Fuzz Testing Report

Revision

Version 5
11/15/21 10:44 AM

SME

Marwan Abi-Antoun

Abstract

This document describes the process to produce a fuzz testing report.

Group / Owner

Security / Vulnerability Assessment Analyst

Motivation

This document is motivated by the need to have runtime-specific, security-related feedback in the development of software for use within safety-critical, cyber-physical systems for certification of compliance to standards such as **ISO 21434** and **ISO 26262**.

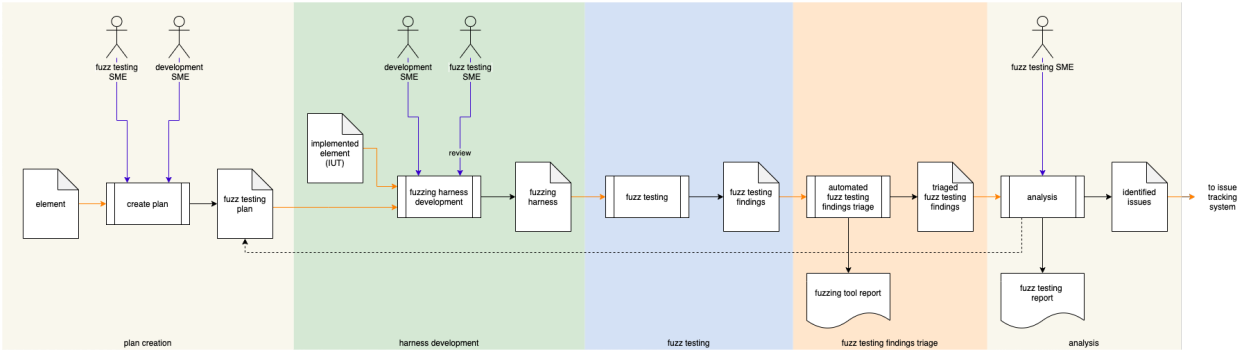
License

Overview

Although the quality of security-related feedback from the compiler, static and dynamic analysis tools have become much better over time, there are many situations they do not consider. Fuzz testing provides coverage for some of those deficiencies. Although lacking the level of automation of other methods, fuzz testing allows for highly tailored behavior analysis at the unit level.

Note: Fuzz testing is a team exercise, encompassing program and project managers, developers, cybersecurity engineers, and testers.

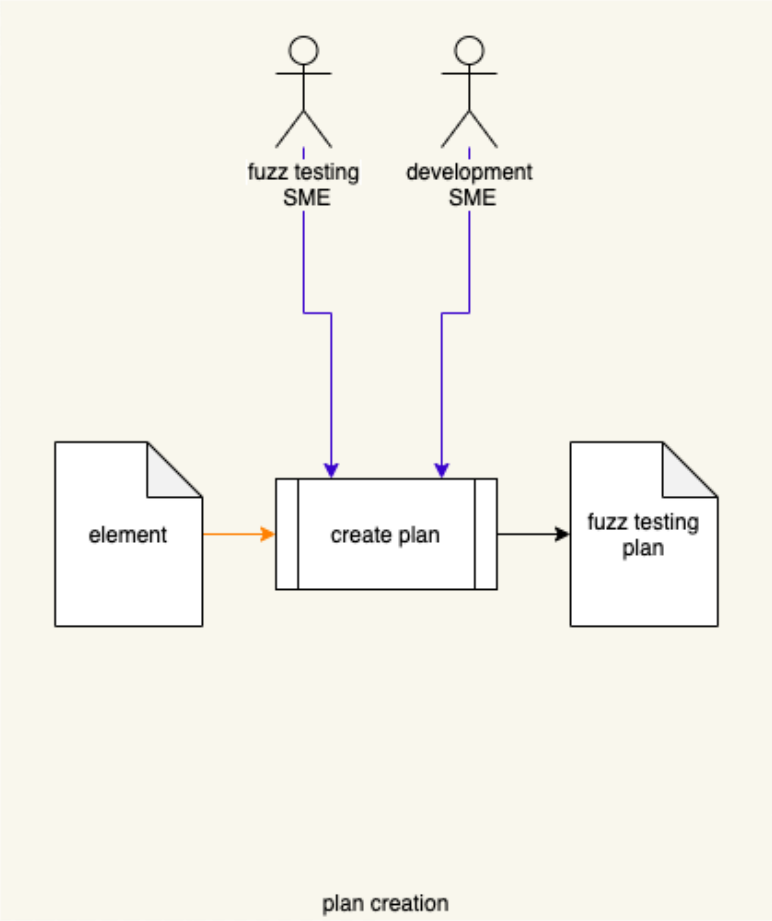
The following diagram illustrates the process to be used:



Process

Plan Creation

Inputs	element
Outputs	Fuzz Testing Plan
Participants	Fuzz Testing SME Development SME

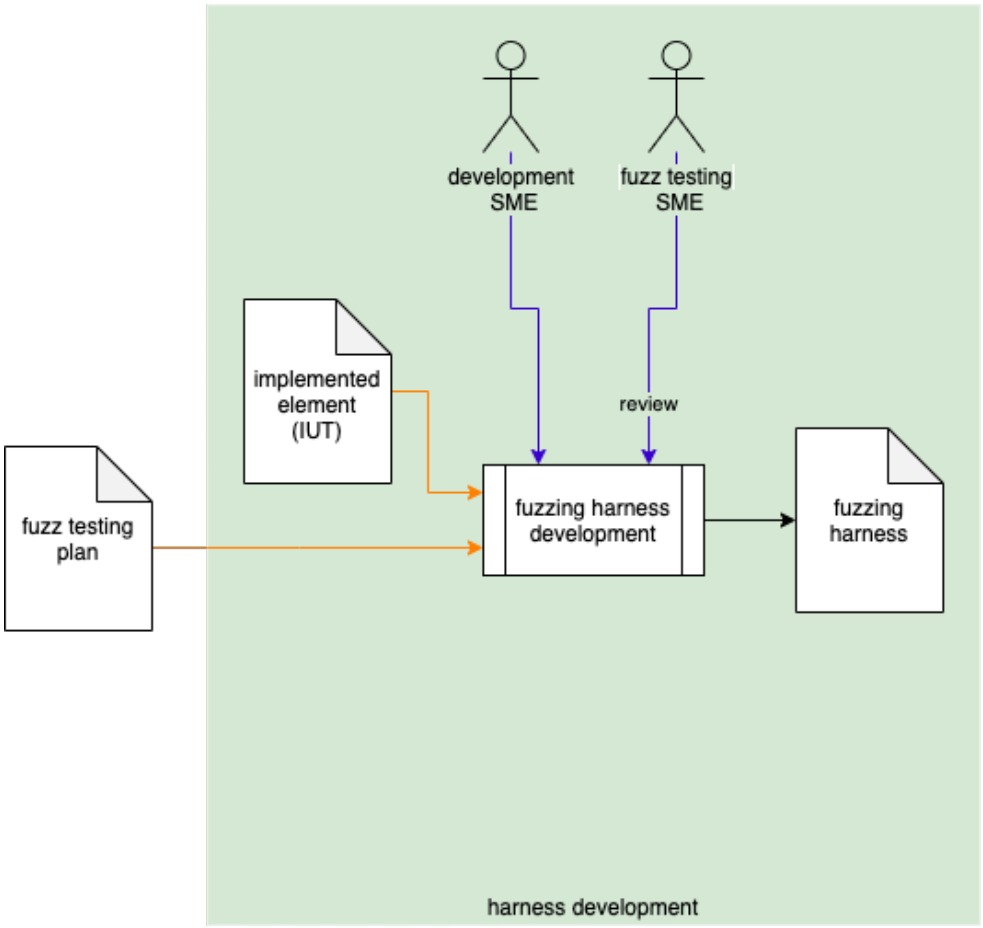


Using the **Element**'s interface, the Fuzz Testing SME and Development SME, identify the areas of the implementation that would benefit from fuzz testing, and develop a **Fuzz Testing Plan**.

Note: The **Fuzz Testing Plan** is refined iteratively.

Harness Development

Inputs	Implemented system element Fuzz Testing Plan
Outputs	Fuzzing harness
Participants	Fuzz Testing SME Development SME



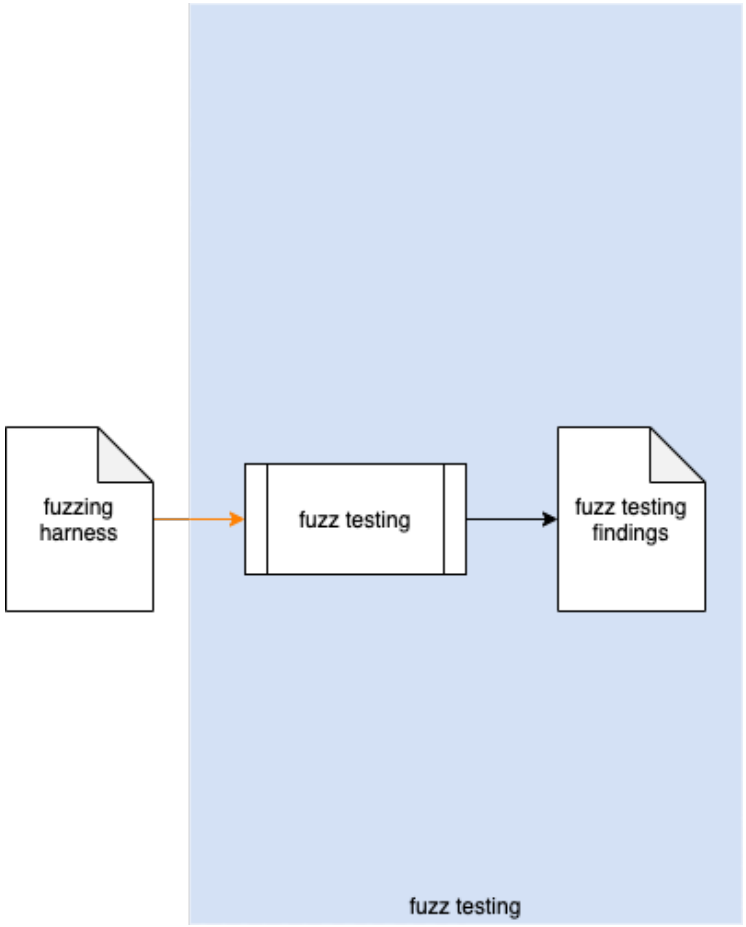
The Fuzz Testing SME and Development SME develop a **Fuzzing Harness** for the **Implemented Element** based on the **Fuzz Testing Plan**.

Note: It is easier to develop a fuzzing harness for code that is unit testable.

Note: The fuzzing harness should be created concurrently with the unit tests.

Fuzz Testing

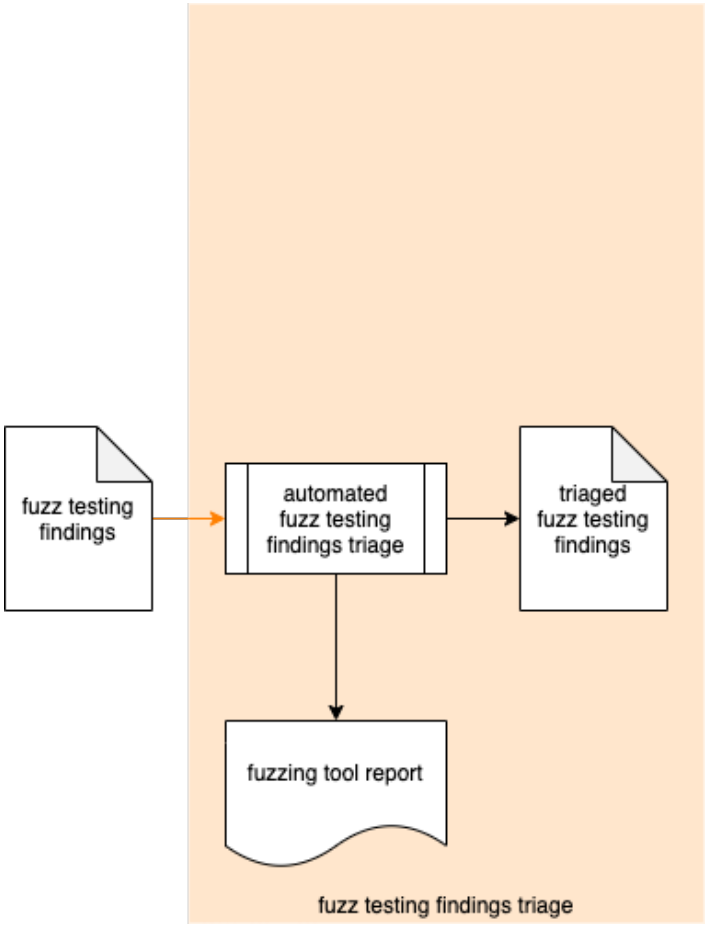
Inputs	Fuzz harness
Outputs	Fuzz testing findings
Participants	**None**



The **Fuzz Harness** is invoked and produces **Fuzz Testing Findings**.

Fuzz Testing Findings Triage

Inputs	Fuzz testing findings
Outputs	Triaged fuzz testing findings Fuzzing Tool Report
Participants	**None**



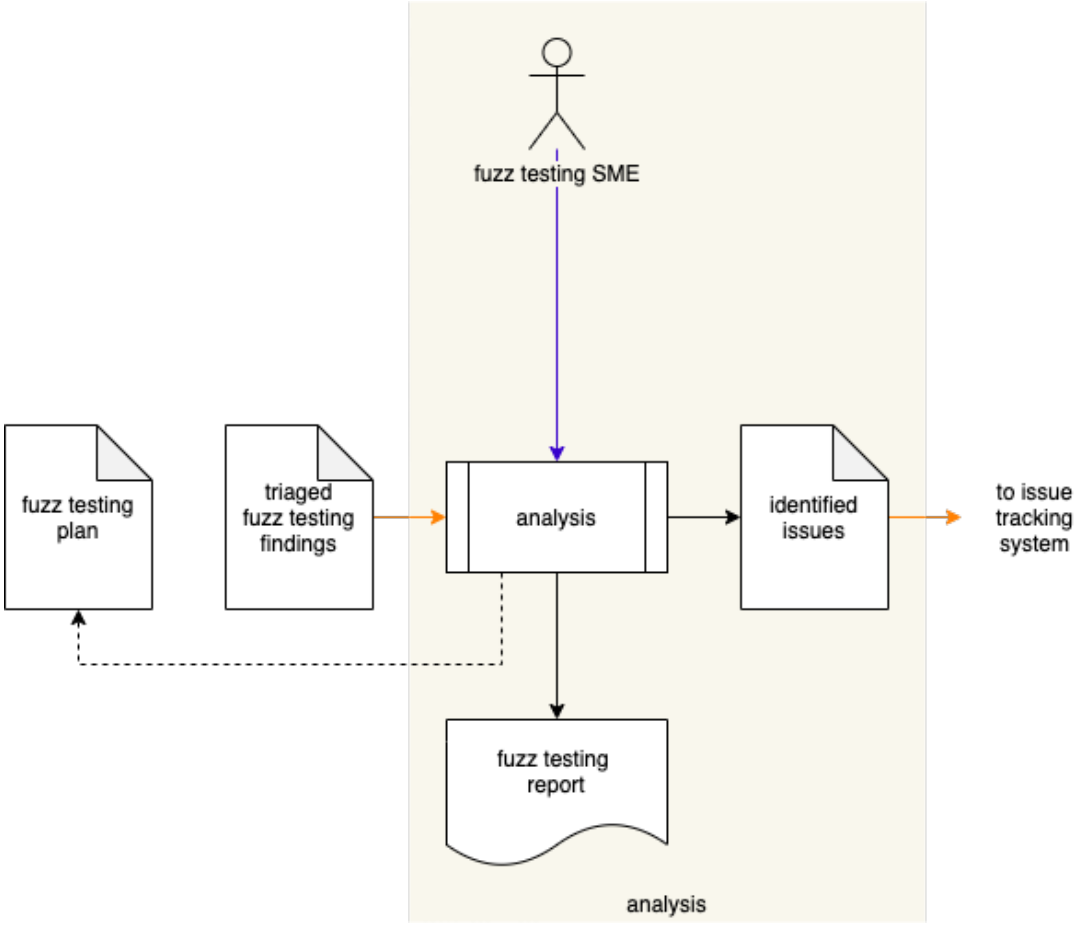
An automated triage of the **Fuzz Testing Findings** produces a set of **Triaged Fuzz Testing Findings**. A **Fuzzing Tool Report** is generated.

The **Fuzzing Tool Report** includes information such as:

- harnesses executed
- duration of fuzzing
- defects found
- defect classification

Analysis

Inputs	Triaged fuzz testing findings
Outputs	Identified issues Fuzz Testing Report
Participants	Fuzz Testing SME



The Fuzz Testing SME analyzes the **Triaged Fuzz Testing Findings** and attempts to reproduce the issues on the current software. Reproducible issues are collected into a set of **Identified Issues** and submitted to the issue tracking system.

Findings are summarized in a **Fuzz Testing Report** [2]. The **Fuzz Testing Plan** is updated based on issues found.

Identified Issues

The recommended form of the **Identified Issues** artifact is a Static Analysis Results Interchange Format (**SARIF**) encoded JSON. This document assumes SARIF version 2.1.0 [6] or later.

Fuzz Testing Report

The **Fuzz Testing Report** is recommended to be produced from the **Identified Issues** artifact and should detail the exposed issues.

The report contains one or more analysis runs. Each run includes:

- Description of the tool used
- Description of the analyzed element
- Tool invocation settings
- Results of the analysis

The tool description includes:

- Name
- Version
- URI to tool documentation
- Tool rules

The tool rules (one or more) convey the classes of analysis performed. Each includes:

- ID (unique)
- Name
- Short description of the rule
- Full description of the rule
- URI to rule documentation
- Reference (optional) to associated taxonomy entry (Common Weakness Enumeration [1], …)

The analyzed element description (one or more) provides information related to the element under consideration. Each includes:

- URI to analyzed element
- URI to repository the element came from

The analysis results (one or more) describe the issues exposed by the analysis. Each includes:

- Human-readable description
- Location within the element of the issue
- Severity of the issue
- Reference (optional) to the associated taxonomy entry

Note: The above information is directly provided for by the SARIF specification.

It is recommended to include the following additional fuzzing-specific information:

- Type of exposure (first-order attack surface)
- Type of external data (sensor, socket, file, …)
- Type of parsing performed
- Triage level (regular, additional dynamic analysis, specific sanitizers, …)
- Fuzzing harness compiler (**`gcc`**, **`gcc-afl`**, **`clang-libfuzzer`**, …)
- Type of harness (standalone, library, dual [recommended])

Note: This set of information is not provided for by the baseline SARIF specification but is supported by the SARIF specification's extension capability.

Note: When extending the SARIF encoding to include the above, a namespace to denote fuzzing (**`Fuzzing::`*`tag`***) is recommended.

References

1. **Common Weakness Enumeration (CWE)** https://cwe.mitre.org/
2. **Fuzzing for Software Security Testing and Quality Assurance** (Takanen, Demott, and Miller)
3. **Fuzz Testing Report** (AVCDL tertiary document)
4. **Static Analysis Report** (AVCDL secondary document)
5. **Dynamic Analysis Report** (AVCDL secondary document)
6. **Static Analysis Results Interchange Format (SARIF) Version 2.1.0** https://docs.oasis-open.org/sarif/sarif/v2.1.0/os/sarif-v2.1.0-os.pdf