Security Testing with Fault-Models and Properties

Matthias Büchler
Technische Universität München, Germany
buechler@cs.tum.edu
Advisor: Prof. Dr. Alexander Pretschner, Project: SPaCIoS EU project

***Abstract*—Web applications are complex and face a massive amount of sophisticated attacks. Since manually testing web applications for security issues is hard and time consuming, automated testing is preferable. In model-based testing, test cases are often generated using structural criteria. Since such test cases do not directly target security properties, this Ph.D thesis proposes to use a fault model for generating tests for web applications. Faults are represented as known source code vulnerabilities that, by using respective mutation operators at the model level, are injected into models of a System Under Validation to generate "interesting" test cases. To achieve this, advantages of penetration testing are combined with model-checkers dedicated to security analysis. To find attacks on real systems the gap between an abstract attack trace output by a model-checker and a penetration test needs to be addressed. This Ph.D thesis contributes with a semi-automatic methodology to turn abstract attack traces operational.**

***Keywords*-security testing; property based testing; mutation testing; model checking; semi-automatic test execution;**

I. Preliminary Hypothesis

An "interesting" test case exploits a known vulnerability to violate a security property. They can be generated by applying semantic mutation operators on models of a System Under Validation. Semantic mutation operators bind high level security properties to source code level vulnerabilities. A semi-automatic instantiation methodology executes "interesting" test cases to find potential security vulnerabilities in a real system.

II. Introduction

Web applications are part of our lives and deal with sensitive data. Such applications usually address both declarative security properties like the CIA properties (confidentiality, integrity, availability), as well as properties of security mechanisms. Examples of security properties are: Secret values are only known by the a priori defined users to guarantee confidentiality, sensitive data can only be modified in a well-defined way to guarantee integrity. Properties of security mechanisms are: Before accessing a user profile a user has to be authenticated and authorized, admin functions are only executable by users of the group "admin", every access needs to be evaluated by the access control system. Corresponding web applications need to be tested for violations of both kind of properties. Ideally corresponding test cases are generated automatically and executed on a System Under Validation (SUV) to find attacks.

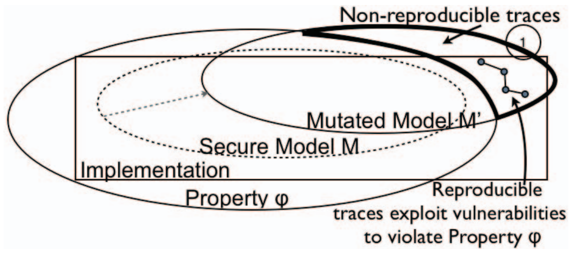

Figure 1. Characteristics of Interesting Test Cases

III. The Problem

In model-based testing, structural criteria on models are often used for test case generation. They are on a syntactic level and with a few exceptions (e.g., Fraser and Wotawa [5]) are not related to security properties due to the lack of an obvious link between structural criteria and security properties. The problem we want to address is: How can we generate "interesting" test cases that test a SUV for violations of high level security properties, under the assumption that such a violation is indeed present in the SUV? An "interesting" test case exploits a known vulnerability to violate a security property. At the model level, model checking can verify security properties ϕ and reported counter examples can be considered as abstract test cases. One can model potential violations of the security properties by mutating properties or by mutating models. If the underlying model M is secure, mutated properties are not immediately useful for test case generation since a secure model satisfies all original properties ($\forall\varphi \in \phi : M \models \varphi$). If the security properties are mutated, the model checker can only report traces from the secure model that all satisfy the initial security properties. Therefore the model M itself needs to be mutated to M' so that a trace in M' exists that violates at least one specified property ($\exists\varphi \in \phi : M' \not\models \varphi$). Model checking tools might now report traces of the mutated model that violate an initial security property $\varphi \in \phi$ (① in Figure 1). To test whether the SUV is vulnerable, such reported traces need to be instantiated and executed.

IV. Proposed Research Approach

Our approach is related to Dadeau et al. [4]. We address the question how test cases for security properties are generated from a secure model by combining penetration testing and model checking techniques. Penetration testing techniques provide knowledge about security properties and source code vulnerabilities that could be exploited by an attack to violate a security property. Source code vulnerabilities are captured by model-level mutation operators and are injected into a secure model. Model checking techniques are used to report Abstract Attack Traces (AATs) that violate the security property due to the injected vulnerability (see Figure 1). Since a single syntactic mutation operator might not be powerful enough to represent a source code vulnerability at model level, they are aggregated into semantic mutation operators. They are hence higher-order mutation operators that consider the semantic of the model as well. E.g. for access control policies, Martin and Xie [7] show evidence that structural coverage for test selection is far from optimal for fault-detection effectiveness. Using our approach, generated test cases are "interesting" (① in Figure 1) because they test for specific vulnerabilities, acknowledge the presence if the vulnerability indeed exists, or raise confidence that the vulnerability would have been found if present. Since AATs are on an abstract level, they are mapped to executable test cases to find potential attacks on the SUV. The instantiation uses a 2-step mapping to separate application-dependent from application-independent data and hence makes it reusable for multiple test cases.

V. Expected Contribution

The contribution of this Ph.D thesis is an approach for generating and executing "interesting" test cases for security properties based on vulnerabilities. To achieve that I in particular propose useful semantic mutation operators which represent a domain-specific fault model for web applications. In addition, tool support for mutating secure models to generate AATs at model level is provided. Since semantic mutation operators are higher-order mutation operators, research will be carried out to compare them with first-order syntactic mutation operators that only apply 1 small syntactic change (also see Jia and Harman [6]). For executing AATs a semi-automatic methodology is proposed to map test cases at model level to operational test cases at implementation level.

VI. Summary of results to date

Research [1, 2, 3] has already been carried out in the context of the SPaCIoS EU project. Among others, I have collected a non-exhaustive set of security properties and corresponding source code vulnerabilities. Currently, I am working on the design of semantic mutation operators and the development of a mutation tool for the ASLan++ modeling language, a formal language for specifying security-sensitive service-oriented architectures. For the instantiation of AATs I have a preliminary version of the Web Application Abstract Language, which describes abstract browser actions, together with a mapping to executable API calls. Finally, I am working on heuristics and methodologies used by the Test Execution Engine to involve test experts when necessary during instantiation and the adaptation to non-expected behavior of the SUV since the used model for test generation can be implemented in different ways.

VII. Evaluation

As an evaluation I first want to focus on the effectiveness of our approach for web applications with documented security issues since we generate test cases from known vulnerabilities. Second, I want to compare our approach with first-order mutation operators, that mutate a secure model by a single syntactic change, and with different random approaches. The random approach is both applied to secure models as well as to the mutated models. For each approach, n test cases are considered. The goal of the comparison is the evaluation how many vulnerabilities are discovered with n test cases of each approach. Finally as a proof-of-concept and use case scenario, I would like to describe how our approach is embedded in the software development process.

VIII. Conclusion

The aim of this Ph.D thesis is an approach to generate test cases for security properties of web applications. Following a model-based testing approach, a fault model represented as a set of semantic mutation operators for secure models is proposed. Semantic mutation operators are higher-order mutation operators that bind security properties and source code vulnerabilities. Model checking techniques are used to find abstract test cases that violate security properties based on the injected vulnerability. A semi-automatic test execution engine finally executes reported tests to find attacks on a System Under Validation.

References

[1] M. Büchler, J. Oudinet, and A. Pretschner, "Security mutants for property-based testing," in *TAP*, 2011, pp. 69–77.

[2] ——, "Spacite - web application testing engine," in *ICST*, G. Antoniol, A. Bertolino, and Y. Labiche, Eds. IEEE, 2012, pp. 858–859.

[3] ——, "Semi-automatic security testing of web applications from a secure model," in *SERE*. IEEE, 2012, pp. 253–262.

[4] F. Dadeau, P.-C. Héam, and R. Kheddam, "Mutation-based test generation from security protocols in HLPSL," in *ICST*, 2011, pp. 240–248.

[5] G. Fraser and F. Wotawa, "Property relevant software testing with model-checkers," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 6, pp. 1–10, Nov. 2006.

[6] Y. Jia and M. Harman, "Higher order mutation testing," *Inf. Softw. Technol.*, vol. 51, no. 10, pp. 1379–1393, Oct. 2009.

[7] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *WWW*, 2007, pp. 667–676.