Fuzz & Furious

Accelerating Smart Contract Security with Cairo-Fuzzer

FUZZING LABS

Summary

- Fuzzing Labs
- Introduction to Fuzzing
- What is Cairo-Fuzzer
  - Installing Cairo-Fuzzer
  - Understanding the output of cairo-fuzzer
  - Cairo-Fuzzer CheatSheet
  - Property Testing using Cairo-Fuzzer
  - Fuzzing using a dictionary
- Testing
- Why Cairo-Fuzzer does not support Cairo 1.0
- Future of Cairo-Fuzzer

FuzzingLabs

Nabih Benazzouz (@Raefko)

- Security engineer @**FuzzingLabs** | Junior **Security Researcher**
  - Fuzzing and vulnerability research
  - Development of security tools
  - Worked on **Browser Fuzzing** training during my internship
  - Contact : **Nabih@fuzzinglabs.com**

- Main focus
  - **Fuzzing** and **Auditing** of Blockchain VM and tools
  - Rust, Golang, Python, C/C++

- Background
  - SRS 2022 (sécurité, réseau et système) @**EPITA**
  - LSE Student and **Researcher** from 2020 to 2022
  - CTF profile :
    - Root-me
    - CTF LSE

Patrick Ventuzelo (@Pat_Ventuzelo)

- Founder & CEO of **FuzzingLabs** | Senior **Security Researcher**
  - Fuzzing and vulnerability research
  - Development of security tools
  - Contact : **Patrick@fuzzinglabs.com**

- Training/Online courses
  - **Youtube** channel - Link
  - **Rust** Security Audit & Fuzzing
  - **Go** Security Audit & Fuzzing
  - **WebAssembly** Reversing & Analysis
  - Practical Web **Browser** Fuzzing

- Main focus
  - **Fuzzing**, Vulnerability research
  - Rust, Golang, **WebAssembly**, **Browsers**
  - Blockchain Security, Smart contracts

- Previous speaker at:
  - BlackHat US, OffensiveCon, REcon, RingZer0, ToorCon, hack.lu, NorthSec, etc.

FuzzingLabs

- Research and development (R&D)
- Audit & Consulting
- Development of tools and fuzzers
- Setting up of training courses in relation with the security fields
- Presentations at conferences
- Strong interest in blockchain and its security
- Youtube tutorials

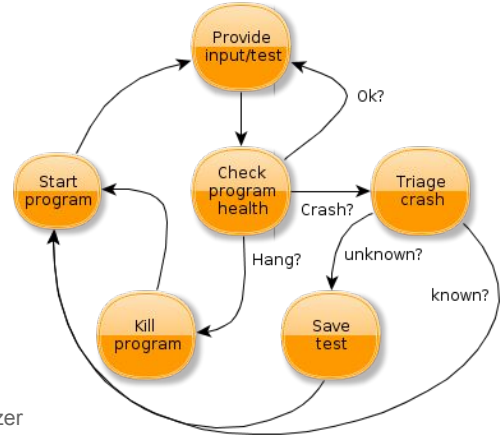
Introduction to Fuzzing

What's fuzzing?

- Fuzzing or fuzz testing is an **automated software testing technique** that involves providing invalid, unexpected, or **random data as inputs** to a computer program. The program is then **monitored for exceptions** such as crashes, failing built-in code assertions, or for finding potential memory leaks and other unexpected behaviors - link

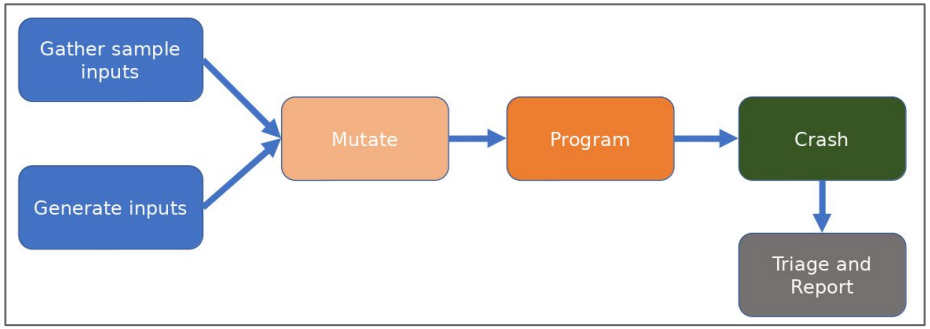
- **The most efficient technique to find bugs!**

- Different fuzzing approaches:
  - **Black box**:
    - You don't have any real knowledge of the target
    - You don't have access to the source code
    - You are not able to recompile the target.
  - **Gray box**:
    - You have some knowledge of the target
    - You are not able to recompile the target.
  - **White box**:
    - You have access to the source code
    - You can recompile the target.



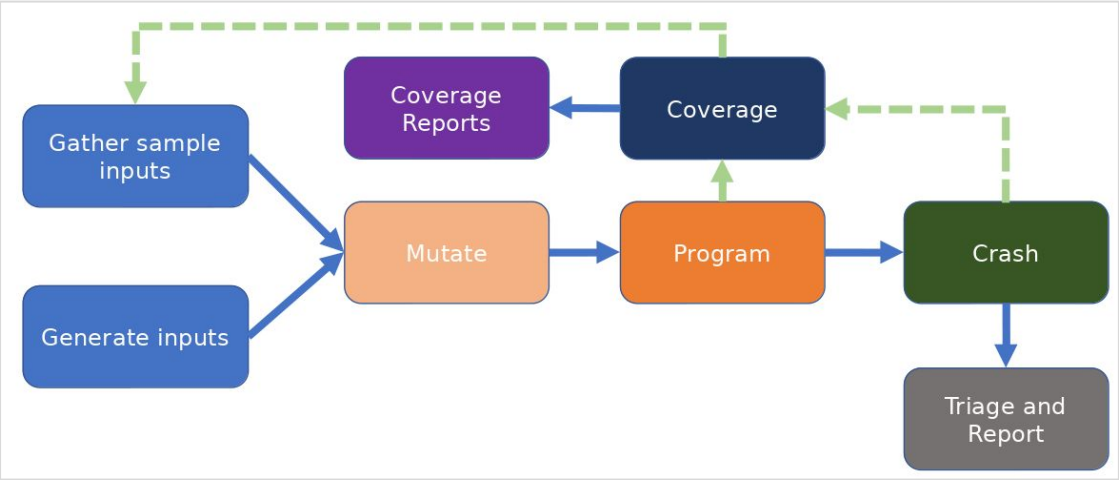
Fuzzing techniques #1 - Really basic

- **Dumb** fuzzing
  - Input data is corrupted randomly without awareness of the expected format.
    - cat /dev/urandom | mytarget
- **Smart** fuzzing
  - Input data is corrupted with awareness of the expected format, such as encodings, relations (offset, checksum, etc).
- **Mutation-based** fuzzing
  - Modification of known-valid input data is made according to certain patterns.



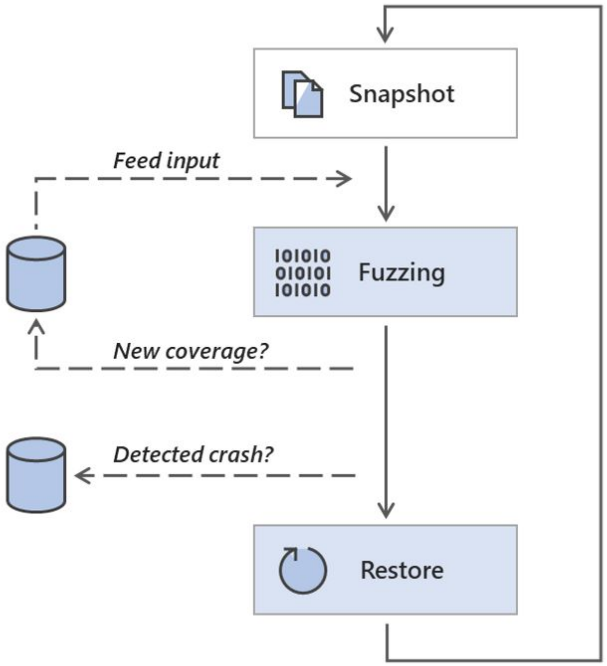
Fuzzing techniques #2 - Most common techniques

- **Feedback-driven / Coverage-guided** fuzzing
  - Observe how inputs are processed to learn which mutations are interesting.
  - Save those inputs to be re-used in future iterations.



Fuzzing techniques #3 - Advanced

- **In-Process/In-memory/Persistent** fuzzing
  - Target and fuzz a specific function entry point of the program in only one process i.e., for every test case the process isn't restarted but the values are changed in memory.
- **Generation-based** fuzzing
  - Generate semi-well-formed inputs from scratch, based on knowledge of file format or protocol.
- **Differential** fuzzing
  - Observe if two program implementations/variants produce different outputs for the same input.
- **Snapshot** fuzzing
  - In-Process fuzzing with previous memory/register state restored for each fuzz case



What is Cairo-Fuzzer

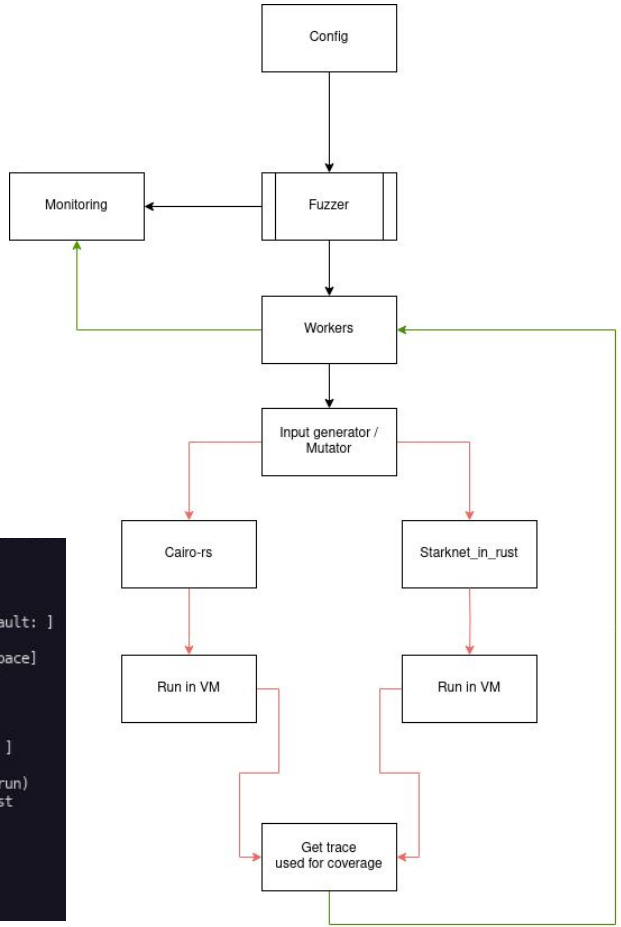
Cairo-Fuzzer - Architecture

- **Architecture**
  - Coverage-guided
  - Multithreaded with good scalability
    - 70k exec/s for 1 thread
    - 440k exec/s for 10 threads
  - Execution engines
    - lambdaclass/cairo-vm for Cairo contract
    - lambdaclass/starknet_in_rust for StarkNet contract
  - Usable as a library

- **Features**
  - Property testing
  - Minimizer
  - Replayer
  - Usage of dictionary

```
Usage: cairo-fuzzer [OPTIONS]

Options:
      --cores <CORES>              Set the number of threads to run [default: 1]
      --contract <CONTRACT>        Set the path of the JSON artifact to load [default: ]
      --function <FUNCTION>        Set the function to fuzz [default: ]
      --workspace <WORKSPACE>      Workspace of the fuzzer [default: fuzzer_workspace]
      --inputfolder <INPUTFOLDER>  Path to the inputs folder to load [default: ]
      --crashfolder <CRASHFOLDER>  Path to the crashes folder to load [default: ]
      --inputfile <INPUTFILE>      Path to the inputs file to load [default: ]
      --crashfile <CRASHFILE>      Path to the crashes file to load [default: ]
      --dict <DICT>                Path to the dictionnary file to load [default: ]
      --logs                       Enable fuzzer logs in file
      --seed <SEED>                Set a custom seed (only applicable for 1 core run)
      --run-time <RUN_TIME>        Number of seconds this fuzzing session will last
      --config <CONFIG>            Load config file
      --replay                     Replay the corpus folder
      --minimizer                  Minimize Corpora
      --proptesting                Property Testing
      --iter <ITER>                Iteration Number [default: -1]
  -h, --help                       Print help
```



Cairo-Fuzzer - Example

- 12 cores
- 460k exec/seconds
- [demo]

```
→  cairo-fuzzer git:(update_03_07_2023) ✗ cargo run --release -- --cores 12 --contract tests/fuzzinglabs.json --function "Fuzz_sym
   Finished release [optimized] target(s) in 0.18s
    Running `target/release/cairo-fuzzer --cores 12 --contract tests/fuzzinglabs.json --function Fuzz_symbolic_execution`
      ==========================================================================================================================

                 _____   _____   _______   _____   _____        _____   _____        _____   _____   _____   _____
                ( ___ \(  ___  )\__   __/( ___ )( ___ )      ( ___ \|\     /|/ ___  )/ ___  )( ___ \( ___ )
                | (   \/| (   ) |   ) (   | (   )|| (   ) |      | (   \/| )   ( |\/  )  |\/  )  || (   \/| (   )|
                | |     | (___) |   | |   | (___)|| |   | | _____ | (__   | |   | |    /   )    /   )| (__   | (___)|
                | |     |  ___  |   | |   |     __)| |   | |(_____)|  __)  | |   | |   /   /    /   / |  __)  |     __)
                | |     | (   ) |   | |   | (\ (   | |   | |      | (     | |   | |  /   /    /   /  | (     | (\ (
                | (___/\| )   ( |___) (___| ) \ \__| (___) |      | )     | (___) | /   (_/\ /   (_/\| (___/\| ) \ \__
                (_______/|/     \|\_______/|/   \__/(_____)      |/      (_____)(_____/(_____/(_____/|/   \__/

      ==========================================================================================================================
                                                Seed: 1689265491399
                                                Inputs loaded 0
                                                Running 12 threads
      ==========================================================================================================================
      1.00 uptime |    467000 fuzz cases |    466269.88 fcps |      8 coverage |      8 inputs |      0 crashes [     0 unique]
      2.00 uptime |    921000 fuzz cases |    460106.95 fcps |      8 coverage |      8 inputs |      0 crashes [     0 unique]
      3.00 uptime |   1365000 fuzz cases |    454726.64 fcps |      9 coverage |      9 inputs |      0 crashes [     0 unique]
WORKER 7 -- INPUT => [102, 117, 122, 122, 105, 110, 103, 108, 97, 98, 115] -- ERROR ""An ASSERT_EQ instruction failed: 2 != 0.""
      4.00 uptime |   1806000 fuzz cases |    451285.76 fcps |     11 coverage |     12 inputs |    613 crashes [     1 unique]
      5.00 uptime |   2248000 fuzz cases |    449421.35 fcps |     11 coverage |     12 inputs |   1462 crashes [     1 unique]
```

```
func Fuzz_symbolic_execution(
    f: felt,
    u: felt,
    z: felt,
    z2: felt,
    i: felt,
    n: felt,
    g: felt,
    l: felt,
    a: felt,
    b: felt,
    s: felt,
) {
    if (f == 'f') {
        if (u == 'u') {
            if (z == 'z') {
                if (z2 == 'z') {
                    if (i == 'i') {
                        if (n == 'n') {
                            if (g == 'g') {
                                if (l == 'l') {
                                    if (a == 'a') {
                                        if (b == 'b') {
                                            if (s == 's') {
                                                assert 0 = 2;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return ();
}
```

Why Cairo-Fuzzer?

- **Example of a scenario where the use of Cairo-Fuzzer may be of interest**
  - You want to do bug bounty or just look for bugs on on-chain contracts.
  - You only have the artifact (json)
    - Yes **Thoth** can help you to decompile the artifact so you can understand the code.
  - You want to automate the audit or the contract testing
    - Writing tests is not funny at all …

Installing Cairo-Fuzzer

Installing Cairo-Fuzzer

- `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
- `git clone` https://github.com/FuzzingLabs/cairo-fuzzer.git
- The first run take 5 to 10mins to compile

Understanding the output of Cairo-Fuzzer

Example

- Let's use this example:
  - Compile it with
    - `cairo-compile fuzzinglabs.cairo --output fuzzinglabs.json`
  - Let's run the fuzzer using
    - `cargo run --release -- --cores 3 --contract tests/fuzzinglabs.json --function Fuzz_symbolic_execution`

```
func Fuzz_symbolic_execution(
    f: felt,
    u: felt,
    z: felt,
    z2: felt,
    i: felt,
    n: felt,
    g: felt,
    l: felt,
    a: felt,
    b: felt,
    s: felt,
) {
    if (f == 'f') {
        if (u == 'u') {
            if (z == 'z') {
                if (z2 == 'z') {
                    if (i == 'i') {
                        if (n == 'n') {
                            if (g == 'g') {
                                if (l == 'l') {
                                    if (a == 'a') {
                                        if (b == 'b') {
                                            if (s == 's') {
                                                assert 0 = 2;
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return ();
}
```

Understanding the output - The fuzzer

- Understanding the output : **1.00 uptime | 93000 fuzz cases | 92979.48 fcps | 5 coverage | 5 inputs | 0 crashes [0 unique]**:
  - **1.00 uptime**: Number of seconds the fuzzer is running
  - **93000 fuzz cases**: Number of executions done
  - **92979.48 fcps**: Number of Fuzz Case Per Second
  - **5 coverage**: Number of instruction reached by the fuzzer
  - **5 inputs**: Number of interesting inputs that generate a new coverage
  - **0 crashes [0 unique]**: Number of crashes and unique crashes

```
~/cairo-fuzzer (starknet-rs-fuzzer*) » cargo run --release -- --cores 3 --contract tests/fuzzinglabs.json --function Fuzz_symbolic_exe
cution

    Finished release [optimized] target(s) in 0.21s
     Running `target/release/cairo-fuzzer --cores 3 --contract tests/fuzzinglabs.json --function Fuzz_symbolic_execution`
    ==========================================================================================================================

    ==========================================================================================================================
                                                                Seed: 1680764433492
                                                                Inputs loaded 0
                                                                Running 3 threads
    ==========================================================================================================================
    1.00 uptime |     93000 fuzz cases |     92979.48 fcps |      5 coverage |      5 inputs |      0 crashes [     0 unique]
    2.00 uptime |    186000 fuzz cases |     92983.89 fcps |      6 coverage |      6 inputs |      0 crashes [     0 unique]
    3.00 uptime |    279000 fuzz cases |     92986.71 fcps |      6 coverage |      6 inputs |      0 crashes [     0 unique]
    4.00 uptime |    371000 fuzz cases |     92737.90 fcps |      6 coverage |      6 inputs |      0 crashes [     0 unique]
    5.00 uptime |    463000 fuzz cases |     92588.64 fcps |      6 coverage |      6 inputs |      0 crashes [     0 unique]
    6.00 uptime |    556000 fuzz cases |     92655.91 fcps |      7 coverage |      7 inputs |      0 crashes [     0 unique]
    7.00 uptime |    648000 fuzz cases |     92561.21 fcps |      8 coverage |      8 inputs |      0 crashes [     0 unique]
    8.00 uptime |    740000 fuzz cases |     92489.93 fcps |      9 coverage |      9 inputs |      0 crashes [     0 unique]
    9.00 uptime |    832000 fuzz cases |     92434.57 fcps |     10 coverage |     10 inputs |      0 crashes [     0 unique]
```

Understanding the output - The crash

- Once the fuzzer will find a unique crash you will have something like this:
  - You can see that the good input to reach the **assert 0 = 2** is **[102, 117, 122, 122, 105, 110, 103, 108, 97, 98, 115]**.
    - In ascii we get **[f,u,z,z,i,n,g,l,a,b,s]**.
  - Running the function **Fuzz_symbolic_execution** with **(102, 117, 122, 122, 105, 110, 103, 108, 97, 98, 115)** will lead to the assert.

```
        19.00 uptime |   1757000 fuzz cases |     92464.78 fcps |     10 coverage |     10 inputs |      0 crashes [     0 unique]
        20.00 uptime |   1849000 fuzz cases |     92441.20 fcps |     10 coverage |     10 inputs |      0 crashes [     0 unique]
        21.00 uptime |   1942000 fuzz cases |     92467.49 fcps |     10 coverage |     10 inputs |      0 crashes [     0 unique]
        22.00 uptime |   2036000 fuzz cases |     92536.84 fcps |     11 coverage |     11 inputs |      0 crashes [     0 unique]
        23.00 uptime |   2128000 fuzz cases |     92513.16 fcps |     11 coverage |     11 inputs |      0 crashes [     0 unique]
WORKER 0 -- INPUT => [102, 117, 122, 122, 105, 110, 103, 108, 97, 98, 115] -- ERROR ""An ASSERT_EQ instruction failed: 2 != 0.""
        24.00 uptime |   2221000 fuzz cases |     92533.07 fcps |     11 coverage |     12 inputs |     74 crashes [     1 unique]
        25.00 uptime |   2315000 fuzz cases |     92591.39 fcps |     11 coverage |     12 inputs |    269 crashes [     1 unique]
        26.00 uptime |   2408000 fuzz cases |     92606.73 fcps |     11 coverage |     12 inputs |    469 crashes [     1 unique]
```

Cairo-Fuzzer CheatSheet

CheatSheet

- Fuzzing function of a contract:
  - `cargo run --release -- --cores 13 --contract tests/fuzzinglabs-starknet.json --function "fuzzinglabs_starknet"`
- Fuzzing function of a contract with a number of iteration max:
  - `cargo run --release -- --cores 13 --contract tests/fuzzinglabs-starknet.json --function "fuzzinglabs_starknet" --iter 100000`
- Load old corpus:
  - `cargo run --release -- --cores 13 --contract tests/fuzzinglabs-starknet.json --function "fuzzinglabs_starknet" --inputfile "fuzzer_workspace/fuzzinglabs_starknet/inputs/fuzzinglabs_starknet_2023-04-04--22:53:23.json"`

CheatSheet

- Fuzzing using a config file:
  - `cargo run --release -- --config tests/config.json`
- Advantages:
  - Allows permanent configuration
  - Easier to understand configuration than command lines
  - Allows easy transmission of configuration to others
  - Allows multiple configurations

```
{
    "cores": 1,
    "logs": false,
    "replay": false,
    "minimizer": false,
    "contract_file": "tests/fuzzinglabs.json",
    "function_name": "Fuzz_symbolic_execution",
    "input_file": "",
    "crash_file": "",
    "input_folder": "",
    "crash_folder": "",
    "workspace": "fuzzer_workspace",
    "proptesting": false,
    "iter": -1,
    "dict": "tests/dict"
}
```

Property Testing

Property Testing

- This feature automates the search for the functions you want to fuzz.
- If the function starts with "Fuzz", it will be added to the list of functions to fuzz during fuzzer execution.
- This feature can be useful for creating test functions (functional or unit tests).
- You can enable it with **- -proptesting**
  - `cargo run --release -- --cores 13 --contract tests/fuzzinglabs.json --proptesting --iter 500000`

```
%builtins output
func Fuzz_one(…
) {…
}

func Fuzz_two(…
) {…
}

func Fuzz_three(
) {…
}

func main{output_ptr: felt*}() {
    return ();
}
```

Fuzzing using a dictionary

Dictionary

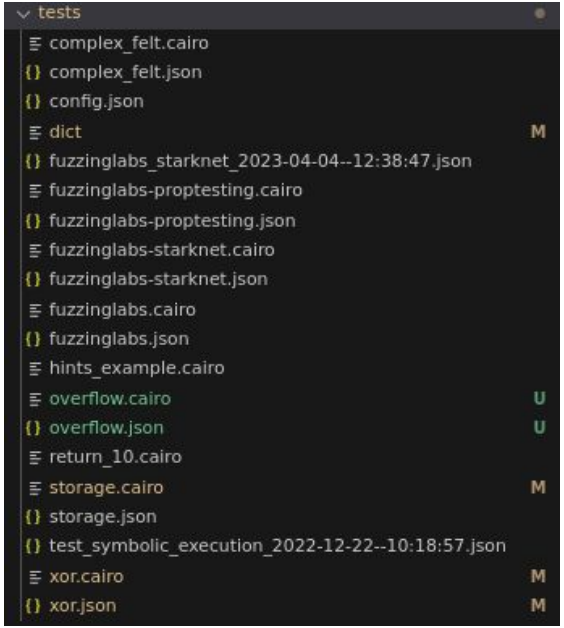
- Cairo-Fuzzer supports user-supplied dictionaries with input language keywords or other interesting byte sequences (e.g. multi-byte magic values).
- Use **- -dict=DICTIONARY_FILE**. For some input languages using a dictionary may significantly improve the search speed.
- Dictionnary format is the same as other fuzzers such as Honggfuzz or libafl.
  - `cargo run --release -- --cores 13 --contract tests/fuzzinglabs.json --function "Fuzz_symbolic_execution" --dict tests/dict`

```
dict
key1=9992913
key2=7423848
key3=7214781287489724
key4=757483838389399
```

Testing

Example

- Run the `fuzzinglabs` contract “tests/fuzzinglabs.json”
- Run the `fuzzinglabs-starknet` contract “tests/fuzzinglabs-starknet.json”
- Run the `storage` contract “tests/storage.json”
- Run the `complex_felt` contract “tests/complext_felt.json”
  - Play with the dictionary to improve fuzzing



Your example

- Choose one of your contract and come to fuzz it with us!

Why cairo-fuzzer does not support Cairo 1.0

Support of Cairo 1.0 - Problems

- The json artifact of Cairo1.0 does not contain all the information Cairo-Fuzzer needs
  - We can get the offset of the functions from the json
  - We can get the function names from the sierra
  - We cannot match the names with the offsets
    - We cannot get the number of arguments

- Cairo-Fuzzer is not a black-box fuzzer
  - Fuzzing a function we don't know is not really efficient
  - We will need to **guess** the number of arguments

- The artifact of Cairo1.0 is still unstable and changes a lot

```
pub fn run_from_entrypoint(
    &mut self,
    entrypoint: usize,
    args: &[&CairoArg],
    verify_secure: bool,
    program_segment_size: Option<usize>,
    vm: &mut VirtualMachine,
    hint_processor: &mut dyn HintProcessor,
```

Support of Cairo 1.0 - Potential solutions

- Github issue on the Cairo repository to ask them to add a global output containing all the data needed
  - A command like `--global_output`

- Working with Thoth and try to guess some information and correlate them to generate a global output
  - But it will be a mess to maintain

Future of Cairo-Fuzzer

Future of Cairo-Fuzzer

- Next features:
  - Sequence of call using the same context
  - Support of Cairo1.0

- Do not hesitate to make github issues if you have any
  - Feature proposition
  - Bug to report

- **Contacts us** if you need **customs security tool** development!
  - Twitter: @Pat_Ventuzelo
  - Mail: patrick@fuzzinglabs.com

- Thanks for your time! Any questions?