Model-based Behavioural Fuzzing

Martin Schneider (Fraunhofer FOKUS)
Outline

- Introduction to fuzzing
- Behavioural fuzzing of UML sequence diagrams
- Test case selection by augmenting the model
- Conclusions and outlook
Introduction to Fuzzing
Definition

- Fuzzing is about injecting invalid or random inputs
  - to obtain unexpected behaviour
  - to identify errors and potential vulnerabilities

- Interface robustness testing

- Fuzzing is able to find (zero-day-)vulnerabilities, e.g.
  - crashes
  - denial of service
  - security exposures
  - performance degradation

- No false positives
**Random-based fuzzers** generate randomly input data. They don’t know nearly anything about the SUT’s protocol.

fuzzed input: `HdmxH&k dd#**&%`

**Template-based fuzzers** uses existing traces (files, ...) and fuzzes some data.

_template:_ `GET /index.html`

_fuzzed input:_ `GE? /index.html, GET /inde?.html`

**Block-based fuzzers** break individual protocol messages down in static (grey) and variable (white) parts and fuzz only the variable part.

```
GET /index.html
```

_fuzzed input:_ `GET /inde?.html, GET /index.&%ml`

**Dynamic Generation/Evolution-based fuzzers** learn the protocol of the SUT from feeding the SUT with data and interpreting its responses, for example using evolutionary algorithms.
Introduction to Fuzzing
Model-based Fuzzers

- **Model-based fuzzers** uses models of the input domain (protocol models, e.g. context free grammars), for generating systematic non-random test cases.

- The model is executed on- or offline to generate complex interaction with the SUT.

- Thus it is possible fuzz data after passing a particular point.

- Model-based fuzzers finds defects which human testers would fail to find.

In current state of the art, behavioural fuzzing is done only in small portions using state machines:
  - by fuzzing the message type,
  - by reordering messages and
  - by inserting, repeating and dropping messages.

The motivation for behavioural fuzzing is that vulnerabilities can be revealed not only when invalid input data is accepted and processed by the SUT, but also by stimulating the SUT with invalid sequences of messages.

A real-world example is given in [Tak10] where a vulnerability in Apache web server was found by repeating the host header message in an HTTP request.

To start with, we fuzz existing functional test cases.

Behavourial Fuzzing of UML Sequence Diagrams

Approach of Behavioural Fuzzing

- Test cases are generated by **fuzzing valid sequence diagrams, e.g. functional test cases.**
- Behavioural fuzzing is realized by changing the order and appearance of messages in two ways:
  - **By rearranging messages.** This enables straight-lined sequences to be fuzzed. Fuzzing operations are for example remove, move or repeat a message.
  - **By utilising control structures of UML 2 sequence diagrams,** such as combined fragments, guards, constraints and invariants. This allows more sophisticated and specific behavioural fuzzing.
- By applying one or more fuzzing operations to a valid sequence, **invalid sequences** (= **behavioural fuzzing test cases**) are generated.
Behviournal Fuzzing of UML Sequence Diagrams

Operators for Messages

- remove, repeat, move, change type of message, insert message
- swap messages
- permute messages
  - regarding a single SUT lifeline (weak sequencing)
  - regarding several SUT lifelines (strict sequencing)
- rotate messages
Behavoural Fuzzing of UML Sequence Diagrams

Combined Fragments

- combined fragment are **control structures** for UML sequence diagrams

- a **combined fragment** consists of:
  - interaction operator
    - Alternatives
    - Option
    - Break
    - Weak Sequencing
    - Strict Sequencing
    - Negative
    - Consider/Ignore
    - Loop
  - interaction operands
  - guards
Behavourial Fuzzing of UML Sequence Diagrams
Operators for Combined Fragments

- negate interaction constraint, **interchange interaction constraints**
- change bounds of loop
- disintegrate combined fragments
- insert, remove, repeat, move combined fragments
- change interaction operator
Three values have an impact on the number of test cases that can be generated from one valid sequence using behavioural fuzzing:

- number of fuzzing operators \( o = |\text{operators}| \)
- number of elements in the sequence diagram \( e = |\text{elements}_{sequence}| \)
- maximum number of fuzzing operators per test case \( n \)

A fuzzing operator can be applied to an element of a sequence in different ways. The number of possibilities to apply all fuzzing operators to all elements of a sequence is

\[
o' = o \cdot e^k
\]

where \( k \) is a constant representing the maximum number of possibilities to apply a fuzzing operator to one model element.

The following formula is a first approximation of the complexity:

\[
O \left( \sum_{i=1}^{n} \frac{o'!}{(o' - i)!} \right)
\]
UMLsec provides the following tags for role-based access control:

- **role**: assigns users and roles by a set of tuples \((user, role)\)
- **right**: assigns roles and rights by a set of tuples \((role, protected resources)\)
- **protected**: identifies the protected resources users

UMLsec’s stereotype **rbac** is annotated to UML activity diagrams.

We use it to annotate UML sequence diagram:
- New tag **authentication** to identify messages for authentication/login and logout.
Test Case Selection by Augmenting the Model

Test Case Generation

Application of fuzzing operator
MoveMessage
Conclusions

- Test evaluation and assessment
- Large number of generated test cases, incl.
  - duplicates
    - move message x + remove message x for generating one test case is equivalent to remove message x
  - unintentionally valid sequence diagrams
    - modifying one sequence diagram by applying fuzzing operators may result in a valid sequence diagram

Outlook

- Studies are necessary to show the efficacy of the presented approach.
  - At the moment, we are applying the presented approach to a DIAMONDS case study from the banking domain:
    See DIAMONDS booth at ITEA Co-summit next week!

- Optimized test generation (risk and security oriented test generation, finding more appropriate security stereotypes)
- Combination with data fuzzing methods.
Thank you for your attention!

Any questions?