Systematizing Security Test Case Planning Using Functional Requirements Phrases

Ben Smith
North Carolina State University
890 Oval Drive, Campus Box 8206
Raleigh, NC 27695
+1(859) 619-8076

ben_smith@ncsu.edu

http://bensmith.zapto.org

ABSTRACT

Security experts use their knowledge to attempt attacks on an application in an exploratory and opportunistic way in a process known as penetration testing. However, building security into a product is the responsibility of the whole team, not just the security experts who are often only involved in the final phases of testing. Through the development of a black box security test plan, software testers who are not necessarily security experts can work proactively with the developers early in the software development lifecycle. The team can then establish how security will be evaluated such that the product can be designed and implemented with security in mind. *The goal of this research is to improve the security of applications by introducing a methodology that uses the software system's requirements specification statements to systematically generate a set of black box security tests.* We used our methodology on a public requirements specification to create 137 tests and executed these tests on five electronic health record systems. The tests revealed 253 successful attacks on these five systems, which are used to manage the clinical records for approximately 59 million patients, collectively. If non-expert testers can surface the more common vulnerabilities present in an application, security experts can attempt more devious, novel attacks.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Verification, Documentation

Keywords

security, testing, verification, vulnerabilities, requirements

1. INTRODUCTION

Security experts use their knowledge to attempt attacks on an application in an exploratory and opportunistic way in a process known as penetration testing. Penetration testing and similar techniques require the security expert's knowledge to be effective [1]. For example, a security tester might browse through a web application, find a form, and submit several attacks to test that the system properly validates input. She will use the knowledge of successful attacks to drive her next attempt. A software tester with no security training would lack the knowledge to pursue defects the same way an expert does.

Due to time and resource constraints, building security into a product must be the responsibility of the whole team, not just the security experts who are often only involved in the final phases of testing [3]. Through the development of a black box *security* test plan that is based on functional requirements specifications, software testers who are not necessarily security experts can work proactively with the developers early in the software development lifecycle. The team can then establish how security will be evaluated such that the product can be designed and implemented with security in mind.

The goal of this research is to improve the security of applications by introducing a methodology that uses the software system's requirements specification statements to systematically generate a set of black box security tests. We evaluated our methodology by using a requirements specification[1] to create a black box security test plan for four open source and one proprietary electronic health record (EHR) systems. We executed the resultant test cases on these five released EHR systems that are currently used to manage the records of over 59 million patients: OpenEMR[2], ProprietaryMed[3], WorldVistA[4], Tolven[5], and PatientOS[6].

2. PROPOSED SOLUTION

This section provides our methodology for developing software security tests at the application level based on a functional requirements specification. More information on our methodology and results can be found on our healthcare wiki[7].

The structure of the requirements statement, as well as certain keywords, can help guide the tester to construct an appropriate

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Copyright is held by the author/owner(s).
ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA.
Copyright 2011 ACM 978-1-4503-0445-0/11/05….$10.00.

[1] http://www.cchit.org
[2] http://oemr.org/
[3] ProprietaryMed was developed by an organization that wishes to keep the identity of their product confidential.
[4] http://worldvista.org/
[5] http://tolven.org/
[6] http://patientos.org
[7] http://realsearchgroup.com/healthcare/

type of test. The CWE/SANS Top 25[8] lists the most dangerous security programming errors based on prevalence and potential consequences. Our methodology includes six test types, each of which can uncover one or more common vulnerabilities from the Top 25. We demonstrate the methodology in this paper using functional requirements statements (i.e. functional requirements [5]), but our methodology does not rely on requirements to be provided in "shall" format: as long as the key phrases can be identified, our methodology is applicable.

Our methodology uses *key phrases* and *supporting information* in a requirements statement to determine the type of security test that will most likely reveal vulnerabilities in the system. In these examples, the first phrase that the tester comes to after reading "The system shall provide the ability to…" contains the key action phrase and is followed by the key object phrase. We call these phrases *key* because they define the functionality the system has with respect to its environment.

Consider this requirement, known as AM 02.04: "The system shall provide the ability to modify demographic information about the patient." In AM 02.04, the phrase *modify* is the key action phrase, and *demographic information about the patient* is the key object phrase. There is no supporting information for this requirement. This requirement signals the need for Force Exposure, Input Validation Vulnerability, and Audit test case types. This key action phrase indicates that an attacker has the opportunity to input malicious strings that can take the form of a cross-site scripting [4], SQL injection [2] or many other input validation vulnerabilities. These attacks, if properly executed, have the potential to tamper with or reveal information from the *demographic information* object. Input Validation Tests, which our methodology includes, will attempt to tamper with or reveal information from the demographic information object.

Requirements statements like AM 02.04 typically conform to the following format: "*The system shall provide the ability to <action> a <object> <and/with/in supporting information>*." The object in these statements is most often a data store, such as a listing of users or a report regarding multiple data records for output. The action in these statements is typically an action that the system will perform on that data store, such as store, graph, view, print, or edit. The supporting information in these statements provides additional information as to how, or when the system should achieve the action. Sometimes the supporting information is a prepositional phrase in the same sentence or can extend to an additional sentence.

3. PROGRESS AND EVALUATION

Using the CCHIT requirements, we systematically developed a black box security test plan consisting of 137 tests. Overall, our test plan launched 253 successful attacks in the five EHR systems, averaging 50 failures per EHR. The failures consisted of both implementation-level defects, such as cross-site scripting, and design-level issues, such as the lack of encryption on the backup copy of system data. We developed the security test plan in approximately 60 person hours. Executing the test plan manually on each of the case study subjects consumed approximately six to eight person hours per project. We also alerted developers to the vulnerabilities we found by posting them to respective healthcare IT communities' bug report pages.

[8] http://cwe.mitre.org/top25

4. RESEARCH PLAN

We will conduct our methodology on requirements specifications from other domains. This analysis will help us understand the weaknesses in our existing test case types as well as evaluate the effectiveness of our methodology in revealing security issues in software systems that have been developed outside the realm of health care. We will also create a tool that uses natural language processing to automatically generate black box tests based on rules and patterns similar to those in these case studies.

Also, we will compare our methodology with existing automated techniques to see what security issues a test plan developed using our technique may miss that automated static analysis or automated security scanners may detect. We are also hoping to discover how our technique may point developers and security experts to more high-level issues in the design of a system that can result in security weaknesses. A cross-comparison of security issues revealed by our methodology and those revealed by other security evaluation techniques will provide a wealth of information for both types of security analysis.

Finally, we will investigate how well our methodology could help identify requirements that are unclear. Writing test cases for each requirement as it is elicited will help ensure that requirements statements are testable as well as unambiguous. Further, the discussion of security during requirements elicitation could help prevent the team from missing essential requirements or introducing requirements that produce security problems. We will evaluate our methodology by working with an industrial software development team and incorporating our methodology during the requirements elicitation phase.

5. ACKNOWLEDGMENTS

The National Science Foundation under CAREER Grant No. 0346903 supports this work. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is also supported by the United States Agency for Healthcare Research Quality. Additionally, this work was supposed by an IBM PhD Fellowship.

6. REFERENCES

[1] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security & Privacy,* vol. 3, no. 1, pp. 84-87, 2005.

[2] W. Halfond, and A. Orso, "AMNESIA: Analysis and monitoring for NEutralizing SQL injection attacks," in International Conference on Automated Software Engineering, Long Beach, CA, 2005, pp. 174-183.

[3] S. Lipner, "The Trustworthy Computing Security Development Lifecycle," in 20th Computer Security Applications Conference, Tuscon, Arizona, 2004, pp. 2-13.

[4] G. Wassermann, and Z. Su, "Static detection of cross-site scripting vulnerabilities," in International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 171-180.

[5] K. E. Wiegers, *Software Requirements, 2nd Edition*, Redmond, WA: Microsoft Press, 2003.