAILURE);

    /* Make sure we are setuid root. */
    sudo_check_suid(argc > 0 ? argv[0] : "sudo");
```

This allows argv fuzzing.

```
-fuzz-inl.h"

/();

*envp[])

nfo, **argv_out, **user_env_out;
;;
n, *next;

UG_MAIN)
```

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
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
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
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;

```
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
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' [illegible] */
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;
```

This way, even the program name can be fuzzed.

```
62    char* ptr = in_buf;
63    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
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 */
```

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 != NU  && *__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 != NU  && *__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 != NU  && *__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 != NU  && *__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

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();
526     ud->euid = geteuid();
527     ud->gid = getgid();
528     ud->egid = getegid();
```

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

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

Find vulnerabilities with fuzzing

sudo/src/sudo.c

```
509     /* XXX - bound check
510     user_info = realloca
511     if (user_info == NUL
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 = tcgetpg
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_CMDLINE_LEN 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 afte
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

Bounds Checking Code

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  }
```

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 >= 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;
}
```

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

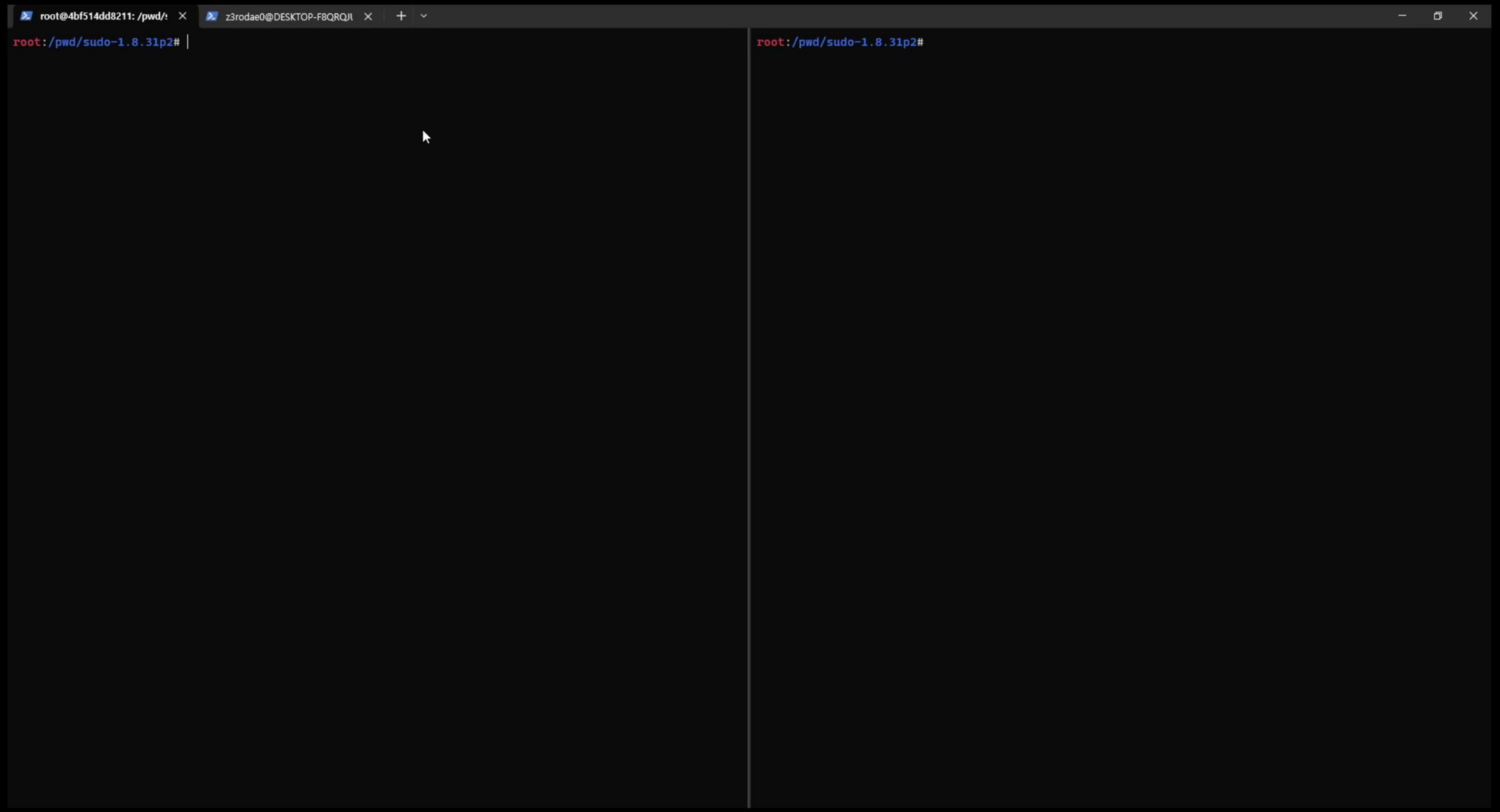
```
if(rc >= MAX_CMDLINE_PAR) {
    break;
}
```

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

Find vulnerabilities with fuzzing

All preparations are complete. Start fuzzing

Find vulnerabilities with fuzzing

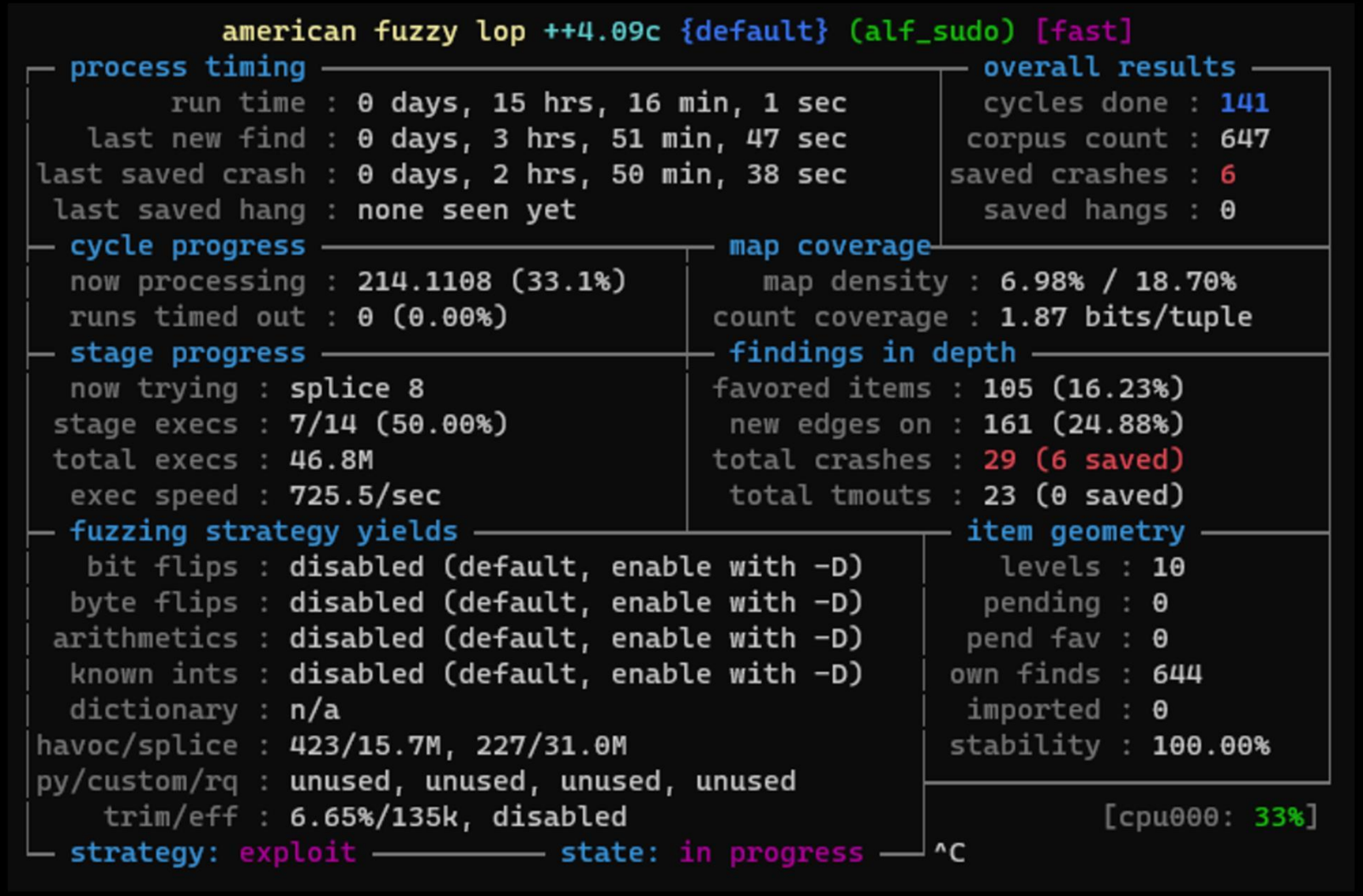


few hours later...

Find vulnerabilities with fuzzing

found crash!!!

Find vulnerabilities with fuzzing



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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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  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  hhhhhhhhhhhhhhhh
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.-nnnnnnnjnnn
00000090: 6e6e 6e6e 002d 2800 2d43 2e41 0011 6200  nnnn.-(.-C.A..b.
000000a0: 2d42 2d41 002d 6e6e 6e6e 6e6e 6e6e 6e6e  -B-A.-nnnnnnnnnn
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/ou
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 M
[+] Program exits with a signal, minimi
[*] Stage #0: One-time block normalizat
[+] Block normalization complete, 345 b
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data..
    Block length = 32, remaining size =
    Block length = 16, remaining size =
    Block length = 8, remaining size =
    Block length = 4, remaining size =
    Block length = 2, remaining size =
    Block length = 1, remaining size =
[+] Block removal complete, 292 bytes d
[*] Stage #2: Minimizing symbols (11 co
[+] Symbol minimization finished, 1 sym
[*] Stage #3: Character minimization...
[+] Character minimization done, 2 byte
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data..
    Block length = 4, remaining size =
    Block length = 2, remaining size =
    Block length = 1, remaining size =
[+] 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!
```

correct crash

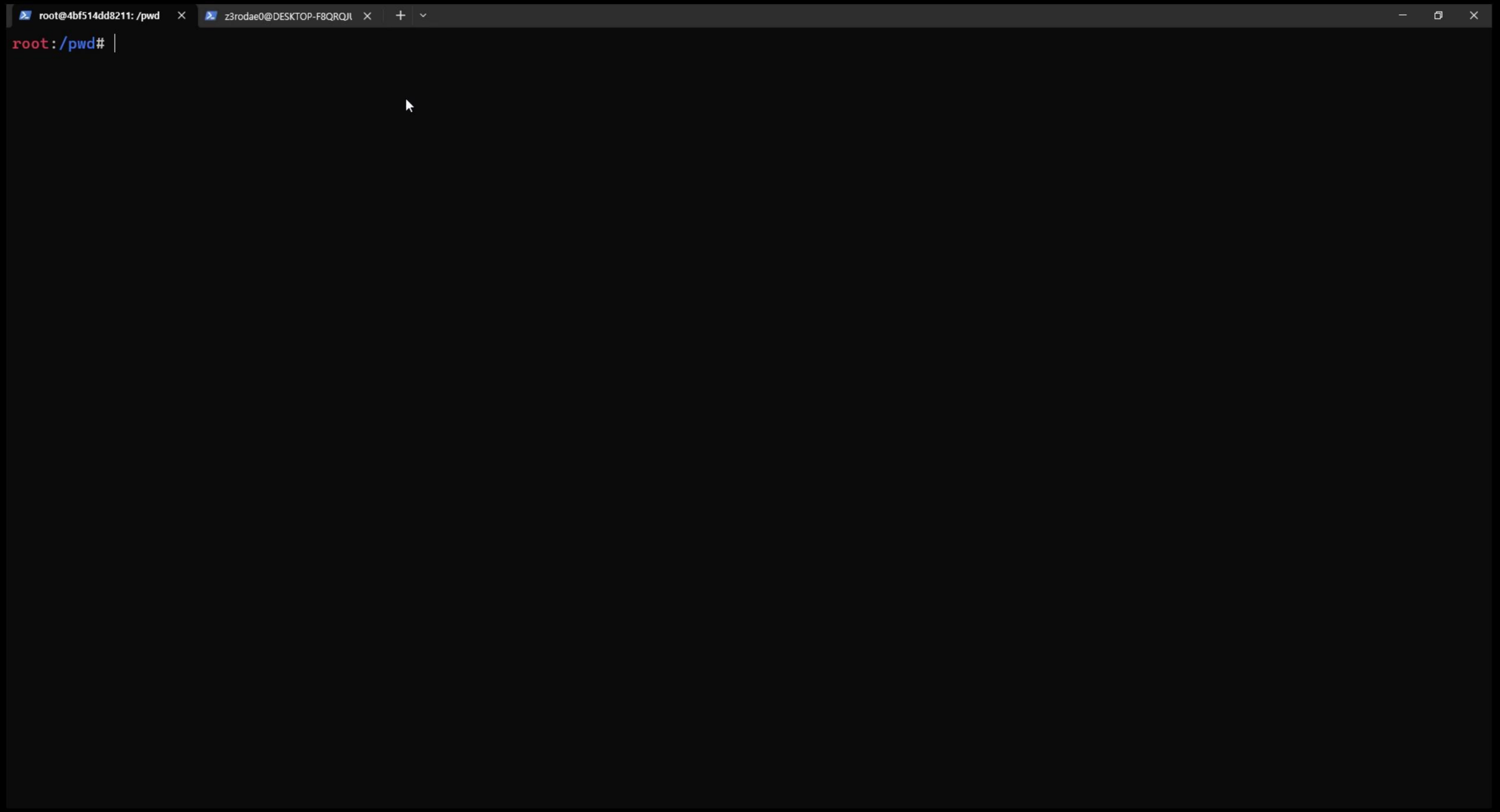
```
_crash
2d73 0030 3030  0edit.-h0.-s.000
3030 3030 3030  00000000\.000000
3030 3030 3030  0000000000000000
3030 3030 3030  0000000000000000

voked as "sudoedit". */

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

.value = "true";
}
```

Minimizing AFL Testcase



QnA