# An Early Testing and Defense Web Application Framework for Malicious Input Attacks

Michael Gegick, Laurie Williams
North Carolina State University, Department of Computer Science, NC 27695
{mcgegick, lawilli3}@ncsu.edu

## Abstract

*We introduce a Java **W**eb **A**pplication **R**eliability and **D**efense (WARD) framework, a two-part security solution composed of a vulnerability detection component, SecureUnit, and a vulnerability protection component, SecureFilter. SecureUnit enables developers to write automated, reusable, and customizable JUnit tests that launch attacks on their systems to reveal security vulnerabilities. SecureFilter is a customizable server-side choke point containing a regular expression-based filter to match legal input according to system requirements. We integrated WARD v2.0 with WebGoat, an open-source web application security test bed, and successfully "warded off" 41 of 43 (95%) injected cross-site scripting (XSS) exploits. The exploits cover the seven classifications of XSS in the Common Weakness Enumeration that are possible in a J2EE environment.*

## 1. Introduction

Input validation vulnerabilities, one of the largest problems in software security today [6], are readily identified by software assurance (SA) tools. Input validation problems are weaknesses in a software system where developers trust that input is benign [3]. Software development organizations are increasingly adopting SA tools to quickly identify these vulnerabilities and others in their software systems. These tools, usually applied when implementation is complete, have a comprehensive and extendible rule set to detect known and new vulnerabilities [5]. However, addressing problems late in the software process is more costly and less effective than a proactive approach where developers can build security into their software [2]. A simple and effective framework is needed to provide developers with a means to secure against malicious input attacks early in software development. *Our research objective is to provide developers with the combined ability of attacking (via testing) and defending their web applications from invalid input early in the software process*. Approaching software security[1] with a two-sided effort is captured by:

*…[there are] two sides of software security – attack and defense, exploiting and designing, breaking and building--into a coherent whole. Like the yin and the yang, software security requires a careful balance.* [5]

Preventing malicious input from entering a software system can be achieved with filtering against a white list[2]. Inputs to software products are likely to be unique due to the differing requirements specifications and thus a predefined/universal security mechanism is not an accurate solution for validating input. To provide developers with a lightweight approach to white listing and security testing with a black list[3], we introduce the Java **W**eb **A**pplication **R**eliability and **D**efense (WARD) framework. WARD[4] is a two-part security solution composed of a vulnerability detection component, SecureUnit, and a vulnerability protection component, SecureFilter. To examine the effectiveness of WARD at preventing attacks, we integrated WARD v2.0 with WebGoat[5], an open-source web application security test bed. We injected the explicit XSS examples documented in [4] into our SecureUnit to determine the efficacy of filtering input achieved by SecureFilter.

## 2. Web Application Reliability and Defense (WARD) Framework

WARD evolves the *test-then-code* paradigm from test-driven development (TDD) [1] to an *attack-then-defend* approach to build security into a software system early in the software process. SecureUnit is a black-box testing tool that identifies input validation vulnerabilities in a Web application. To support the black-box testing aspect of the web application environment in which WARD resides, we employ HttpUnit, a Java-based black box testing framework. HttpUnit emulates browser functionality such as form submissions, cookies, basic authentication, and JavaScript. JUnit uses HttpUnit as a test fixture[6] to handle tests that are submitted to a Web application and the responses sent back by the server. Each test contains an

[1] The practice of building security into the software [3]

[2] A white list defines (e.g. with a regular expression) valid input as specified by the software system requirements. [3]

[3] A black list defines (e.g. with a regular expression) any input not defined as valid by the software system requirements. [3]

[4] http://ward.sourceforge.net

[5] http://suif.stanford.edu/~livshits/securibench/

[6] The set of objects that a test is run against (http://junit.sourceforge.net/doc/cookbook/cookbook.htm)

exploit that is automatically read from either a default list of exploits, an exploit library (EL), in SecureUnit or a modified/new EL created by the developer. Because SecureUnit is based on the JUnit framework, developers can perform automated regression tests when application code is modified.

SecureFilter is an implementation of the J2EE `Filter` Interface, an API that filters HTTP requests and/or responses to/from a Java servlet container (e.g. Apache Tomcat). Matching input against the allowable formats embodied in white list is more feasible than matching input against a potentially-infinite black list [3, 4]. As a result, SecureFilter is white list-centric. Incorporating SecureFilter into a Web application requires only that a developer add the SecureFilter JAR file to the WEB-INF/lib directory and modify their web.xml file so the servlet container routes requests to SecureFilter. The developer's main responsibility with SecureFilter is to strive to maintain a well-formed white list that strictly adheres to the requirements; a black list provides a second check to the input, but is not intended to be a surrogate for the white list. The filtering is provided by SecureFilter and occurs outside of the developer's code. SecureFilter takes as input HTTP headers, cookies and parameters, decodes the data, then passes the data through a white list and a black list. Input is permitted into the Web application when the data are matched by the white list and do not match the EL, otherwise a 403 SC_FORBIDDEN response is returned.

## 3. Illustrative Example

To test WARD, we required a test bed that could serve as a realistic web application. We chose Stanford SecuriBench .91a, an open source suite of software applications that are designed for running static and dynamic tests. These test beds are vulnerable to XSS, SQL injection, HTTP splitting and path traversal attacks. Specifically, we demonstrate the use of SecureUnit on WebGoat 3.7, a J2EE application implemented in part to demonstrate XSS attacks. WebGoat is an educational-based program in that there are security lessons in the program that teach one how to attack the system. We required only the binary release of WebGoat because the WARD framework is not invasive to the source code.

We configured WARD to use a default white list to allow all data to enter WebGoat. The white list (and black list) were both implemented using regular expressions. A "wide open" or faulty white list demonstrates that WARD can still alert developers that a vulnerability may be present. We created a black list containing 12 regular expressions based on the 43 explicitly documented XSS exploits in [4]. We assumed that each exploit would be malicious to WebGoat if it was a live application released into the field.

We wrote an HttpUnit test fixture to log into WebGoat (using basic authentication), browse to a Web page that contained a vulnerable HTML form which used a POST method to submit data to a servlet container (Tomcat 5.5). Using SecureUnit, we injected 43 XSS exploits from [4] into the form and tested the response from the server. The 43 exploits cover the seven classifications of XSS in the Common Weakness Enumeration[7] possible in the J2EE environment. An eighth classification, CWE - Invalid Characters in Identifiers, currently only applies to PHP. Results indicate that 41 of 43 (95%) XSS vulnerabilities were "warded off" when matched against the black list.

## 4. Summary and Conclusion

The goal of our research is to produce a simple and effective framework for developers to secure their software from input validation vulnerabilities in an attack-then-defend software security approach. We hypothesize that if developers adopt a security-oriented TDD process, then they will have a more effective and less costly means to practice due diligence. The WARD framework affords developers with the ability to attack their systems to identify malicious input vulnerabilities with SecureUnit.

### Acknowledgements

We would like to thank the North Carolina State University Software Engineering Realsearch reading group for their helpful suggestions on this paper. We would also like to thank Eric Isakson for contributing to the implementation of WARD. This research was supported by the National Science Foundation.

### References

[1] Beck, K., *Test-Driven Development -- by Example*. Boston: Addison Wesley, 2003.

[2] Boehm, B., "Industrial Metrics Top 10 List," *IEEE Software*, vol. 4, pp. 84-85, 1987.

[3] Hoglund, G. and G. McGraw, *Exploiting Software*. Boston: Addison-Wesley, 2004.

[4] Howard, M. and D. LeBlanc, *Writing Secure Code*, 2nd ed. Redmond: Microsoft Corporation, 2003.

[5] McGraw, G., *Software Security: Building Security In*. Boston: Addison-Wesley, 2006.

[6] Tsipenyui, K., B. Chess, and G. McGraw, "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors," presented at Automated Software Engineering, Long Beach, CA, 2005.

[7] http://cwe.mitre.org