DESIGN DECISIONS IN THE DEVELOPMENT OF A GRAPHICAL LANGUAGE FOR RISK-DRIVEN SECURITY TESTING

Gencer Erdogan, Ketil Stølen
4th Intl. Workshop on Risk Assessment and Risk-driven Quality Assurance
Graz, 18.10.2016
Outline

• Motivation
• Example-driven explanation of the CORAL language
• Major design decisions
• Conclusion
Motivation

• Challenges in testing in general:
  • Impossible to test a system exhaustively (testing all possible execution paths, testing all possible combinations of input to a system, etc.)
  • Very difficult to identify the most important test cases.

• Challenge in security testing: we need to think how security can be compromised (think like an attacker)

• Possible solution in security testing is to support the testing with security risk assessment

• We designed CORAL to help testers determine tests that are most likely to reveal severe security vulnerabilities
Overview of the method

Part 1: Test planning
- Prepare system model
- Identify security assets
- Create risk matrix

The CORAL language
- Design tests
- Execute tests
- Create test report

Part 2: Security risk assessment
- Identify risks
- Estimate risks
- Evaluate risks

Part 3: Security testing
- Prepare system model
- Identify security assets
- Create risk matrix
Example

Prepare system model | Identify security assets | Create risk matrix

sd User registers as a customer in shopping website

- User web browser
  - regCust(info)
- Shopping website
  - insertIntoDB(info)
  - getCustInfo()
- Company database
  - custInfo()
  - dispCustInfo()

Availability of custData

Risk level 9
Risk level 8
Risk level 7
Risk level 6
Risk level 5
Risk level 4
Risk level 3
Risk level 2
Risk level 1

Part 1: Test planning
Example

Identify risks | Estimate risks | Evaluate risks

Hacker deletes customer table in database by carrying out SQL injection.

Availability of custData

Customer table deleted

displayCustInfo()
Example

Part 3: Security testing

Design tests | Execute tests | Create test report

sd Hacker deletes customer table in database by carrying out SQL injection: Verdict

«TestComponent»

User web browser

Shopping website

Company database

insertIntoDB(\texttt{SQLIdel})

getCustInfo()

custInfo()

dispCustInfo()
Design decisions

• Graphical versus textual
• Risk annotation versus tables
• Sequence diagrams versus other UML representations
• CORAL versus attack trees
• CORAL versus formal methods
Graphical vs textual

**Textual**
- Sufficient for the tester to know the source-code language
- No abstraction
- Tester has to create mental model of the risk picture
- Tester has to read code from top to bottom to capture risk-related information such as unwanted incidents, frequencies, etc.

**Graphical**
- Covers a scenario that occurs multiple times in the source code by a single diagram
- Captures the interaction between independent actors/processes in a manner not possible using source code
- Allows describing the behavior of human actors, threats, and working procedures
Risk annotations vs tables

- Tables are also two-dimensional (a cell is identified by the row and column heading)
- Tables present all information consistently w.r.t. the headings
- Guides the reader to relevant info in a structured manner
- Text in tables removes the need to interpret semantics of graphical symbols

<table>
<thead>
<tr>
<th>Element</th>
<th>Characteristics</th>
<th>Guide word</th>
<th>Deviation</th>
<th>Possible Causes</th>
<th>Consequences</th>
<th>Safeguards</th>
<th>Comments</th>
<th>Action required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise Options form</td>
<td>Selling or buying shares in a company</td>
<td>No (not)</td>
<td>HTTP requests not sanitized</td>
<td>Web-application form has no input validation</td>
<td>An admin-feature is accessed</td>
<td>No</td>
<td>HTTP requests may have been tampered to access restricted features</td>
<td>Implement input validation mechanism</td>
</tr>
</tbody>
</table>

Advantage of CORAL: annotate risk info on the locations where it belongs in a style corresponding to the underlying modeling language
Sequence diagrams vs other UML representations

- **State machines**
  - States correspond to chronological events in a lifeline
  - Possible to represent the same risk-related info as in CORAL
  - Advantage of sequence dgrm. – isolate particular scenarios without having to consider the behavior for other scenarios

- **Class diagrams**
  - Useful to describe the structure of system to test
  - The kind of dynamic behavior that CORAL address cannot be specified using class diagrams

- **(mis)use-case diagrams**
  - Useful for capturing high-level threat scenarios a system may be exposed to
  - Useful to design high-level security test procedures
CORAL vs attack trees

Annotating sequence diagrams: sequence diagrams + attack trees

As for tables, we think testers benefit from an integrated approach
CORAL vs formal methods

- Not formal in its classical meaning (this requires a mathematical semantics and a formalization of the natural language semantics)
- Formalization is useful for tool and method developers
- Formalizing the natural-language expressions is counter-productive
- Strength: Testers can freely augment

Abstract syntax

English prose of CORAL diagram:

Do not validate msg parameter then submit entry.

The altered message \texttt{submit(name,XSSscript)} is transmitted from \texttt{GBForm} to \texttt{GBDatabase} with frequency interval \([150, 300>\) per 1y, the transmission leads to its reception with conditional ratio 0.6, and the reception occurs with frequency interval \([90, 180>\) per 1y.

Weakly sequenced by \[
\begin{array}{l}
\text{The message } \texttt{true} \text{ is transmitted from } \texttt{GBDatabase} \text{ to } \texttt{GBForm} \text{ with undefined frequency, the transmission leads to its reception with undefined conditional ratio, and the reception occurs with undefined frequency.}
\end{array}
\]

Weakly sequenced by \[
\begin{array}{l}
\text{The unwanted incident (UI1) XSS script injected in database occurs on } \texttt{GBDatabase} \text{ with frequency interval } [90, 180> \text{ per 1y, and impacts asset } \texttt{Integrity of GB Source Code} \text{ with consequence Major.}
\end{array}
\]
Conclusion

• CORAL is specifically developed for security testers to design and select security tests based on the available risk picture.

• Graphical (vs textual):
  • A single diagram covers scenarios that occurs multiple times in code
  • Captures interactions between actors in a manner not possible using code
  • Allows describing the behavior of human actors, threats, and working procedures

• CORAL (vs formal methods)
  • A mathematical semantics is useful for tool and method developers
  • Formalization of the natural-language semantics is counter productive for testers

• Sequence diagrams (vs other)
  • Isolate particular scenarios without having to consider the behavior for other scenarios (STM)
  • Class diagrams do not capture the dynamic behavior addressed by CORAL
  • Use-case diagrams may be useful to design high-level test procedures

• Risk annotations (vs tables and AT):
  • Convey risk info in the location where it belongs
  • one location conveys more than one kind of info
  • No need for a separate risk assessment language
Teknologi for et bedre samfunn