Vulnerability Discovery: SAST and Fuzz Testing

Foundations of Cybersecurity

Giacomo Benedetti

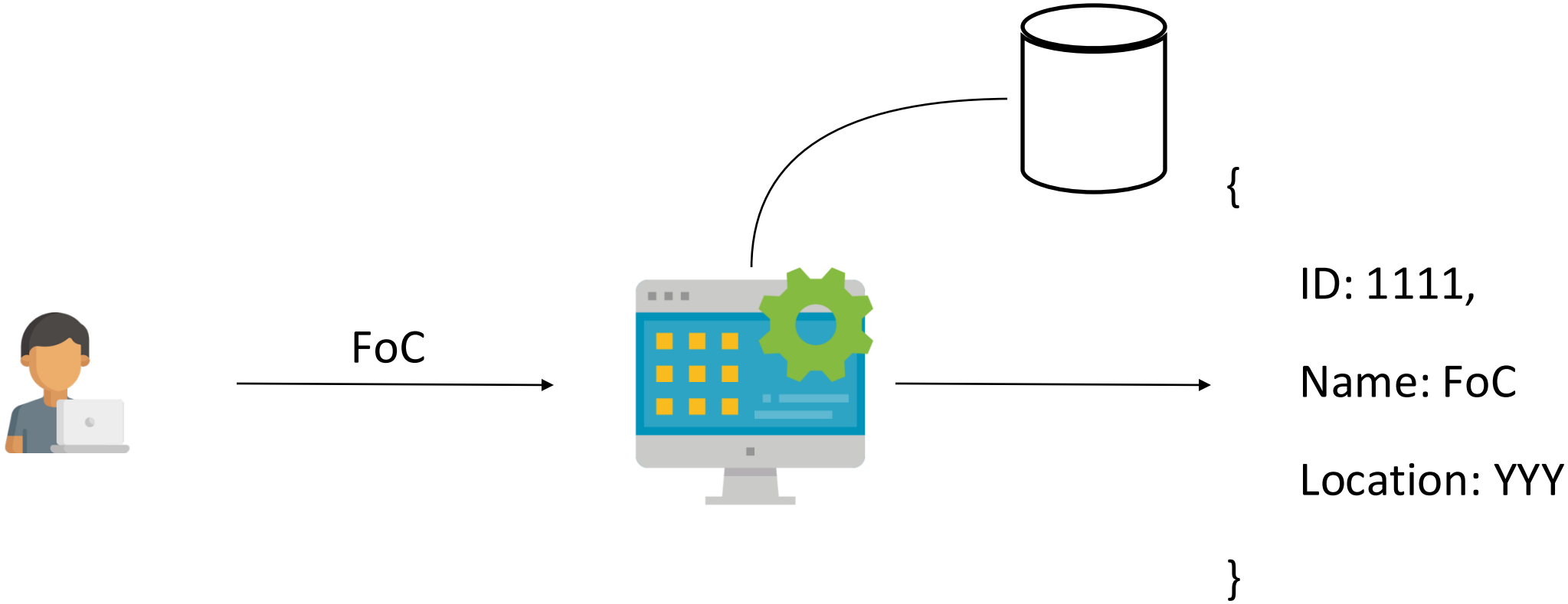
Institute for Applied Mathematics and Information Technologies

National Research Council of Italy

giacomo.benedetti@ge.imati.cnr.it

University of Pavia – Department of Electrical, Computer and Biomedical Engineering

Warm Up



Warm Up

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```

Warm Up

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```


{

ID: 1111,

Name: FoC

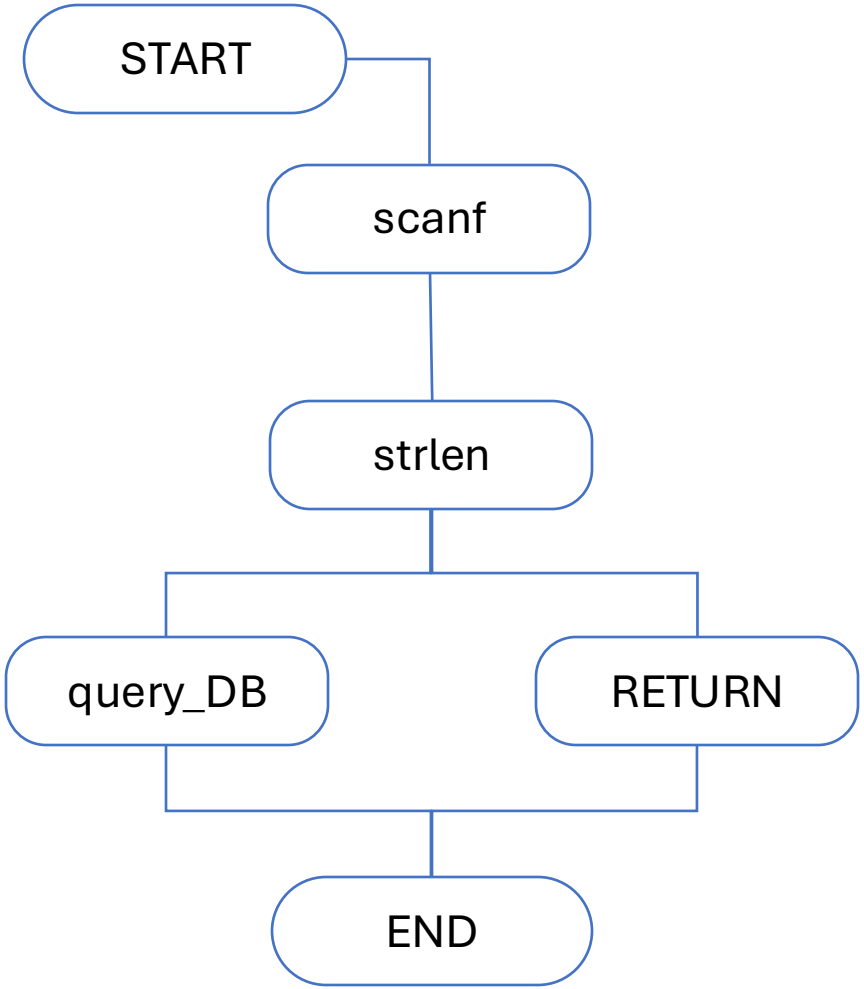
Location: YYY

}

What do we test here?

Warm Up

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```



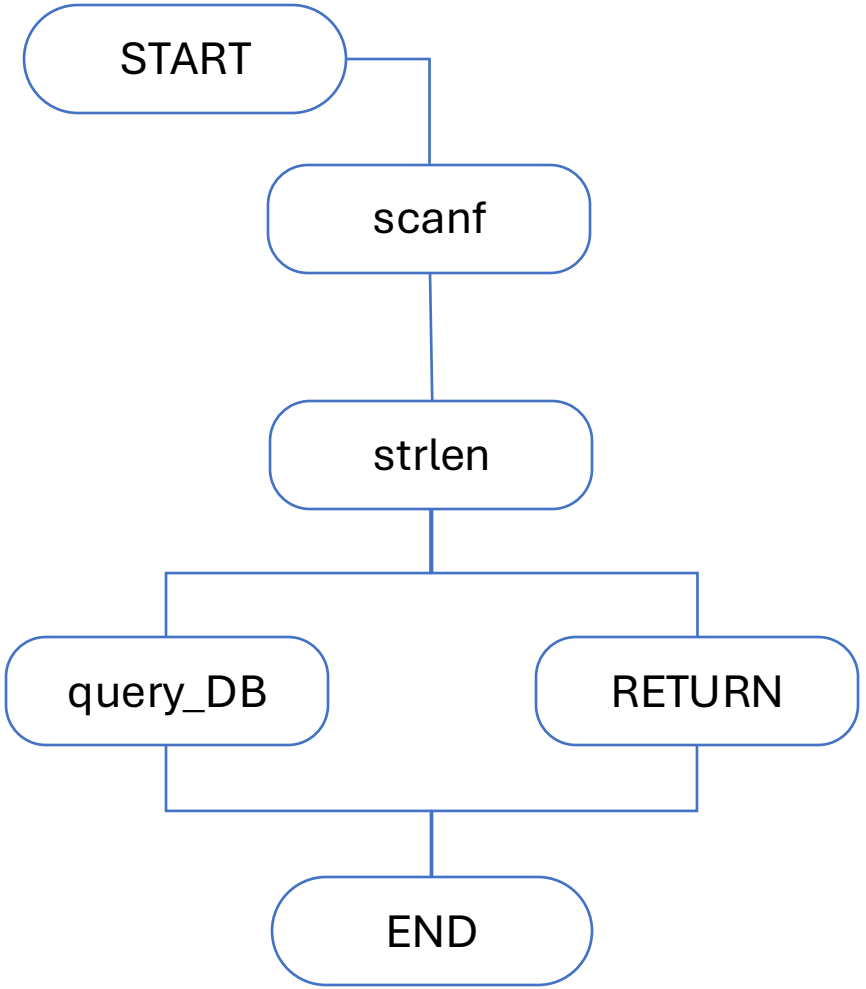
What do we test here?

Warm Up

What do we WANT to test here?

Different types of test

- Functional
- Non-functional
- Security



Three Dimensions of Software Testing

What do we WANT to test here?

Functional

Does it do what it’s supposed to?

Non-functional

How well does it perform?

Security

Can it be broken or exploited?

Three Dimensions of Software Testing

What do we WANT to test here?

Security

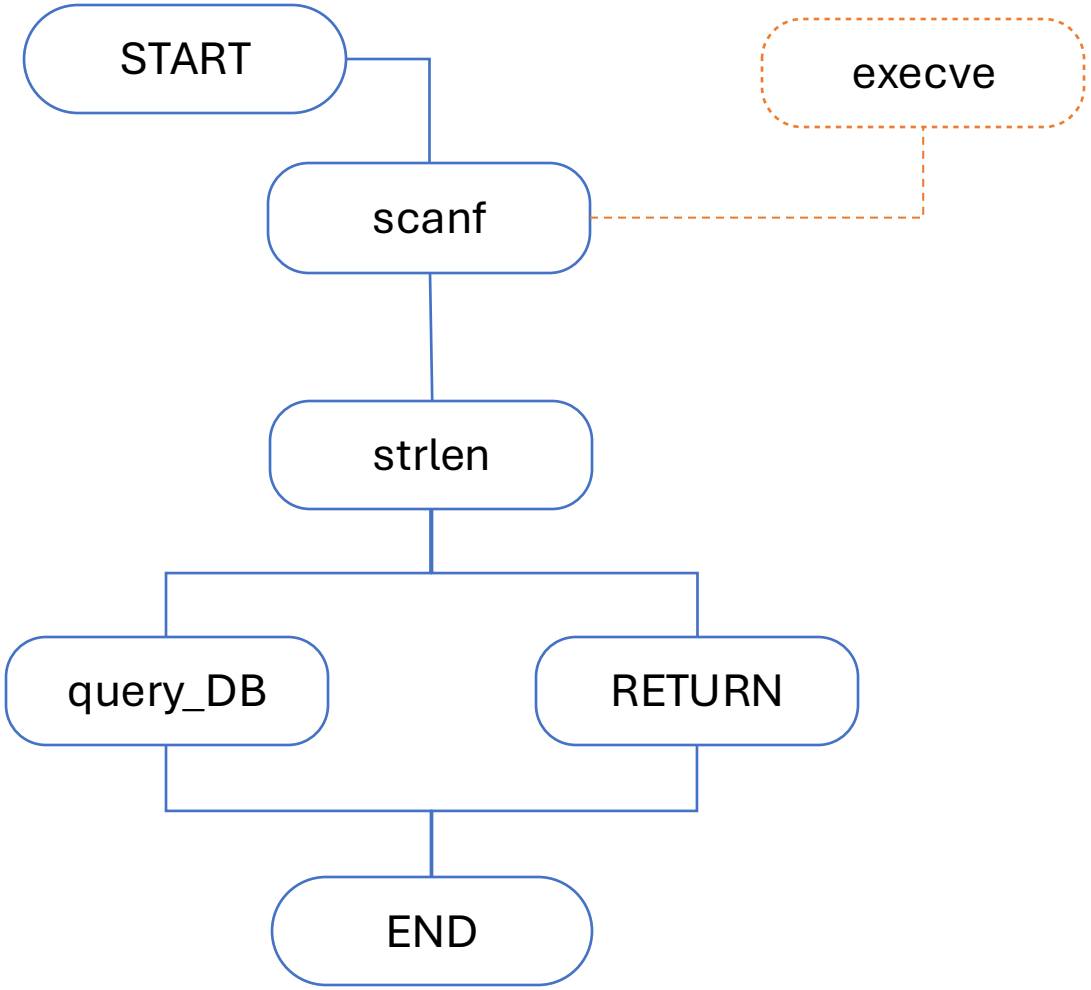
Can it be broken
or exploited?

Three Dimensions of Software Testing

What do we WANT to test here?

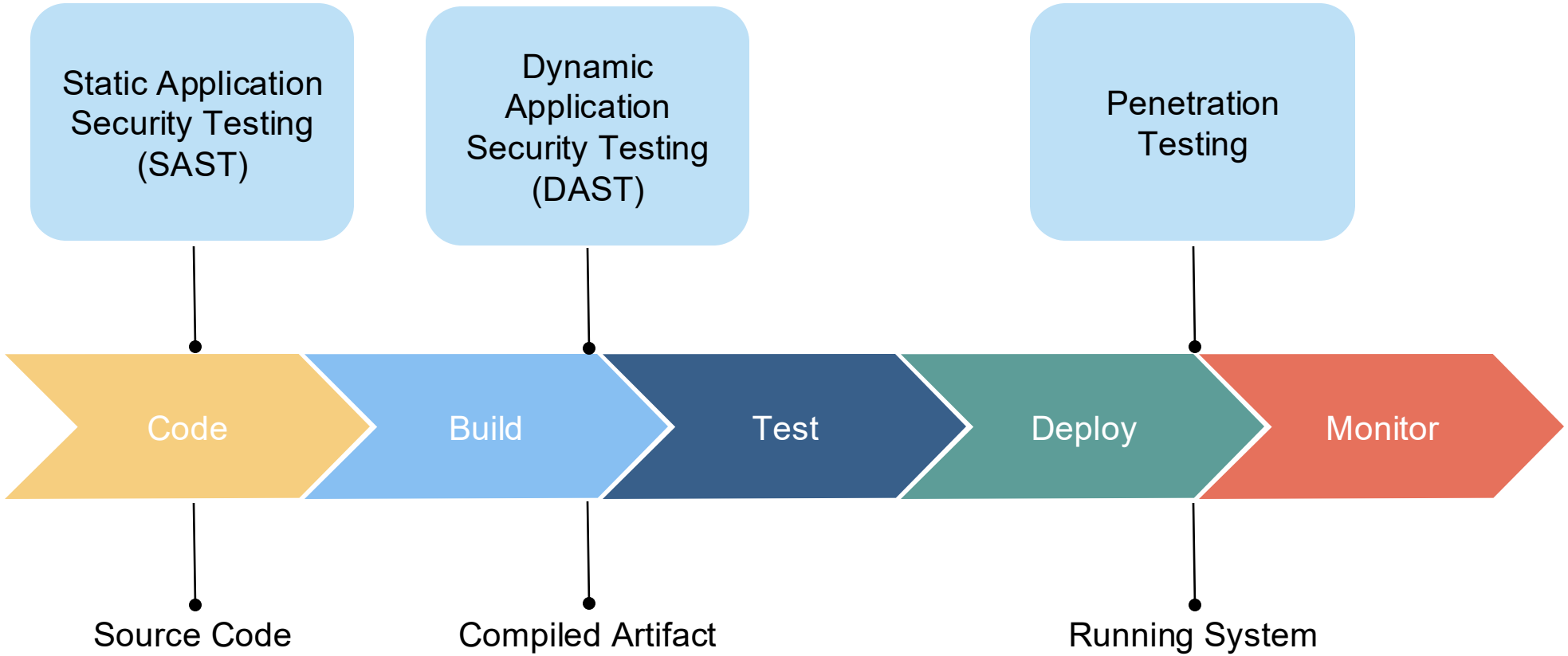
Security

Can it be broken or exploited?

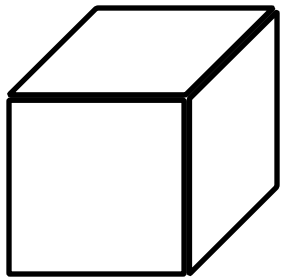


Security Testing

Tests depend on the availability that we have

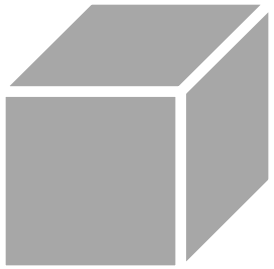


White-box, Grey-box and Black-box



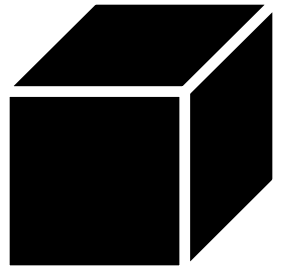
White-box testing

Full knowledge of source code and logic



Grey-box testing

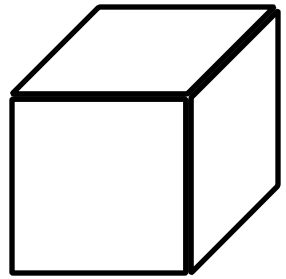
Partial knowledge — Internal structure



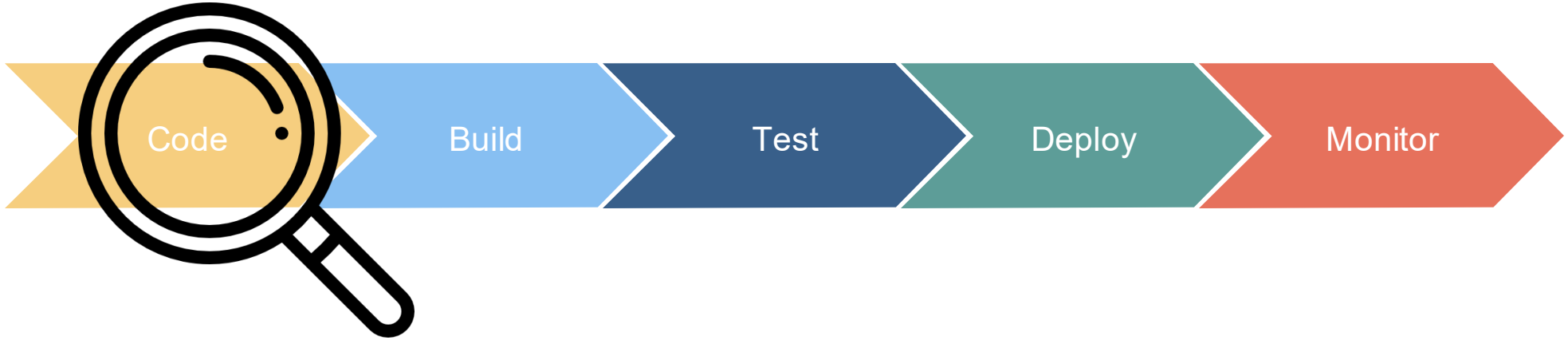
Black-box testing

No internal knowledge

White-box Testing



- Focuses on internal logic and code structure
- Uses source access to detect flaws early
- Ideal for developers and security reviewers



White-box Testing

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```

White-box Testing

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```

We don’t know how this function behaves, this could be dangerous

Example of Vulnerability

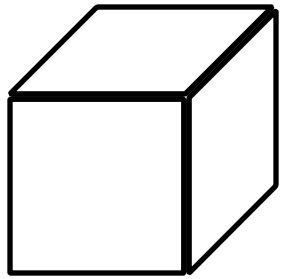
```c
int query_user_by_id(const char *db_name, const char *id_input) {
    sqlite3 *db;
    char *err_msg = NULL;
    int rc;
    char sql[1024];

    snprintf(sql, sizeof(sql), "SELECT id, name, email FROM users WHERE id = %s;", id_input);

    rc = sqlite3_open(db_name, &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        return rc;
    }

    printf("\n[Unsafe] Executing query:\n%s\n\n", sql);

    rc = sqlite3_exec(db, sql, print_callback, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", err_msg ? err_msg : "Unknown");
        sqlite3_free(err_msg);
    }

    sqlite3_close(db);
    return rc;
}
```

Example of Vulnerability

```c
int query_user_by_id(const char *db_name, const char *id_input) {
    sqlite3 *db;
    char *err_msg = NULL;
    int rc;
    char sql[1024];

    snprintf(sql, sizeof(sql), "SELECT id, name, email FROM users WHERE id = %s;", id_input);

    rc = sqlite3_open(db_name, &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        return rc;
    }

    printf("\n[Unsafe] Executing query:\n%s\n\n", sql);

    rc = sqlite3_exec(db, sql, print_callback, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", err_msg ? err_msg : "Unknown");
        sqlite3_free(err_msg);
    }

    sqlite3_close(db);
    return rc;
}
```

Manual inspection can be time consuming and error-prone

Pattern-matching



- Use rules that search for know-vulnerable patterns
- Rules can be complex with multiple patterns to match to understand if there is a vulnerability
- An attacker may use it to make a SQL injection and create a threat for the running application
- Pattern-matching may be not enough
    - It may rise false positives
    - It may miss vulnerabilities

Example of Vulnerability

snprintf(VAR1, VAR2, VAR3, VARS…)

====
VAR1: char[]
VAR2: int
VAR3: char*
VARS…: any
====

```
int query_user_by_id(const char *db_name, const char *id_input) {
    sqlite3 *db;
    char *err_msg = NULL;
    int rc;
    char sql[1024];

    snprintf(sql, sizeof(sql), "SELECT id, name, email FROM users WHERE id = %s;", id_input);

    rc = sqlite3_open(db_name, &db);
                      QLITE_OK) {
                      (stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
                      rc;


                      Unsafe] Executing query:\n%s\n\n", sql);

                      3_exec(db, sql, print_callback, 0, &err_msg);
                      QLITE_OK) {
                      (stderr, "SQL error: %s\n", err_msg ? err_msg : "Unknown");
                      _free(err_msg);


    sqlite3_close(db);
    return rc;
}
```

Example of Vulnerability

```c
int query_user_by_id(const char *db_name, const char *id_input) {
    sqlite3 *db;
    char *err_msg = NULL;
    int rc;
    char sql[1024];

    snprintf(sql, sizeof(sql), "SELECT id, name, email FROM users WHERE id = %s;", id_input);

    rc = sqlite3_open(db_name, &db);
    OK) {
    r, "Cannot open database: %s\n", sqlite3_errmsg(db));


    ] Executing query:\n%s\n\n", sql);

    (db, sql, print_callback, 0, &err_msg);
    OK) {
    r, "SQL error: %s\n", err_msg ? err_msg : "Unknown");
    err_msg);


    );
```

snprintf(VAR1, VAR2, VAR3, VARS…)

sanitizer: remove dangerous characters

====

VAR1: char[]

VAR2: int

VAR3: char*

VARS…: any

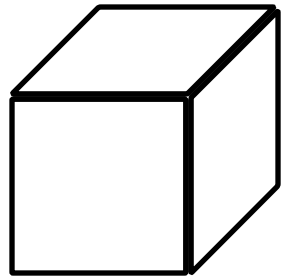
====

Example of Vulnerability

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empty.\n");
        return 1;
    }

    query_user_by_id(db_name, course_name);  // assuming your function handles string input
    return 0;
}
```

A function may be unused and however triggering the alert

Taint Analysis



- By **tainting** a variable we follow the data flow
- A variable may be tainted for various reasons
  - USER controlled
- If a tainted variable ends up in a vulnerable pattern an alarm is triggered
  - This means that a user may alter the execution flow by using a vulnerability

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

int main() {
    const char *db_name = "db";
    char course_name[100];

    printf("Enter Course Name: ");
    scanf("%s", course_name);

    // Check if the user entered something
    if (strlen(course_name) == 0) {
        printf("Error: Course name cannot be empt
        return 1;
    }

    query_user_by_id(db_name, course_name);  // a
    return 0;
}
```

SOURCE

```c
int query_user_by_id(const char *db_name, const char *id_input) {
    sqlite3 *db;
    char *err_msg = NULL;
    int rc;
    char sql[1024];

    snprintf(sql, sizeof(sql), "SELECT id, name, email FROM users WHERE id = %s;", id_input);

    rc = sqlite3_open(db_name, &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        return rc;
    }

    printf("\n[Unsafe] Executing query:\n%s\n\n", sql);

    rc = sqlite3_exec(db, sql, print_callback, 0, &err_msg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", err_msg ? err_msg : "Unknown");
        sqlite3_free(err_msg);
    }

    sqlite3_close(db);
    return rc;
}
```

SINK

Semgrep

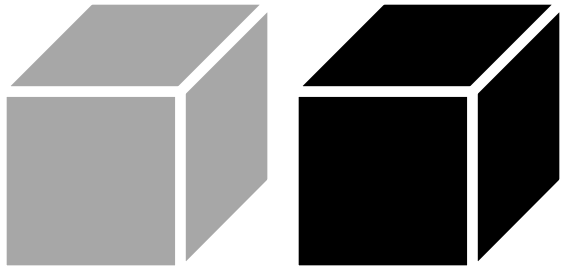
- Grep on Steroids
- It converts the program to a **intermediate representation**
- It extracts the **control flow graph**
- It identifies **vulnerable patterns** according to rules specified in the YAML format

https://semgrep.dev

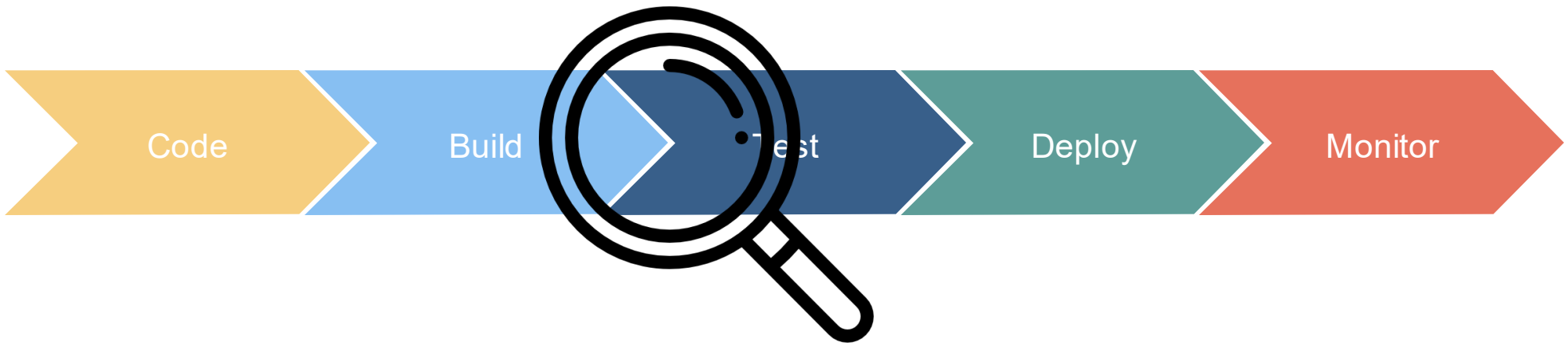
Hands-on (A Little Bit)

https://semgrep.dev/playground

Grey-Box and Black-box



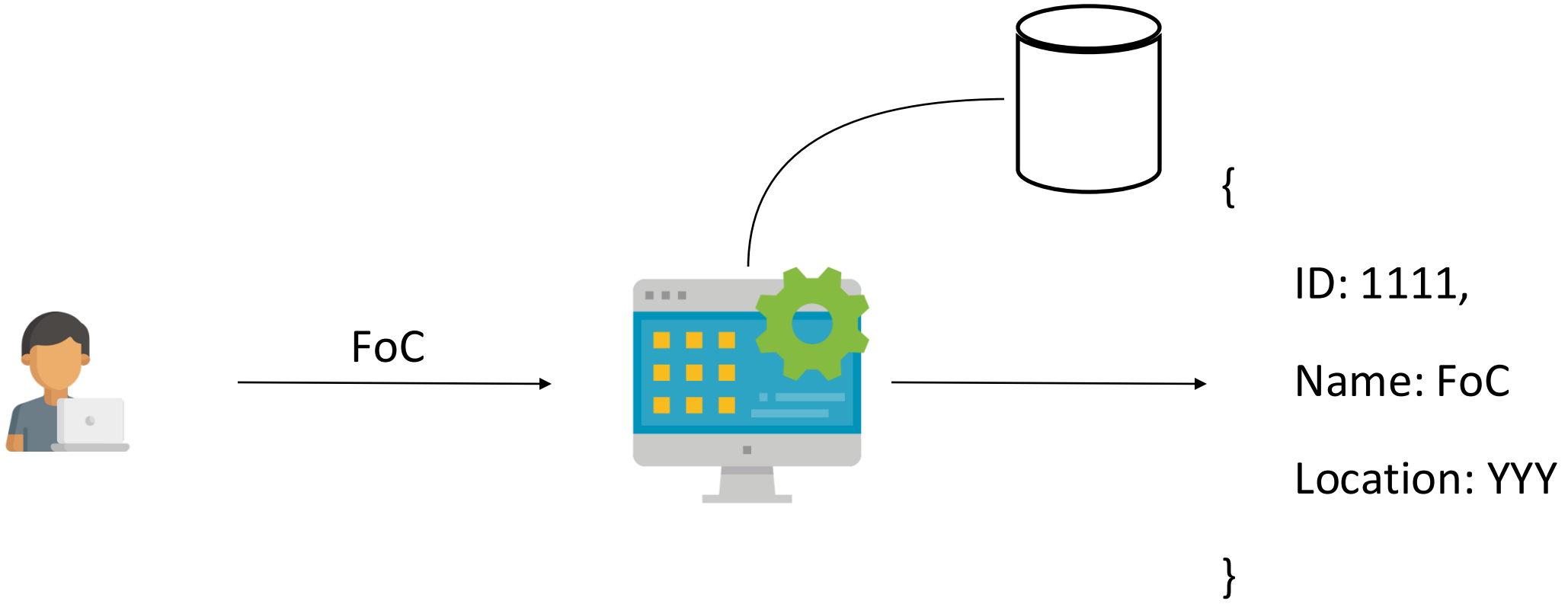
- Some vulnerabilities cannot be easily identified
- They are related to kernel faults such as memory corruption
- **Fuzz testing** is the key to discover this kind of issues



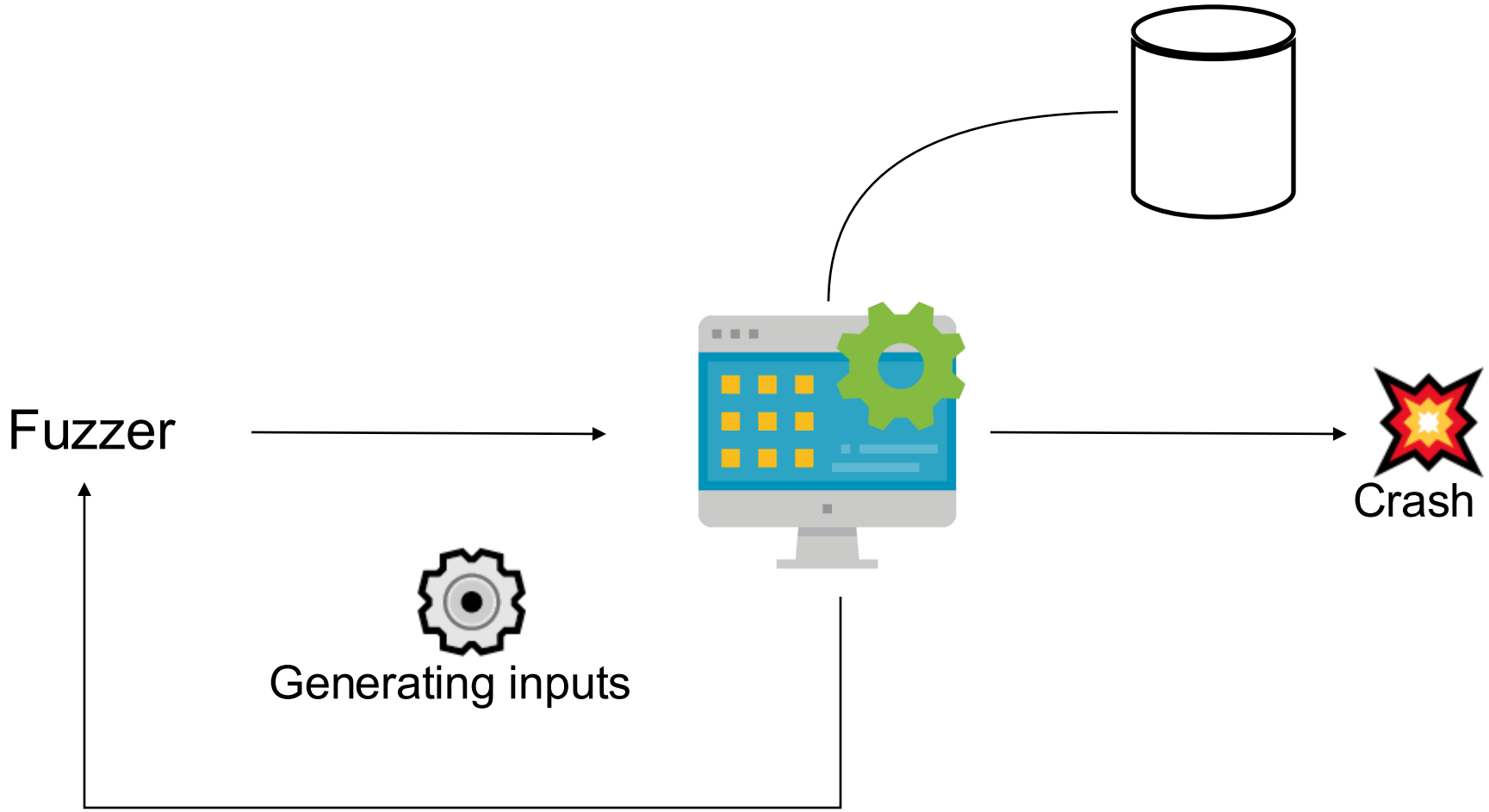
Fuzz Testing

- Tackling software complexity, fuzzing do not even try to be complete
- It is a **probabilistic testing technique**
  - You throw **random inputs** to a program as quickly as you can trying to trigger a **crash**
  - When a crash happen, you store the input

Fuzz Testing



Fuzz Testing



Fuzz Testing

./main test

Test Success

./main AAAAAA

Test Failure

cat /dev/urandom | ./main

==61==ERROR: AddressSanitizer: unknown-crash on address …

Fuzz Testing

cat /dev/urandom | ./main

==61==ERROR: AddressSanitizer: unknown-crash on address …

Testing all possible cases requires a huge amount of inputs

A three-part Tale

Poet

Courier

Oracle

Poet(s)

Types of fuzzers

- Random
- Mutational
- Generational
- Evolutionary

Random

- Give totally random inputs to the target program
    - Piping /dev/urandom
- Hooking the getenv()
- How do you identify the crash?

Mutational

- Collect initial samples and modify them
- It requires less setup time, since you are giving the initial input
- It works best with file formats and clear-text protocols

- It is defined as *dumb fuzzing* since it has no knowledge about the input format
  - It simply appends or modifies bytes of the input

Generational

- Models what the target program should take as input
- The input model is defined by stating its components
- Then during the fuzzing campaign the input is modified according to its definition

Evolutionary

- Applies evolutionary models to the generation of inputs
- An evolutionary model evaluates the output of the software under test when fed with the test case
- This kind of fuzzers are strongly investigated

https://fandango-fuzzer.github.io

Courier

- The fuzzer needs to communicate with the target program
- Depending on what we are fuzzing this activity may differ
  - Fuzzing through network communications
  - Fuzzing APIs
  - Fuzzing User Interfaces

Oracle(s)

How crashes are detected

- We need to know when an application crashes and what kind of crash happened
  - memory errors such as heap and stack buffer overflows, use-after-free, and memory leaks
  - use of uninitialized memory
  - undefined behavior, such as integer overflows, invalid shifts, and negative array indices

Sanitizers

Sanitizers

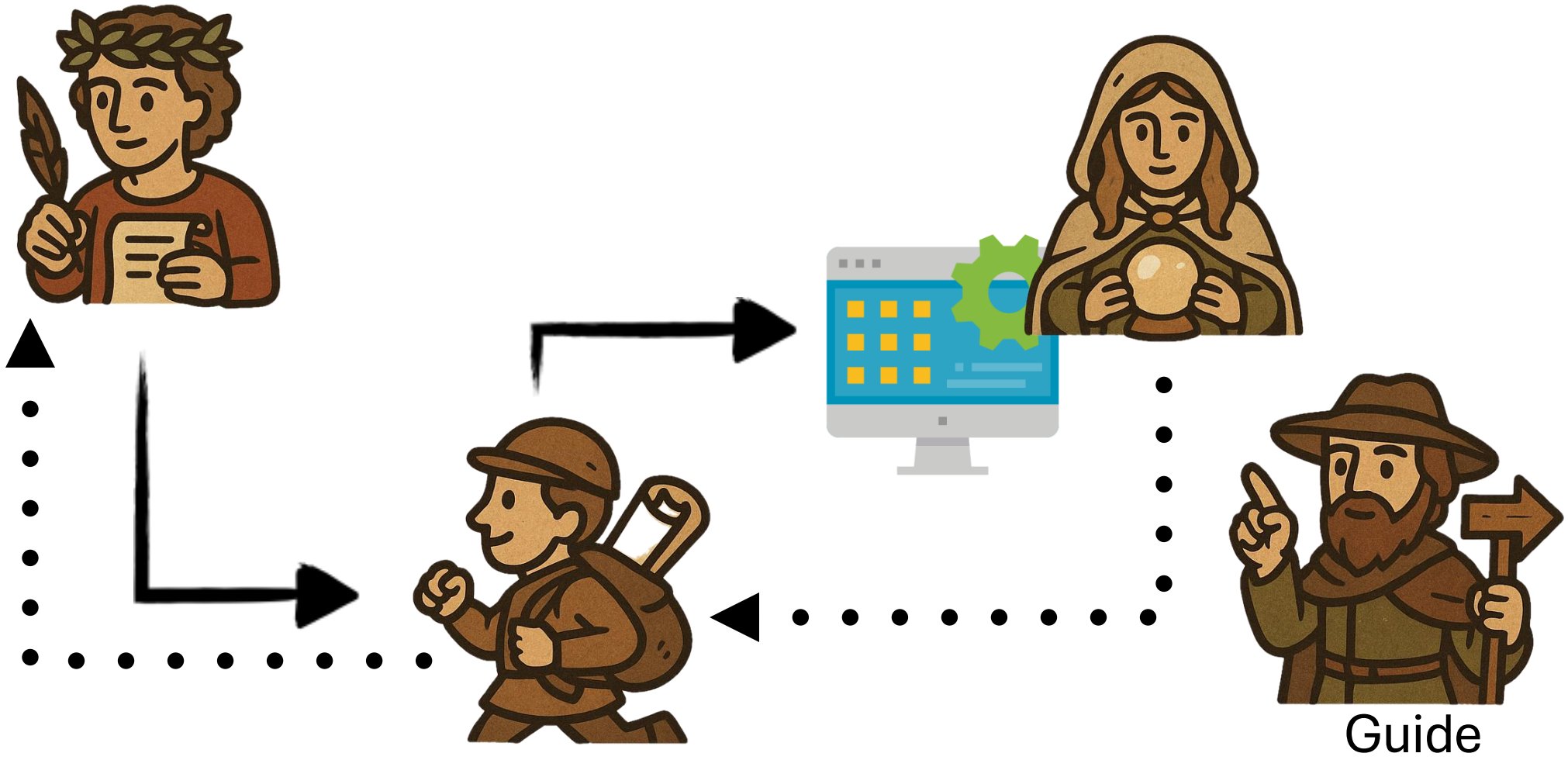
- Depend on what we want to detect
- They are inserted by the compiler to detect specific crashes
- AddressSanitizer (ASan)
  - it detects when a memory corruption happens
- Memory Sanitizer (MSan)
  - It detects the use of initialized memory
- Undefined Behaviour Sanitizier (UBSan)
  - It detects a wide range of events that are not directly linked to memory corruption

```
=================================================================
==1==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x502000000014 a
0x5591ad37f523 bp 0x7ffe6acc8e70 sp 0x7ffe6acc8e68
READ of size 1 at 0x502000000014 thread T0
    #0 0x5591ad37f522 in out_of_bounds(char const*) /app/example.cpp:5:45
    #1 0x5591ad37f59a in main /app/example.cpp:10:4
    #2 0x7f8882a29d8f  (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId:
c289da5071a3399de893d2af81d6a30c62646e1e)
    #3 0x7f8882a29e3f in __libc_start_main
(/lib/x86_64-linux-gnu/libc.so.6+0x29e3f) (BuildId:
c289da5071a3399de893d2af81d6a30c62646e1e)
    #4 0x5591ad2a4324 in _start (/app/output.s+0x2c324)
```

Other oracles…

- There are many other oracles that we can exploit
  - Monitoring resource consumption
  - Input validation through output analysis
  - Network traffic analysis
  - Whatever it makes sense to identify an unexpected behavior

To Recap



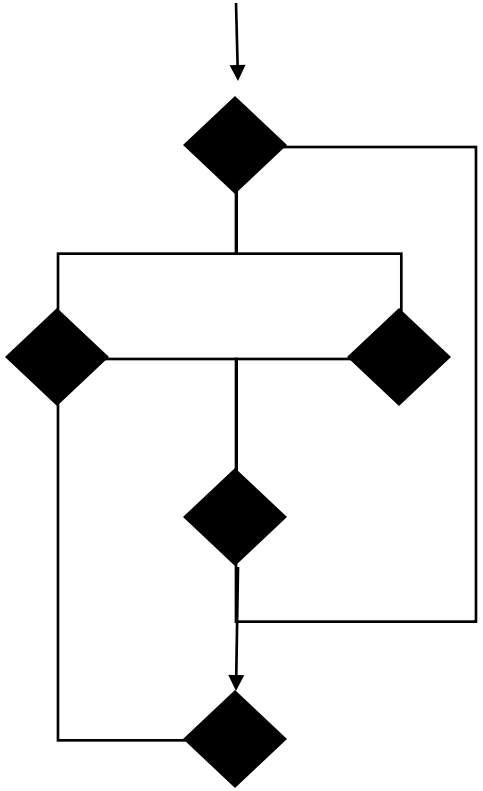
Program Coverage

An unexpected player… the guide

- The courier wants to travel for unexplored paths
- It needs to know how to use what the poet produced
- The guide has the role to map the paths of the program
- We need to know when a test case enables the exploration of a previously unseen execution flow
- The guide communicates to the poet in order to have better test cases

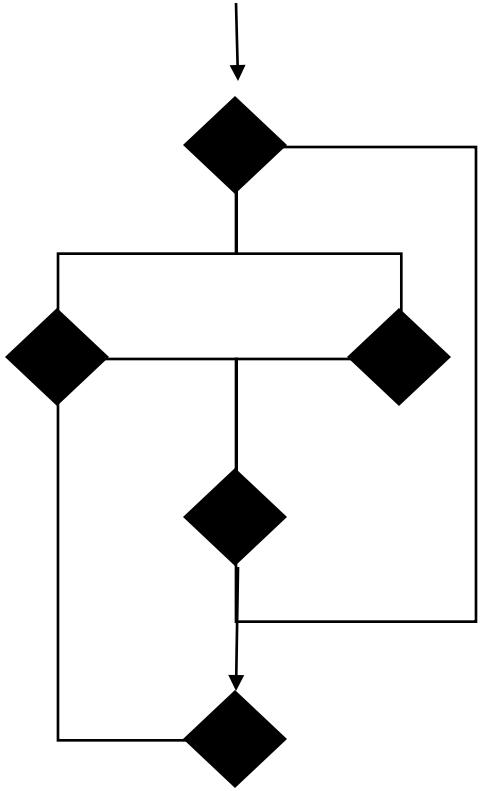
Coverage

- Coverage measures executed program area
  - Function coverage
  - Block coverage
  - Line/statement coverage
  - Edge coverage
  - Path coverage (too complex → requires equivalence check)
  - Data flow coverage (expensive)



Coverage

- Coverage measures executed program area
  - Function coverage
  - Block coverage
  - Line/statement coverage
  - **Edge coverage**
  - Path coverage (too complex → requires equivalence check)
  - Data flow coverage (expensive)

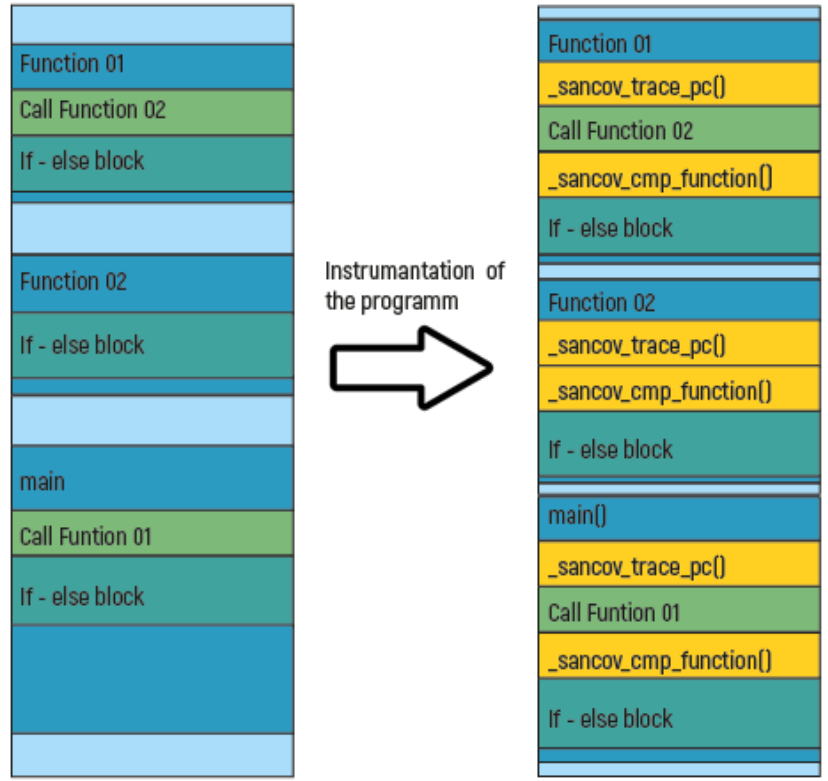


Obtaining Coverage

- We need to collect information on what a test case covered in the program
- Depending on the fuzzer there are different techniques
  - Make the target program log information (instrumentation)
  - Observe external elements (resources, filesystem, environment)

Program Instrumentation

- The compiler inserts tracing functions in specific places of the compiled program
- When the execution hits a tracing function the coverage is updated
- Depending on the increase in coverage the input is modified



Instrumentation

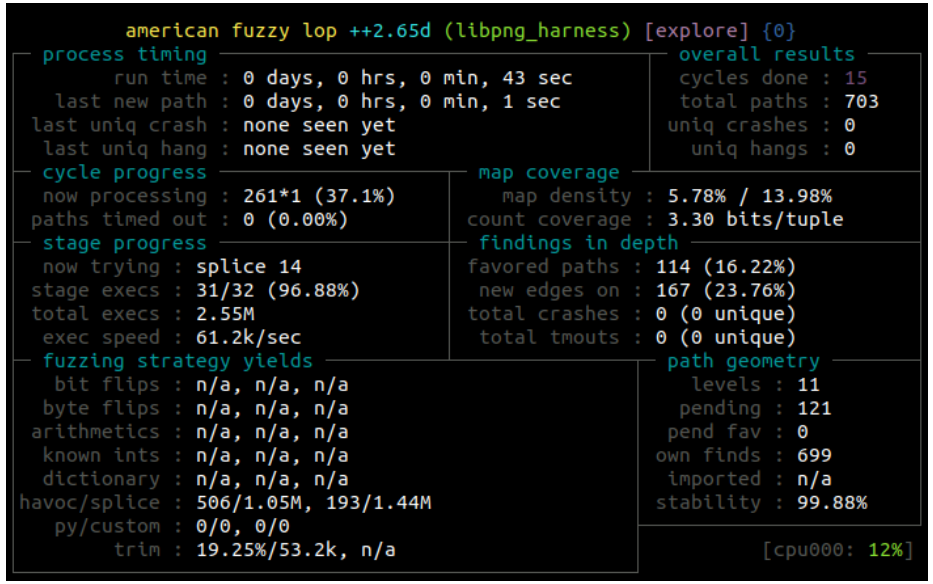
- A Fuzzer is able to track coverage by instrumenting the program
- The Instrumentation technique may vary depending on the fuzzer
- AFL instruments each basic block with a function: __afl_maybe_log
- When the execution hits this function the coverage is stored

```
...
mov     qword ptr [rsp+98h+input+0F88h], rcx
mov     qword ptr [rsp+98h+input+0F90h], rax ; save registers
mov     rcx, 0FC50h ; rcx is the block identifier, generated r
call    __afl_maybe_log
mov     rax, qword ptr [rsp+98h+input+0F90h]
mov     rcx, qword ptr [rsp+98h+input+0F88h]
...
```

A real-world Fuzzer

American Fuzzing Lop ++

- AFL++ is one of the standards for fuzzing
- The framework exposes multiple mutators and configurations
- Instrumentation models for source code and binary
- Test cases and corpus minimization utilities

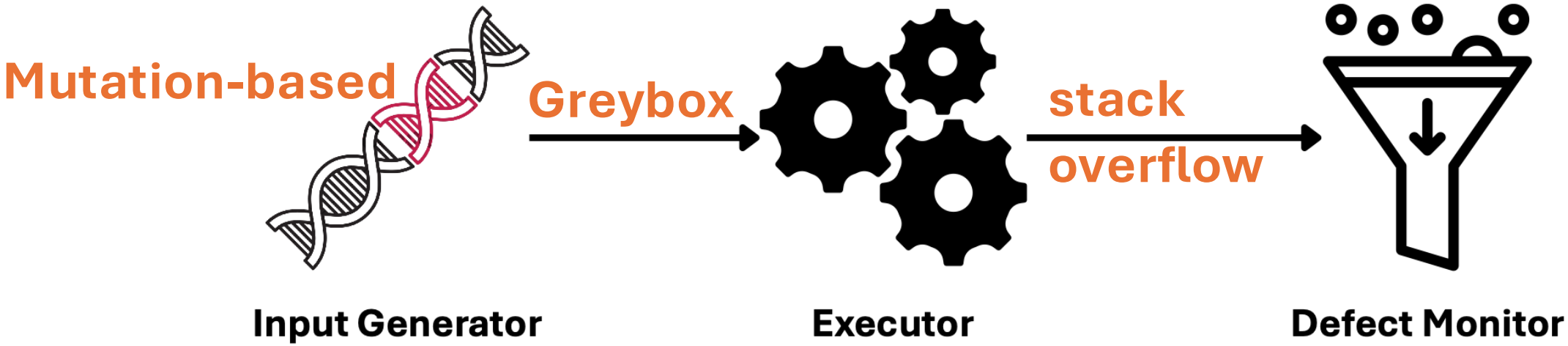


Lot of techniques... Lot of Fuzzers

A simple example…

Tutorial: Fuzzing xpdf

- Step 1: getting the AFL++ fuzzer and some sample inputs
- Step 1.b: install afl++ ()
- Step 2: instrumenting the target program
- Step 3: running afl++ against the instrumented program



Tutorial: Fuzzing xpdf

- Step 1: getting the program…

wget https://dl.xpdfreader.com/old/xpdf-3.02.tar.gz
tar -xvzf xpdf-3.02.tar.gz

…and get some sample inputs

cd $HOME/fuzzing_xpdf mkdir pdf_examples && cd pdf_examples wget https://github.com/mozilla/pdf.js-sample-files/raw/master/helloworld.pdf wget http://www.africau.edu/images/default/sample.pdf wget https://www.melbpc.org.au/wp-content/uploads/2017/10/small-example-pdf-file.pdf

Tutorial: Fuzzing xpdf

- Step 1.b: install afl++ ()

  Install dependencies

  ```
  sudo apt-get update
  sudo apt-get install -y build-essential python3-dev automake git flex bison libglib2.0-dev libpixman-1-dev python3-setuptools
  sudo apt-get install -y lld-11 llvm-11 llvm-11-dev clang-11 || sudo apt-get install -y lld llvm llvm-dev clang
  sudo apt-get install -y gcc-$(gcc --version|head -n1|sed 's/.* //'|sed 's/\..*//')-plugin-dev libstdc++-$(gcc --version|head -n1|sed 's/.* //'|sed 's/\..*//')-dev
  ```

Tutorial: Fuzzing xpdf

- Step 1.b: install afl++ ()

  Install afl

  ```
  cd $HOME
  git clone https://github.com/AFLplusplus/AFLplusplus && cd AFLplusplus
  export LLVM_CONFIG="llvm-config-11"
  make distrib
  sudo make install
  ```

  Check that the fuzzer is installed

  ```
  afl-fuzz
  ```

Tutorial: Fuzzing xpdf

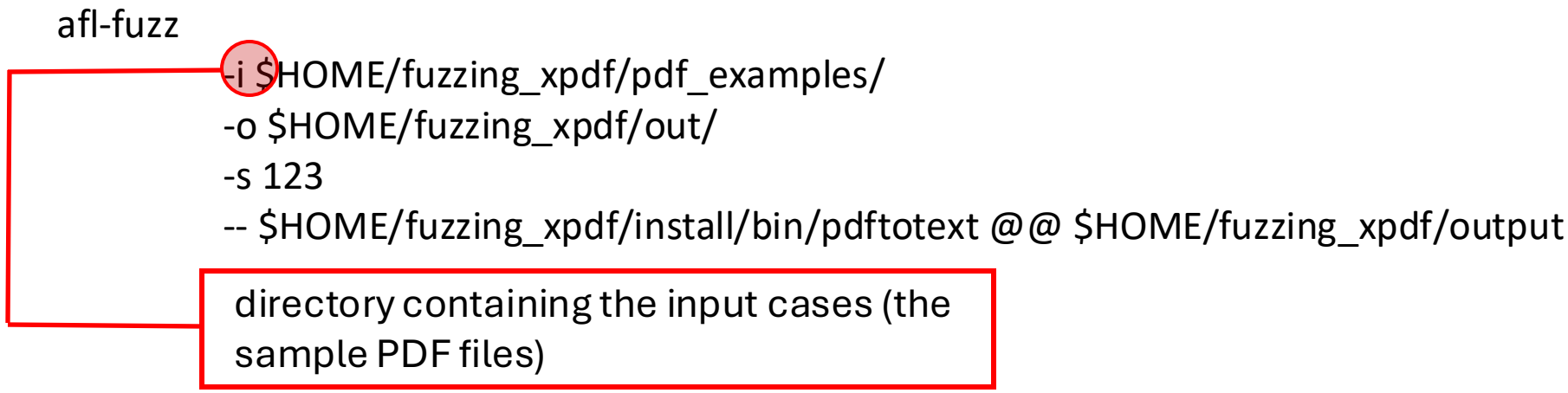
- Step 2: instrumenting the program

  We use afl-clang-fast: one of the compiler provided from this fuzzer to instrument a binary

```
rm -r $HOME/fuzzing_xpdf/install
cd $HOME/fuzzing_xpdf/xpdf-3.02/

export LLVM_CONFIG="llvm-config-11"
CC=$HOME/AFLplusplus/afl-clang-fast CXX=$HOME/AFLplusplus/afl-clang-fast++ ./configure --
prefix="$HOME/fuzzing_xpdf/install/"
make
make install
```

Tutorial: Fuzzing xpdf

- Step 3: running afl++ against the instrumented program



Tutorial: Fuzzing xpdf

- Step 3: running afl++ against the instrumented program

```
afl-fuzz
        -i $HOME/fuzzing_xpdf/pdf_examples/
        -o $HOME/fuzzing_xpdf/out/
        -s 123
        -- $HOME/fuzzing_xpdf/install/bin/pdftotext @@ $HOME/fuzzing_xpdf/output
```

directory where the mutate files will be stored

Tutorial: Fuzzing xpdf

- Step 3: running afl++ against the instrumented program

```
afl-fuzz
        -i $HOME/fuzzing_xpdf/pdf_examples/
        -o $HOME/fuzzing_xpdf/out/
        -s 123
        -- $HOME/fuzzing_xpdf/install/bin/pdftotext @@ $HOME/fuzzing_xpdf/output
```

The static random seed to use. The files will be mutated accordingly to this seed.

Tutorial: Fuzzing xpdf

- Step 3: running afl++ against the instrumented program

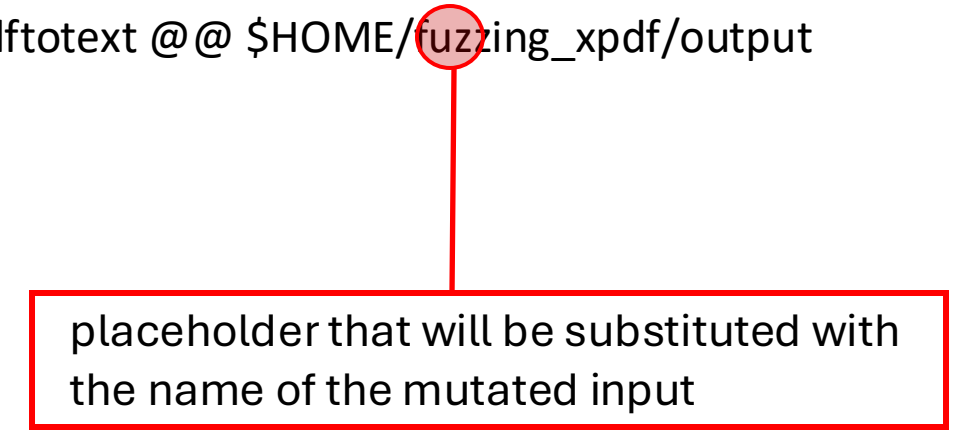
```
afl-fuzz
        -i $HOME/fuzzing_xpdf/pdf_examples/
        -o $HOME/fuzzing_xpdf/out/
        -s 123
        -- $HOME/fuzzing_xpdf/install/bin/pdftotext @@ $HOME/fuzzing_xpdf/output
```

The instrumented binary that is our program under test

Tutorial: Fuzzing xpdf

- Step 3: running afl++ against the instrumented program

```
afl-fuzz
        -i $HOME/fuzzing_xpdf/pdf_examples/
        -o $HOME/fuzzing_xpdf/out/
        -s 123
        -- $HOME/fuzzing_xpdf/install/bin/pdftotext @@ $HOME/fuzzing_xpdf/output
```



Crash Collection

- We found crashes
- Now we need to investigate the cause of the crash by using the inputs generated by the fuzzer
- Let’s try feeding the program with the generated input

```
[AFL++ 7a395683db24] /home/fuzzing_xpdf # install/bin/pdftotext "/home/fuzzing_xpdf/out/default/crashes/id:000000,sig:11,src:000807,time:215127,execs:179946,op:havoc,rep:3" output
Error (3619): Missing 'endstream'
Segmentation fault
```

```
#0  0x0000ffffff7abe900 in _IO_free_backup_area () from /lib/aarch64-linux-gnu/libc.so.6
#1  0x0000ffffff7abd218 in _IO_file_seekoff () from /lib/aarch64-linux-gnu/libc.so.6
#2  0x0000ffffff7aba1a0 in fseeko64 () from /lib/aarch64-linux-gnu/libc.so.6
#3  0x0000aaaaaab80f88 in FileStream::reset (this=0xaaaaac7a16a0) at Stream.cc:602
#4  0x0000aaaaaab6a474 in Object::streamReset (this=<optimized out>) at ./Object.h:280
#5  Lexer::Lexer (this=0xaaaaac7a15f0, xref=<optimized out>, str=<optimized out>) at Lexer.cc:57
#6  0x0000aaaaaabb0ee8 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800200) at XRef.cc:808
#7  0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800390,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800200) at ./Object.h:253
#8  Parser::makeStream (this=this@entry=0xaaaaac7a10c0, dict=dict@entry=0xffffff800390, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#9  0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a10c0, obj=0xffffff800390, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#10 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800390) at XRef.cc:823
#11 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800520,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800390) at ./Object.h:253
#12 Parser::makeStream (this=this@entry=0xaaaaac7a0be0, dict=dict@entry=0xffffff800520, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#13 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0be0, obj=0xffffff800520, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#14 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800520) at XRef.cc:823
#15 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff8006b0,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800520) at ./Object.h:253
#16 Parser::makeStream (this=this@entry=0xaaaaac7a0700, dict=dict@entry=0xffffff8006b0, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#17 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0700, obj=0xffffff8006b0, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#18 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff8006b0) at XRef.cc:823
#19 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800840,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f8006b0) at ./Object.h:253
#20 Parser::makeStream (this=this@entry=0xaaaaac7a0220, dict=dict@entry=0xffffff800840, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#21 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0220, obj=0xffffff800840, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#22 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800840) at XRef.cc:823
#23 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff8009d0,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800840) at ./Object.h:253
#24 Parser::makeStream (this=this@entry=0xaaaaac79fd40, dict=dict@entry=0xffffff8009d0, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#25 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac79fd40, obj=0xffffff8009d0, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#26 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff8009d0) at XRef.cc:823
#27 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800b60,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
```

```
#0  0x0000fffff7abe900 in _IO_free_backup_area () from /lib/aarch64-linux-gnu/libc.so.6
#1  0x0000fffff7abd218 in _IO_file_seekoff () from /lib/aarch64-linux-gnu/libc.so.6
#2  0x0000fffff7aba1a0 in fseeko64 () from /lib/aarch64-linux-gnu/libc.so.6
#3  0x0000aaaaaab80f88 in FileStream::reset (this=0xaaaaac7a16a0) at Stream.cc:602
#4  0x0000aaaaaab6a474 in Object::streamReset (this=<optimized out>) at ./Object.h:280
#5  Lexer::Lexer (this=0xaaaaac7a15f0, xref=<optimized out>, str=<optimized out>) at Lexer.cc:57
#6  0x0000aaaaaabb0ee8 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800200) at XRef.cc:808
#7  0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800390,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800200) at ./Object.h:253
#8  Parser::makeStream (this=this@entry=0xaaaaac7a10c0, dict=dict@entry=0xffffff800390, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#9  0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a10c0, obj=0xffffff800390, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#10 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800390) at XRef.cc:823
#11 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800520,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800390) at ./Object.h:253
#12 Parser::makeStream (this=this@entry=0xaaaaac7a0be0, dict=dict@entry=0xffffff800520, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#13 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0be0, obj=0xffffff800520, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#14 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800520) at XRef.cc:823
#15 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff8006b0,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800520) at ./Object.h:253
#16 Parser::makeStream (this=this@entry=0xaaaaac7a0700, dict=dict@entry=0xffffff8006b0, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#17 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0700, obj=0xffffff8006b0, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#18 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff8006b0) at XRef.cc:823
#19 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800840,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f8006b0) at ./Object.h:253
#20 Parser::makeStream (this=this@entry=0xaaaaac7a0220, dict=dict@entry=0xffffff800840, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#21 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac7a0220, obj=0xffffff800840, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#22 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff800840) at XRef.cc:823
#23 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff8009d0,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
f800840) at ./Object.h:253
#24 Parser::makeStream (this=this@entry=0xaaaaac79fd40, dict=dict@entry=0xffffff8009d0, fileKey=fileKey@entry=0x0, encAlgorithm=encAlgorithm@entry=cryptRC4,
    keyLength=0, objNum=7, objGen=objGen@entry=0) at Parser.cc:156
#25 0x0000aaaaaab794e8 in Parser::getObj (this=0xaaaaac79fd40, obj=0xffffff8009d0, fileKey=0x0, encAlgorithm=cryptRC4, keyLength=0, objNum=7, objGen=0) at Parser.cc:94
#26 0x0000aaaaaabb1378 in XRef::fetch (this=0xaaaaaae99630, num=7, gen=0, obj=0xffffff8009d0) at XRef.cc:823
#27 0x0000aaaaaab79cd0 in Object::dictLookup (this=0xffffff800b60,
    key=0xaaaaaae997d0 "%PDF-1.4\n%\307\354\217\242\n4 0 obj\n<</Linearized 1/L 4107/H[ 1306 137]/O 6/E 1306/N 1/T 3986>>\nendobj\n", ' ' <repeats 55 times>, "\nxref\n4 10\
```

The Bug

- Comparing the code of xpdf@3.02 with xpdf@4.02
- The getObj function confirms that the issue was linked to an infinite recursion
- An attacker can exploit it for a Denial of Service attack

```
-Object *Parser::getObj(Object *obj, Guchar *fileKey,
+Object *Parser::getObj(Object *obj, GBool simpleOnly,
+                       Guchar *fileKey,
                        CryptAlgorithm encAlgorithm, int keyLength,
-                       int objNum, int objGen) {
+                       int objNum, int objGen, int recursion) {
```

Concluding

- Software testing is essential to produce, distribute, and consume secure applications
- The kind of test depends on the setting you are in
- Testing techniques are often complementary, one is not enough

Thanks for your attention!

Giacomo Benedetti

Institute for Applied Mathematics and Information Technologies

National Research Council of Italy

giacomo.benedetti@ge.imati.cnr.it

giacomobenedetti.github.io