Evolutionary Security Testing of Web Applications (Fast Abstract)

Xiang Fu
Dept. of Computer Science
Hofstra University
Hempstead, NY 11549
Xiang.Fu@hofstra.edu

Kai Qian
School of Computer and Software Engineering
Southern Polytechnic State University
Marietta, GA 30060
kqian@spsu.edu

***Abstract*—Complexity of modern web applications usually leads to low coverage of test cases which are designed manually. This paper proposes an evolutionary testing strategy that automatically synthesizes test cases for penetrating web applications, based on an initial set of data on user interaction sessions.**

I. Introduction

Unlike system softwares which often go through a formal modeling, testing and verification process, web applications are usually developed by application programmers under market pressure. It is estimated by WSAC recently that 99% of web applications world wide do not comply with PCI DSS Standard. $80-96\%$ of them have high risk level vulnerabilities, and 13% can be compromised automatically. The high frequency of vulnerabilities in web applications makes testing (rather than formal model checking) a more productive solution in identifying security vulnerabilities.

On the other hand, static analysis is very expressive and powerful in finding potential vulnerabilities. For example, Su et al. uses forward string analysis for identifying the existence of SQL injection [5]. This is further extended by Kiezun et al. in backward string analysis that generates precise attack signatures [3]. In test case generation, symbolic execution also exhibits more versatility in generating test input that guarantees higher branch and statement coverage. The static analysis method, however, is limited by the strength of the back-end constraint solver and usually does not scale well.

We propose the use of evolutionary testing (see survey by Harman et al. [2]), assisted by light weight static analysis, for improving the test coverage of web applications and discovering security vulnerabilities. Starting from a collection of user sessions, recorded by a web server proxy, we construct an initial test suite for security testing of a web application. Then by applying evolutionary testing operations, e.g., cross over and mutation, the test suite is optimized for its size, code coverage, and detection of vulnerabilities. The framework (code name GATWEB) has a general fitness function, which is further customized for each of the popular web security attacks. GATWEB is currently under development. We would like to compare it with existing work, e.g., session based testing of web applications [4], and evaluate the effectiveness of the proposed approach.

II. Motivating Example

We motivate our technique using one fictitious example called `Shopping Cart`, which consists of four PHP servlets: (1) collecting shipping address, (2) uploading credit card information, (3) reviewing receipts, and (4) checking out. A user session is maintained using cookies.

The `Shopping Cart` example suffers from several attacks due to the weak user input validation at server side. It is possible for an attacker to invoke step 1 and directly jump to step 4, without entering the credit card information. This is usually called the *workflow attack*. In addition, replaying step 4 can cause multiple charges on the same credit card.

Evolutionary testing technique has a certain advantage in discovering such attacks. If we regard a user session as a sequence of HTTP requests and responses, an attack can be generated by applying cross-over and mutation operators on existing user session data. Compared with systematic or random search, this approach is more flexible in practice. For example, it is not realistic to apply random search in generating username/password pairs, but evolutionary testing is not restricted in such applications.

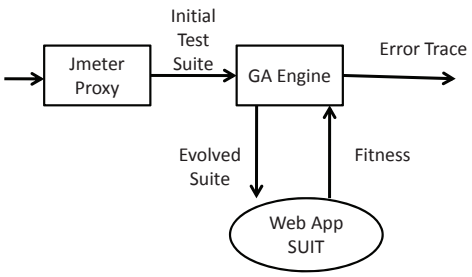

Figure 1. GATWEB Framework

III. GATWEB Framework

The GATWEB framework (shown in Figure 1) leverages a web application load testing tool called JMeter for collecting user session data. The efficiency of test suite evolution depends on three key factors: chromosome representation, mutation/cross-over operators, and fitness function.

A. Chromosome Representation

Naturally, an HTTP session is a sequence of request/response pairs. Each request is further equipped with a collection of parameters. In GATWEB, a *chromosome* is a sorted array of HTTP sessions. Then the *distance* between two chromosomes can be defined, with many options available, e.g., raw text comparator and tree structure comparators that are frequently used in data mining.

B. Cross Over and Mutation Operators

We use the one point cross over operator and the algorithm is straight forward: Let C_1 and C_2 be two chromosomes (collections of sessions), a cross over operator is defined to randomly merge the sessions from C_1 and C_2. As usual, its purpose is to propagate effective mutations faster into the population.

There are many mutation operators available, and the following is a typical set: (1) deletion operator which randomly removes a session from a chromosome, or an HTTP request from a session, or a parameter from a request; (2) insertion operator which randomly inserts sessions, requests, and parameters; and (3) value change operators which randomly change the value of parameters.

Light Weight Static Analysis: We use static analysis and best effort constraint solving for generating constant values that satisfy or discharge branch conditions. These values are collected in a global data store for the insertion and value change operators.

IV. Fitness Function

Fitness functions vary according to the purpose of testing needs. In the following we briefly introduce several types of them.

A. Test Suite Reduction

Given a pair (a, b) and a natural number N, where $a \in [0, N]$ and $b \in [0, \infty]$. A function $\varphi(a, b)$ is defined as below:

$$\varphi(a, b) = (a + 1/b)/N$$

Given a web application, let N be its size (lines of code). For a chromosome C, let $coverage(C)$ be the total number of statements that are covered by chromosome C. $size(C)$ is the total number of HTTP requests in C.

The fitness function for test suite reduction is defined as below:

$$f_1(C) = \varphi(coverage(C), size(C))$$

Clearly, f_1 measures the quality of a chromosome in two ways: (1) the coverage achieved by the chromosome and (2) the size of the chromosome. When evaluating two chromosomes, the coverage metric has the priority.

B. Workflow Attack

Given a servlet $\mathcal{S}$ (usually represented by its URL r), let $link(r)$ be the set of URLs that can be generated by $\mathcal{S}$. Although it is undecidable to compute $link(r)$, an over-approximation can be constructed using forward string analysis. Let it be $olink(r)$.

Definition 4.1: A sequence of HTTP requests $r_1 r_2 ... r_n r_{n+1}$ is a workflow attack *if $r_{n+1} \notin olink(r_n)$.* ∎

The fitness function for discovering workflow attacks is defined as below:

$$f_2(C) = \begin{cases} 1 & \exists s \in C\text{: } s \text{ is a workflow attack} \\ f_1(C) & o.t. \end{cases}$$

The fitness function first guides the search to achieve coverage measure, while at the same time tries to find a workflow attack. Our intuition is that a good test coverage helps to discover vulnerabilities. The fitness function for cross-site scripting attacks can be defined similarly.

C. SQL Injection Vulnerabilities

The problem can be reduced to a string constraint solving problem, as shown in [1], [3]. Current work only provides solution for restricted forms of string equations. To solve more complex equations needs exploration of a huge solution space. We are interested in developing the fitness function that guides the search process.

V. Conclusion

This paper states our current position in automatic discovery of web security vulnerabilities using mutation and session based testing. We plan to complete the implementation of GATWEB and collect data for analyzing the performance of the proposed approach.

References

[1] X. Fu and C.C. Li. A string constraint solver for detecting web application vulnerability. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, pages 535–542, 2010.

[2] M. Harman, S. A. Mansouri, and Y. Zhang. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-09-03.pdf.

[3] Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. Hampi: A solver for string constraints. In *Proc. of 2009 International Symposium on Testing and Analysis (ISSTA'09)*, 2009.

[4] Sara Sprenkle, Emily Gibson, Sreedevi Sampath, and Lori Pollock. Automated replay and failure detection for web applications. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 253–262, 2005.

[5] G. Wassermann and Z. Su. Sound and precise analysis of web applications for injection vulnerabilities. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation*, pages 32–41, 2007.