



Secologic

Open Source Static Analysis Tools for Security Testing of Java Web Applications

An Analysis

Document Version 1.0 – Dec. 2006



These materials are provided by SAP AG for informational purposes, without representation or warranty of any kind. These material was created in the context of the project “*Secologic*”, a public funded project of the German Ministry of Economy and Technology (BMWi). We are grateful to the BMWi for supporting this project. For further information have a look at www.secologic.de.

Contents

<i>Clarification</i>	4
<i>Target Group</i>	4
<i>Purpose</i>	4
<i>Related work</i>	5
<i>Document organization</i>	5
Executive summary	6
Basics	7
Tool Overview	7
<i>Scientific Prototypes</i>	9
<i>Open Source Bug Pattern Detectors</i>	9
FindBugs	10
Conclusion	11
References and Resources	12

Clarification

The focus of our analysis, which is summarized in this document, was to evaluate the use of Open Source Java test tools in the context of Java web application security. Most of the tools discussed here are not designed for this purpose. Thus, we want to emphasize that the tools listed and discussed in this document should not be considered inferior or inappropriate. We do not assess the value of any tool in general or in comparison with others. Most of the tools discussed might be suitable for their intended purpose and application context.

Target Group

The target group of this documentation should have a fundamental understanding of the security problems of web applications. Please find corresponding references in the Related work section. Moreover, a basic understanding of static analysis tools for Java is necessary, since we only provide a brief introduction to the section 'basics of static analyzers' in section Executive summary.

Purpose

This document summarizes our analysis of Open Source tools which we have selected and examined to evaluate their characteristics for enabling security tests for Java web applications. In the context of our work, we focused on static analysis tools only. Thus, in the following we always refer to Java and static analysis when we speak about testing tools. The following material and references should be useful for Java software developers with security concerns.

Please note that we present the current status of our analysis in July 2006.

The most important result of our analysis is the fact that there are no Open Source tools for static analysis with sufficient support for security tests. Even though there are several commercial¹ tools in this sector, such as Fortify Tools [5], CodeAssure [6], or Coverity Prevent [7], Open Source projects simply do not provide corresponding bug detectors and rules for security. Consequently, we focused on the extensibility of Open Source tools for our analysis. Our purpose was to examine whether the selected tools provide sufficient methods for implementing security tests. Please note, when we speak about Open Source tools, we refer to all projects which provide their tools and their sources freely to the public.

Although there are no Open Source static analyzers which provide sufficient support for security tests, this analysis focused on Open Source tools, because these are cost-cutting and freely available².

¹ The costs for these tools range from four-digit to five-digit EURO sums.

² Naturally, Open Source tools often have more or less serious limitations, such as bad or non-existent documentation. However, such issues were not the focus of our analysis.

Related work

This document continues our effort in the area of Java web application security. In [1] we provide best practices to deal with the following topics:

- Code Injection
- Cookie Security
- Cross Site Scripting
- Information Disclosure
- Missing Input Validation
- Resource Tampering
- SQL Code Injection
- Unreleased Resources

For definitions and detailed descriptions of each topic see [1]. Thus, the analysis provided in this document aimed to evaluate the features of each tool with respect to the topics listed above. Our original intention was to provide automatic testing procedures for the best practices of [1]. As practice showed, this is not possible with Open Source tools used out-of-the-box.

As far as we know, our analysis is the first analysis of this kind. However, there are several analyses of bug finding tools for Java, such as [8] which also provided interesting references for us.

Document organization

This document is organized as follows: First, we present the main results of our analysis (Executive summary). In the second section, we explain fundamental aspects related to the use of static analysis tools for security tests (Basics). This is followed by a description of the tools which we examined (Tool Overview). Then, we briefly summarize our most important results (Scientific Prototypes / Open Source Bug Pattern Detectors). In the next section, we look at the tool FindBugs in greater detail (FindBugs). Finally, we make concluding remarks about our analysis (Conclusion).

Executive summary

Since the original intention of our analysis was to provide automatic testing procedures for the best practices of [1], we developed a test suite which contains corresponding violations of the best practices. The purpose of the test suite was to apply the Open Source tools to it and to evaluate the detected security vulnerabilities. However, practice showed that the Open Source tools (static analyzers for Java) which we deployed could not be used for this kind of evaluation. Nevertheless, the test suite supported us during our evaluation process, although most of the tools do not provide tests for the best practices. The test suite helped us, for instance, to determine whether corresponding tests could be realized with a certain tool.

Thus, as already mentioned, the main result of our analysis was that there is no tool for our purpose, which is to provide automatic security tests for the best practices of Java web application security [1]. This is probably due to the fact that it is very hard for Open Source projects to provide a large amount of detectors for security bugs, as is done by commercial software [5] [6] [7]. It is hard for Open Source and mostly non-commercial projects to accomplish this, because at least three fundamental steps have to be realized in a permanent process for make such large security detector databases available:

- Discovering new possible and concrete software bugs on security
- Writing corresponding bug detectors
- Verifying the completeness and reliability of detectors which are published

The tools which have been evaluated during our analysis can be divided into two categories:

- Open Source Bug Pattern Detectors
- Scientific Prototypes

There are many Open Source projects which provide static analyzers for Java. However, these usually do not focus on web application security. On the other hand, the category of scientific prototypes partially provides tools which focus on certain security topics.

Thus, we evaluated all the tools we discovered with respect to their features for detecting security vulnerabilities in the context of our best practices for Java web applications [1]. Our second purpose was to examine the extensibility of all tools – even of those which are not designed for security tests – to evaluate whether they include features which are useful for security.

For instance, in the case of security tests, the ability to perform dataflow analysis and tracking is an important feature of static analyzers. However, for many bug detectors, this ability is of little or no relevance. For the creation of security vulnerability detectors, dataflow analysis is highly relevant. We can state that – in accordance with the basic intention of Open Source projects – all tools examined are extensible. However, extending the tools is a demanding task and usually has to be performed by application developers.

Basics

In this section, we briefly present several basic aspects of static analysis for security which are important for the understanding of our analysis. We do not deal with the fundamentals of static analysis here. The Department of Homeland Security (USA) provides a short introduction on static analysis focusing on security in [2]. Other theoretical basics about the Java byte code can be found in [19].

Static analysis – as the term “static” suggests – is a program analysis which is not performed during application runtime (such as dynamic tests), but when the application is on downtime. Fundamentally, static analysis on Java applications can be applied on the application’s source code and/or Java byte code³. For instance, some tools construct an Abstract Syntax Tree (AST) by parsing the Java source code and then performing an individual analysis of the application by traversing this tree, using specific algorithms.

However, it seems to us that the most interesting tools for our purpose follow approaches which operate on Java byte code. Furthermore, it seems that dataflow analyses might be realized more easily on Java byte code instead of source code. Dataflow analysis is an important key feature for realizing static security tests for Java applications. Tracking the dataflow of certain variables – which, for instance, might contain input data from users – enables the detection of certain insecure code constructs which might lead to vulnerabilities, like SQL injection, etc.

Most evaluated tools use the Byte Code Engineering Library (BCEL) of the Apache Jakarta Project [10] for byte code operations. BCEL can be used to parse an application’s byte code and also to reengineer applications on the basis of existing byte code. We want to emphasize that BCEL does not provide dataflow analysis methods, because it is simply not intended to do so. If a tool is able to perform dataflow analysis and uses BCEL, the tool itself provides methods for dataflow analyses (as FindBugs [9] does, for instance). For more information on BCEL please refer to the small manual [11]. This also contains a brief introduction to Java byte code.

Tool Overview

In our search for tools, we used several sources. Besides our personal experience and the well-known search options of the Internet, we used existing material and tools listed in [2] [3] [4] [8] [13]. Since our focus is static analysis tools for the Java platform, we also examined plug-ins for Eclipse [23], because Eclipse is a widely used Java development environment. [4] contains a list of static analyzers for Java which are realized as Eclipse plug-ins. [2] is a document of the Department of Homeland Security (USA) and also contains, like [13], references to static analyzers for other platforms and programming languages, such as C.

We want to point out that many tools which are called static analyzers are unsuitable for our purpose in advance. Since static analysis can be used for several services, many tools which fulfill these services do not qualify for use as security testing tools.

³ Byte code is generated by a Java compiler applied on Java sources and represented in class files. The corresponding system’s Java runtime environment or Java engine interprets the byte code and thus executes an Java application.

The following static analysis tools turned out to be unsuitable for our purpose: DoctorJ (checks existing Javadoc against an application's sources), JDepend (gives stochastic information), and CheckStyle (a code style verifier). Thus, in the following discussion, we do not mention all static analyzers which we have examined.

The following table provides an overview of the static analysis tools for Java which we examined in more detail. The table contains each Open Source project's name and corresponding web address where more information on the respective tool and the tool's download is available. We divided the tools examined into two categories (types): Open Source Bug Pattern Detectors and Scientific Prototypes. We think this is appropriate, because often scientific prototypes are not developed any further as they are usually only a proof-of-concept, although some Open Source projects are based on former scientific prototypes. The table shows our assessment of the suitability of the included security detectors for our purpose. In the table, we distinguish between security detectors for the best practices [1] and other security detectors. Some tools provide security detectors for other purposes, such as checking the correct usage of Java's Security Manager. However, we did not evaluate the quality of these kinds of security detectors nor did we examine them any further.

Name	Type	Security Detectors		Homepage/Download
		Best Practices	Other	
bddbddb ⁴	B	No	No	http://bdbddb.sourceforge.net/
ESC/Java ⁵	A	No	Unknown	http://research.compaq.com/SRC/esc
FEAT ⁶	A	No	Unknown	http://www.cs.ubc.ca/labs/spl/projects/feat
FindBugs	A	Partly	Yes	http://findbugs.sourceforge.net
Hammurapi	A	No	Yes	http://www.hammurapi.biz
JCSC ⁷	A	No	No	http://jcsc.sourceforge.net
Jlint	A	No	Unknown	http://jlint.sourceforge.net
Julia ⁸	B	No	No	http://profs.sci.univr.it/~spoto/julia/
OWASP CodeSpy	A	No	Yes	http://prdownloads.sourceforge.net/owasp/owasp-codespy.zip
PMD	A	No	No	http://pmd.sourceforge.net
QJ-Pro	A	No	No	http://qjpro.sourceforge.net

Legend: **Type A:** Open Source Bug Pattern Detector; **Type B:** Scientific Prototype

⁴ BDD-Based Deductive DataBase

⁵ Extended Static Checker for Java

⁶ Feature Exploration and Analysis Tool

⁷ Java Coding Standard Checker

⁸ Java Universal Interpreter through Abstraction

The respective tool's homepage and the material provided in [2] [3] [4] [8] [13] contain further information on each tool. [2] [3] [4] [8] [13] might contain other tools in the future.

In the following, we give some important information on the tools which emerged from our analysis.

Scientific Prototypes

A very interesting scientific approach for static security tests is PQL, a so-called Program Query Language for Java. The authors of PQL showed in several papers, such as [15], how to use PQL for defining security tests for Java applications – which might be done very easily. PQL is related to the static analysis tool bddbldb [20] which works on the basis of Datalog⁹. A front-end for bddbldb, bddshell [21], also exists. However, although the authors discuss how to convert PQL instructions into Datalog in [15], we were not able to use PQL for static tests, as no suitable tool for converting PQL into Datalog could be found. A usable PQL software package exists which can be downloaded from [22] and is independent of bddbldb. This package only provides the possibility to perform tests based on PQL during application runtime. Thus, dynamic security tests for Java applications can be realized with PQL. Other tools, like Julia [16], do not seem to have the same potential as PQL. Please note that all tools related to PQL do not include a kind of library of security detectors. All security tests based on PQL have to be developed with individual effort.

Other approaches to static analysis work closer to the exploration of the Java code. BAT/Magellan [17], for instance, transforms several pieces of information of an application, such as its code or property files, into a uniform XML database. Thus, it enables query-based parsing and exploring of the Java application by applying XQuery¹⁰ to the created XML database. However, BAT/Magellan does not seem to have the same potential for security tests as PQL.

There are several other approaches which refer to other platforms and present interesting ideas, but do not provide a public prototype. [18], for instance, presents several interesting ideas on the basis of C.

Open Source Bug Pattern Detectors

Many of the Open Source projects listed above are available as standalone tools and as Eclipse plug-ins, such as FindBugs [9]. Some tools are only available as an Eclipse plug-in, such as FEAT. Thus, if developers do not use Eclipse, they have to determine whether a tool is available in a standalone version.

We consider the tool FindBugs as the most suitable for our purpose of providing static security tests for the best practices on web applications, which we presented in [1]. However, we want to emphasize that commercial tools (such as [5] [6] [7]) provide much more sophisticated possibilities. Moreover, FindBugs covers only two out of eight security categories which we have defined in [1]. For more details on FindBugs, we refer to section *FindBugs* below. We also refer to [8], which is a scientific comparison of several bug finding tools, including FindBugs.

⁹ <http://en.wikipedia.org/wiki/Datalog>

¹⁰ <http://www.w3.org/TR/xquery/>

From our point of view, all the other tools listed in the table above are lacking in several issues, in comparison to FindBugs. One of the most important such issues is the degree of extensibility. Most Open Source tools seem to be difficult to extend. No matter whether building extensions is possible on the basis of an Abstract Syntax Tree (AST), or on the basis of byte code, extensions always have to be realized by experienced developers. The commercial tools mentioned in [5] [6] [7], for instance, usually provide an easy-to-use front-end for the individual extension of their rule databases, and thus their security detectors.

Another important feature lacking in other tools, in comparison to FindBugs, was the platform support. Jlint, for instance, does not support the analysis of Java Servlets. Obviously, the support for Servlets is essential to us, because we want to build security tests for Java web applications. Furthermore, Jlint is written in C. Thus, new test methods for Jlint usually have to be developed in C and might not be realizable in Java.

The tools listed above which provide other security detectors are considered unsuitable for our specific security efforts. CodeSpy, for instance, checks for Java security conventions, such as dealing with protected fields or signed code. Although this is an important feature for static security tests, CodeSpy does not offer ways of realizing our security efforts with regard to the best practices [1].

Finally, we want to refer briefly to other known tools which provide security tests for platforms other than Java. The basic principles of these tools might be of interest for some readers. Please note that we do not give an assessment of the quality of these tools which provide tests for finding a variety of security vulnerabilities: FlawFinder¹¹ (C/C++), RATS¹² (C, C++, Python, Perl, PHP), and Splint¹³ (C).

FindBugs

In the context of Open Source tools, FindBugs is the tool that we consider most suitable for our purpose, because it provides the best possibilities to write security detectors for the best practices of [1]. Nevertheless, writing these detectors is a very complex process. In [12], an IBM developer describes exemplarily how to extend FindBugs.

The most surprising result which emerged from our analysis of FindBugs was that FindBugs found all security flaws in two out of eight categories of the best practices inside our test application. These two categories were:

- SQL Code Injection
- Unreleased Resources

These results were obtained by using FindBugs out-of-the-box. We want to highlight that our category of Cookie Security [1] includes SQL injection vulnerabilities which are based on cookies. Thus, our test application to which we applied FindBugs also includes SQL injection vulnerabilities which can be exploited via cookies. FindBugs does not detect this kind of SQL injection vulnerability. Thus, it seems that FindBugs'

¹¹ <http://www.dwheeler.com/flawfinder>

¹² http://www.securesoftware.com/resources/download_rats.html

¹³ <http://www.splint.org>

dataflow analysis of its SQL injection detector does not recognize `javax.servlet.http.Cookie` as a potentially malicious data source.¹⁴

Building new security tests in FindBugs has to be done by writing a detector in Java code. However, this detector has to be developed close to the Java byte code of the pieces of code which must be identified as vulnerable. This applies also to the Eclipse version of FindBugs. It seems that writing an XSS detector, for instance, in this way is very complicated.

It has to be emphasized that FindBugs uses BCEL for its byte code inspections. FindBugs extends BCEL to dataflow and control flow features. Thus, it performs dataflow analyses on a method using a control flow graph. As mentioned above, dataflow analysis is extremely important for creating security tests for the best practices.

FindBugs was created as a scientific project of the University of Maryland. For more information on FindBugs, see papers about FindBugs, such as [14], or the FindBugs homepage [9].

Conclusion

Building static security tests for Java web application security [1] on the basis of Open Source tools is very difficult. Our analysis shows that there is no tool which provides sufficient security tests out-of-the-box. The most suitable tool, FindBugs, includes security detectors which fulfill the requirements to detect two out of eight security categories. However, this is not enough for an appropriate security analysis of Java web applications. FindBugs makes it possible to build corresponding security detectors for the other six categories, but it will not be suitable for the average software company or developer.

¹⁴ This was observed with version "1.0.0-rc1" of FindBugs.

References and Resources

- [1] Java Web Application Security, Best Practice Guide / White Paper, 9/2006
http://www.secologic.org/downloads/java/JavaBestPracticeGuideSAP_V3.pdf
- [2] Source Code Analysis Tools – Overview (commercial and free tools; with an focus on C/C++), Department of Homeland Security, 1/2006
<https://buildsecurityin.us-cert.gov/daisy/bsi/articles/tools/code/263.html>
- [3] Free Java Source Code Analyzers
<http://www.java-source.net/open-source/code-analyzers>
- [4] Plug-ins for Eclipse, Java Source Code Analyzers (commercial and free tools)
[http://eclipse-](http://eclipse-plugins.2y.net/eclipse/plugins.jsp?category=Source+Code+Analyzer)
[plugins.2y.net/eclipse/plugins.jsp?category=Source+Code+Analyzer](http://eclipse-plugins.2y.net/eclipse/plugins.jsp?category=Source+Code+Analyzer)
- [5] Fortify Software, <http://www.fortifysoftware.com/>
- [6] Secure Software, <http://www.securesoftware.com/products/>
- [7] Coverity, http://www.coverity.com/products/prevent_security.html
- [8] A Comparison of Bug Finding Tools for Java, Nick Rutar, Christian B. Almazan, Jeffrey S. Foster, IEEE International Symposium on Software Reliability Engineering, 2004
- [9] FindBugs
<http://findbugs.sourceforge.net/>
- [10] BCEL main project page, Apache Jakarta Project
<http://jakarta.apache.org/bcel>
- [11] BCEL introduction, Apache Jakarta Project
<http://jakarta.apache.org/bcel/manual.html>
- [12] FindBugs – Part 2: Writing custom detectors, Chris Grindstaff, IBM, 5/2004
<http://www-128.ibm.com/developerworks/library/j-findbug2/>
- [13] Building Security In – Static Analysis for Security, Brian Chess (Fortify Software) and Gary McGraw (Cigital), IEEE Security & Privacy, 2004
- [14] Finding Bugs is Easy, David Hovemeyer and William Pugh (University of Maryland), OOPSLA'04, 2004
- [15] Finding Application Errors and Security Flaws Using PQL: a Program Query Language, Michael Martin and Benjamin Livshits and Monica S. Lam, OOPSLA'05, ACM, 10/2005
- [16] Julia – A Generic Static Analyser for the Java Bytecode, Fausto Spoto, 2005
- [17] BAT2XML – XML-based Java Byte code Representation, Michael Eichberg (Darmstadt University of Technology), 2005
- [18] Using Programmer-Written Compiler Extensions to Catch Security Holes, Ken Ashcraft and Dawson Engler, 2001
- [19] Java byte code verification - algorithms and formalizations, Xavier Leroy, 2002
- [20] bddb
<http://bddb.sourceforge.net>
- [21] bddshell, Front-end for bddb
<http://bddshell.sourceforge.net>
- [22] PQL
<http://pql.sourceforge.net>
- [23] Eclipse Project, <http://www.eclipse.org>