Fuzzing programs with Structure Aware Fuzzers

Collaboration with Dr. Rahul Gopinath, University of Sydney

TEAM

Parvathy C M TCR21CS049
Libna Kuriakose T TCR21CS033
Krishnahari P TCR21CS032
Darshana Das K TCR21CS020

GUIDE

Dr. Ezudheen P

Project Overview

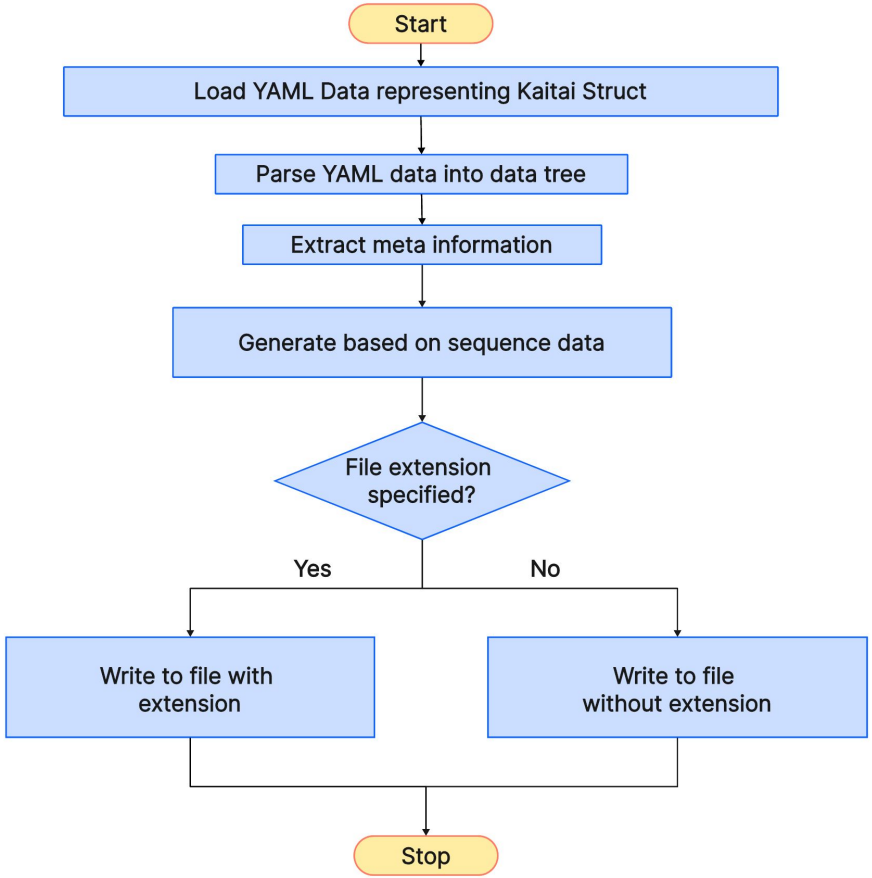
The project has two parts:

- **Generation-based fuzzer:** Designing a fuzzer to generate test cases based on the specified format in Kaitai Struct.
- **Integration with LibFuzzer:** Converting Kaitai Struct into Protobuf to utilize libprotobuf-mutator for structured input generation, coupled with coverage guidance and fuzzing logic provided by LibFuzzer.

Generation-based fuzzer

We developed a fuzzer that generates test cases according to the format specified in Kaitai Struct.

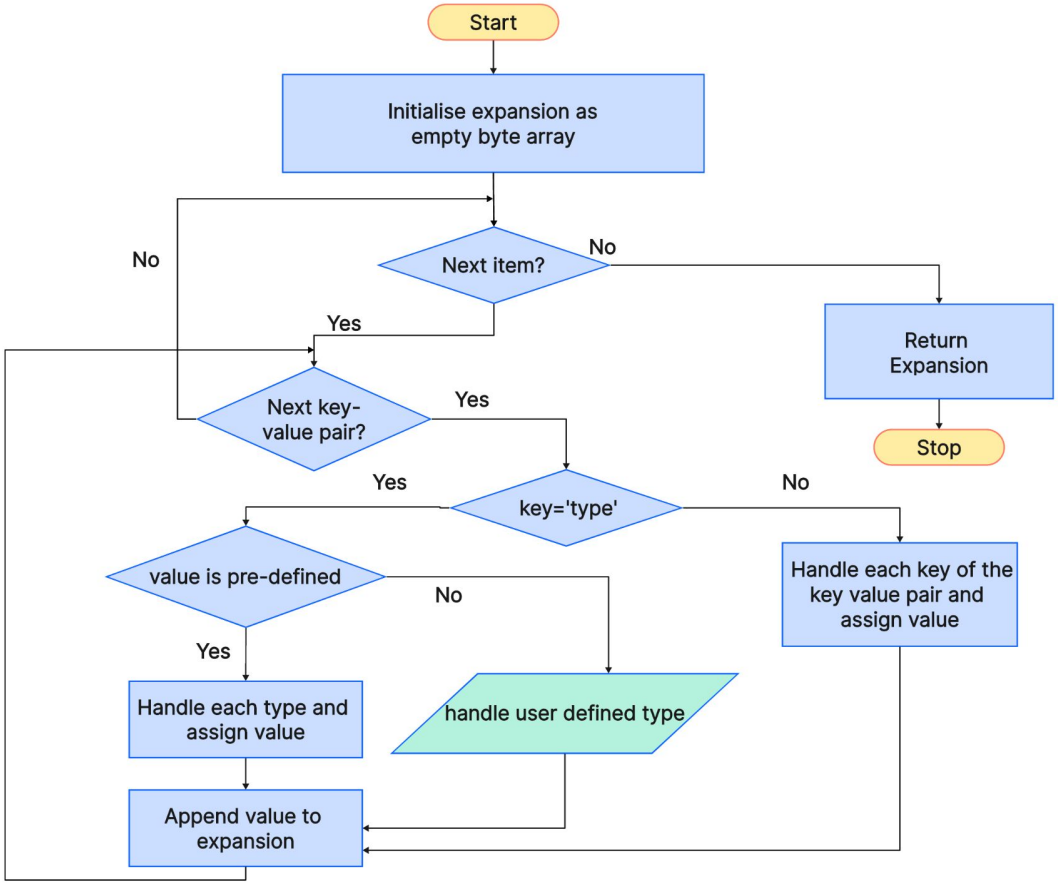
High Level Representation



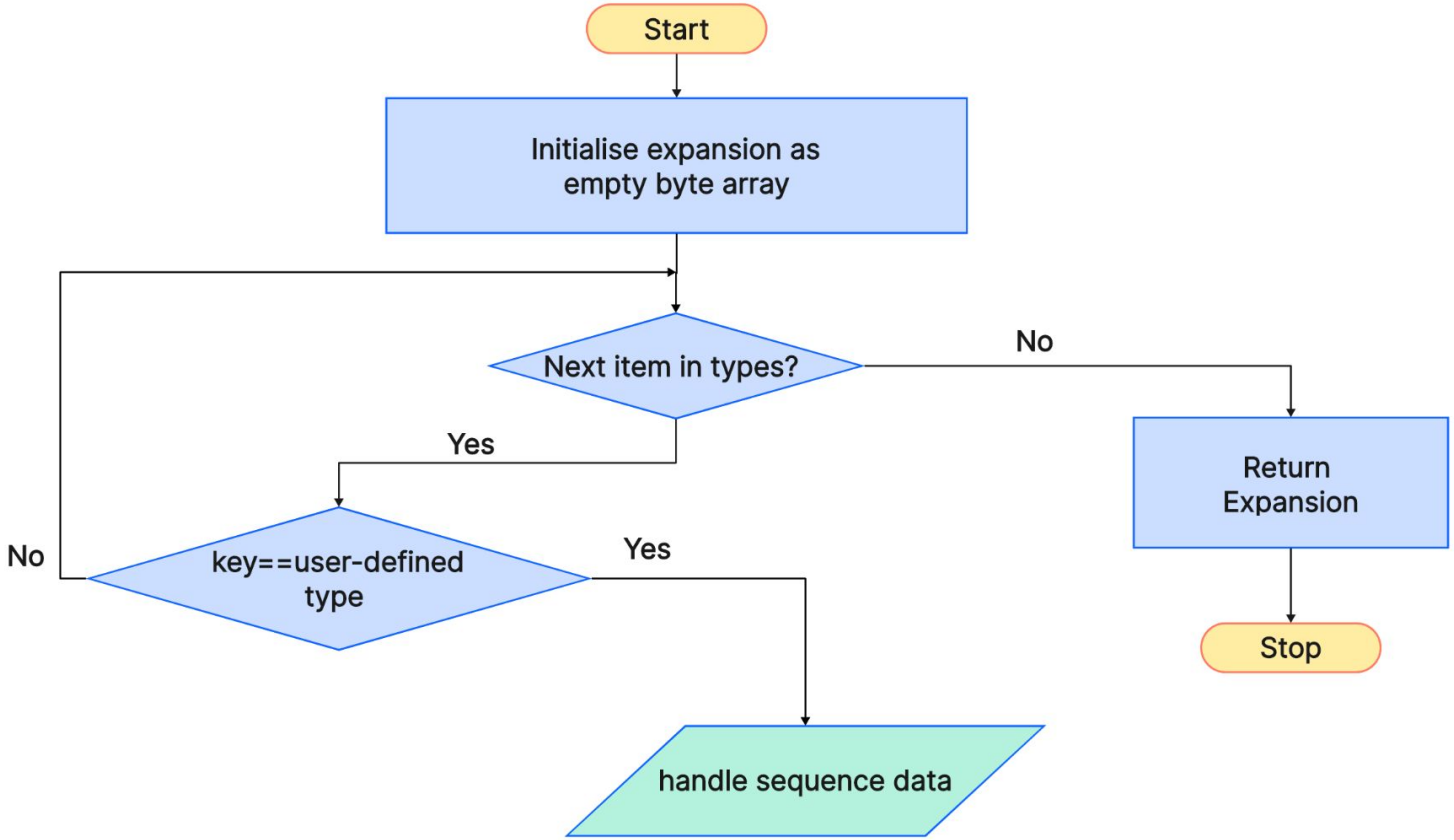
An example of Kaitai Struct:

```
meta:
  id: example
  endian: le
seq:
  - id: magic
    contents: [12, 22]
  - id: attribute1
    type: str
    size: 24
    encoding: UTF-8
  - id: attribute3
    type: attribute3_type
types:
  attribute3_type:
    seq:
      - id: attribute5
        type: u2
```

Handle Sequence Data



Handle User-defined type



Implementation

We implemented 80 percentage of the kaitai struct features.

- Implemented meta
  - File extension
  - Endianness
  - id
- Seq section
  - Fixed size structures
  - Basic data types
  - Doc strings
  - Checking for magic signatures
  - Repetitions
- Seq section
  - Doc key
  - Encoding- UTF and ASCII Encoding
  - Variable length structures
- Type section
  - User defined types

Structure aware fuzzing using LibFuzzer and Kaitai Struct

This fuzzer employs LibFuzzer for in-process, coverage-guided, evolutionary fuzzing, with structure defined by Kaitai Struct for enhanced structure-aware fuzzing.

LibFuzzer or AFLSmart?

- **LibFuzzer:** coverage-guided evolutionary fuzzing engine developed by Google, designed for in-process fuzzing. It automatically instruments target programs and guides the generation of test inputs to maximize code coverage, helping uncover bugs and vulnerabilities efficiently.

- **AFLSmart**: AFLSmart extends AFL (American Fuzzy Lop) by integrating symbolic execution, enhancing its ability to discover complex bugs by guiding fuzzing towards deeper code exploration.

AFLSmart

- Based on AFL and input structure component of Peach
- Smart greybox fuzzing - Coverage based greybox fuzzing with input structure awareness
- Smart mutation operators
- Validity based power schedule

Analysis

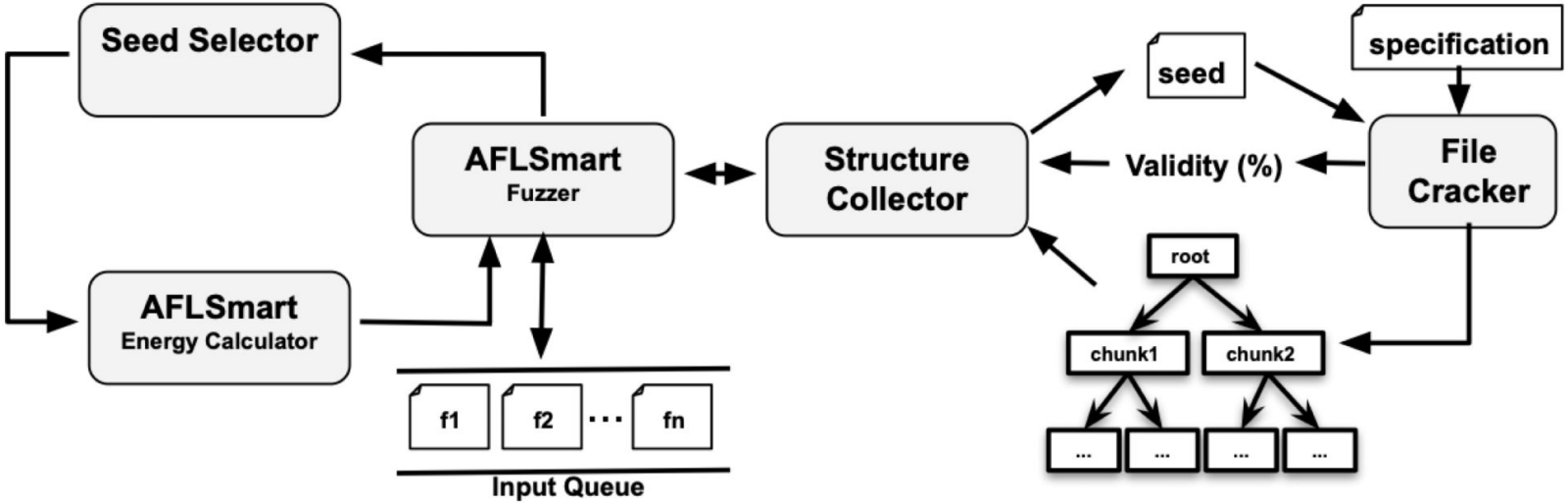


Fig: Architecture of AFLSmart

How to integrate Kaitai Struct?

- Implement a parser for Kaitai Struct that can read the file format and generate a tree structure.
- Replace the Peach-specific code in **update_input_structure()** with the logic to parse the input Kaitai Struct file using your parser.
- Modify the code to handle the parsed tree structure appropriately, updating the queue entry and validity calculation based on the parsed data.

LibFuzzer

- LibFuzzer is part of the LLVM compiler infrastructure project
- In-process fuzzing
- Coverage guided fuzzing
- Integration with sanitizers
- Seed corpus fuzzing
- Structure Aware Fuzzing

LibFuzzer

- LibFuzzer runs the target program within the same process
- It can handle structured input formats more effectively with the use of custom mutators, making it suitable for structure-aware fuzzing.
- It offers built-in functionality to minimize the corpus size, which can be helpful for managing storage and improving fuzzing efficiency

LibFuzzer or AFLSmart?

LibFuzzer is our answer. Why?

- LibFuzzer is favored in industry circles due to its numerous advantages, including its efficiency, robustness, and seamless integration with modern development workflows.
- AFLSmart relies on the Peach framework, which is not currently widely used in the industry.
- Deploying AFLSmart on operating systems like Windows can be challenging, whereas LibFuzzer can be used on Windows with ease.

Input Generation in libFuzzer

1. Generic random fuzzing
2. Custom mutators
3. Structured fuzzing using libprotobuf-mutator

libprotobuf-mutator

Libprotobuf-mutator is a library to randomly mutate protobuffers.
It could be used together with guided fuzzing engines, such as libFuzzer.

Protocol Buffers

- Protocol Buffers, or protobuf, is a data serialization format developed by Google.
- It's designed to be language-neutral, platform-neutral, and efficient.
- Protocol Buffers use a simple language to define the structure of the data, enabling easy communication between different systems.
- Libprotobuf mutator leverages Protocol Buffers' structure to implement structure aware fuzzing.

Protobuffers - Example

Ex.proto file

```
syntax = "proto3";

package libfuzzer_example;
import "google/protobuf/any.proto";
message Msg {
  float optional_float = 1;
  uint64 optional_uint64 = 2;
  string optional_string = 3;
  bytes attribute = 4
}
```

Ex.ksy file

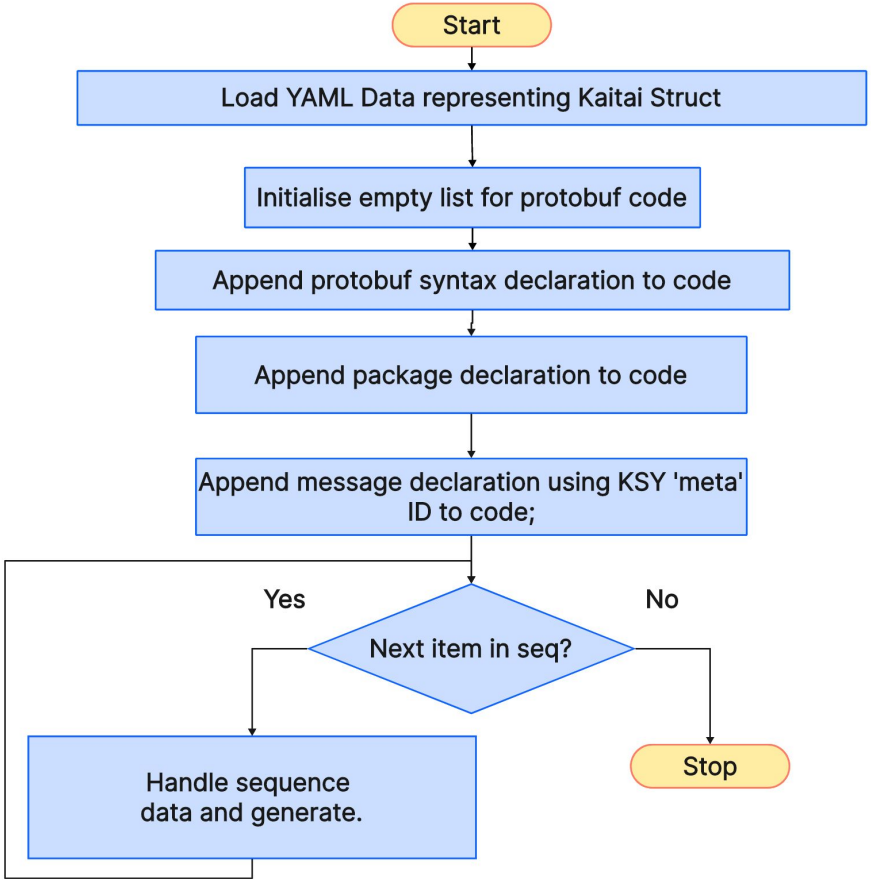
```
meta:
  id: msg
  endian: le
seq:
  - id: optional_float
    type: f4
  - id: optional_uint64
    type: u8
  - id: optional_string
    type: str
  - id: attribute
    type: str
```

How we integrate Kaitai Struct?

Three approaches:

1. Convert Kaitai Struct definitions to Protocol Buffers (protobuf) format for compatibility with existing protobuf libraries and tools.
2. Convert Kaitai Struct definitions into a representation that aligns with the format produced by the libprotobuf-mutator library after parsing the protobuf message object.
3. Develop a specialized system, akin to libprotobuf mutator, tailored specifically to handle Kaitai Struct definitions. This approach involves creating custom mutation strategies and fuzzing techniques optimized for Kaitai Struct-based data formats.

Conversion of Kaitai Struct to Protobuf



Thank you