Practical Measurements Framework for Software Integrity

تطبيق عملي لقياس سلامة البرمجيات

Prepared By

Omar Natheer Basheer

Supervisor
Dr. Hebah H. O. Nasereddin

And Co. Supervisor
Dr. Sharefa F. Murad

Master Thesis

Submitted in Partial Fulfilment of the requirements of the Master Degree In Computer Science

Department of Computer Science

Faculty of Information Technology

Middle East University

Amman – Jordan

January, 2015

Authorization Statement

I'm Omar Natheer Basheer, authorize Middle East University, to distribute and provide hard and electronic copies from this thesis for any concerned educational institution, to be used for scientific studies and researches on demand.

Name : Omar Natheer Basheer

التاريخ : 2015 / 3 /27

التوقع :

إقرار تفويض

انا عمر نذير بشير أفوض جامعة الشرق الأوسط بتزويد نسخ من رسالتي ورقيا والكترونيا للمكتبات ، أو المنظمات ، أو الهيئات والمؤسسات المعنية بالابحاث والدراسات العلمية عند طلبها

الاسم : عمر نذير بشير

التاريخ : 2015 / 3 /27

التوقع :

Examination Committee Decision

This is to certify that the thesis entitled " **Practical Measurements Framework for Software Integrity** " was successfully defended and approved on, 2015

Examination Committee Members		**Signature**
Dr. Hebah H. O. Nasereddin Associate Professor, Department of Computer Information Systems. (Middle East Unevirsity)	Supervisor	.....
Dr. Sharefa F. Murad Assistant Professor, Department of Computer Science. (Middle East Unevirsity)	Co-Supervisor	
Dr. Maamoun K. Ahmed Associate Professor, Department of Computer Science. (Middle East Unevirsity)	(Internal Committee Member)	.....
Abdul Salam W. Arabiyat Associate Professor, Department of Software Engineering. (AL- Balqa' Applied Unevirsity)	(External Committee Member)	.....

Acknowledgement

First of all, I would thank my God, Allah the Almighty, Who gave me the strength and made me reach to this level, Glory of Him.

Then, , I would express my great gratitude to my Co. Supervisor Dr. Sharefa Murad for her support and for her knowledge and for her time, because she was so generous with me. Also, would like to thank my supervisor Dr Hebah H. O. Nasereddin for her support and advise.

Moreover, I would like to thank all of MEU members for teaching me and for providing the time and efforts. Besides, I would love to thank my friends who helped me to get this work done.

Finally, I would like to thank my father and mother for their support through my studying.

Dedication

I dedicate this work to the persons who were with me wherever I go

My lovely parents

Also, I dedicate this work to my lovely

brothers and sisters

May God protect them all.

Table of Contents

Table of Contents

List of Figures

Table of Abbreviations

Abbreviations	Meaning
CBO	Coupling Between Object
CK	Chidamber & Kemerer
CKJM	Chidamber and Kemerer Java Metrics
CIA	Confidentiality, Integrity, Availability
ISO	International Organization for Standardization
LOC	Line OF Code
RFC	Response For Class
NOC	Number of Children

الملخص

تطبيق عملي لقياس سلامة البرمجيات

اعداد : عمر نذير بشير

المشرف : د.هبة ناصرالدين

المشرف المشارك : د. شريفة مراد

الغرض من تكنلوجيا مقياس الجودة هو تقديم اسلوب هندسي منظم من اجل تحديد وتوقع وتقييم جودة البرنامج ، في هذه الدراسة نستخدم مقاييس (Chidamber and Kemerer).

الطريقة التي استخدمها الباحث هي جمع كل المعايير التي تؤثر على السلامة في مرحلة التنفيذ ، بالاضافة الى حساب قيمة السلامة ، وكذلك حساب قيمة المقاييس المستخدمة وهي (Response For Class, Coupling Between Object, and Number Of Children) لمعرفة تاثير هذه المقاييس على السلامة .

في هذا العمل استخدم الباحث ادوات التحليل الثابت من اجل كتشاف كل الاخطاء(المشاكل البرمجية) القابلة للاكتشاف التي توثر على السلامة بالاضافة الى استخدام معادلة الـ (1) لحساب السلامة وكذلك قام الباحث باستخدام اداة (Chidamber and Kemerer Java Metrics) من اجل حساب قيمة المقاييس الثلاثة (Response For Class, Coupling Between Object, and Number Of Children).

ولا بد من الذكر ان الباحث قام باستخدام ثلاث مشاريع مفتوحة المصدر مكتوبة بلغة جافا من اجل معرفة تاثير المقاييس على السلامة في مرحلة التنفيذ واظهرت النتائج الى ان المقاييس كان لها تاثير على السلامة .

فوائد هذا النهج من التطبيق الصارم للمقاييس من الاصدارات المتتابعة من منتجات البرمجيات خلال دورة الحياة البرنامج ، توفر الكشف المبكر للمشاكل التي لها علاقة بالجودة ، التقييم المتكرر لمستويات الجودة توفر رؤية افضل وتمكن من اتخاذ القرارات في الوقت المناسب .

Abstract

Practical Measurements Framework for Software Integrity

Prepared By

Omar Natheer Basheer

Supervisor Dr. Hebah H. O. Nasereddin

Co. Supervisor Dr. Sharefa F. Murad

The purpose of the quality metrics technology is to provide a more disciplined engineering approach to specifying, predicting, and evaluating software quality, this study used Chidamber and Kemerer (CK) metrics.

The suggested method that the researcher uses to combine and select the criteria that affect on integrity in the implantation level. Then, it calculates integrity, and calculates the used metrics which are (Response For Class, Coupling Between Object, and Number Of Children) in order to know the effect of such metrics on integrity.

In this work, the researcher used static analysis tools in order to find all Bugs which it affects on integrity. Additionally, the researcher uses both; the equation (1) to calculate integrity and he also uses Chidamber and Kemerer Java Metrics (CKJM) tool in order to calculate the value of three mentioned metrics (Response For Class, Coupling Between Object, and Number Of Children).

Moreover, the researcher uses three open source Java projects to find the effect of the metrics on integrity at the implantation level, the results of the study indicates that the metrics had an effect on integrity.

The benefits of this approach to the rigorous application of metrics at incremental releases of software products throughout the life cycle provide for early detection of quality-related problems. Periodic assessment of quality levels provides better management visibility and enables timely decision making.

Chapter One

Introduction

1.1 Introduction

We live today in a period of the computer networks and techniques lasting change in our lives. Data security is the foundation of any successful business. Organizations can achieve the goal of data security by possessing the suitable tools to protect data against threats. In distributed information access environment, it is a big problem that how to ensure that access to secure information is safe, (Chandra, S. et al, 2009).

In System Engineering and requirement Engineer, non-functional requirement address the criteria that can be used to evaluate the operation of the system, that's mean describe aspects of the system not related to its execution but rather to its evolution over time to make it more tangible. The main goal of this proposal is to report one of the main important parts of non-functional requirement, which is Security, (De Castro, V. et al, 2014).

The main goal of the security is protecting the worthy and critical information about the organizations and makes it easily obtainable. Assailants use different methods, tools, and techniques to damage the systems and deactivate business operations and profiteering weaknesses of the security controls and policy, and the computer systems[1].

[1]- http://technet.microsoft.com/en-us/library/cc723507.aspx#mainSection :*viewed at: 10 Feb. 2014.*

Achieving secure software has become a challenge for the industry professionals; that's because it required a deep understanding of different or various aspects including security measurements, security policies and security categories.

In software engineering, Security Testing aims at validating software system requirements related to security properties. Although security testing techniques are available for many years, there have been little approaches that allow for specification of test cases at a higher level of abstraction, for enabling guidance of test identification and specification as well as for automated test generation (Schieferdecker, I., et al. 2012).

Software metrics are often used to assess the ability of software to achieve a predefined goal (Jaquith, A. 2007). Software metric is a measure of some property of a piece of software. In addition can use this metric to measure during various software development phases (such as design or coding) and are used to evaluate the quality of software (Chowdhury, 2011).

In the literature, there exist many models addressing security. This work will rely on the main well known model which the Triple Security Model is also named the CIA model. In particular CIA rely on three main security concepts :(Confidentiality, Integrity and Availability), based on this model these three aspects together form guaranteed software. Figure1 demonstrates the CIA three triangle sides: Confidentiality is the term used

to prevent the disclosure of information to unauthorized individuals or systems. (Sattarova, F. et al.2007).

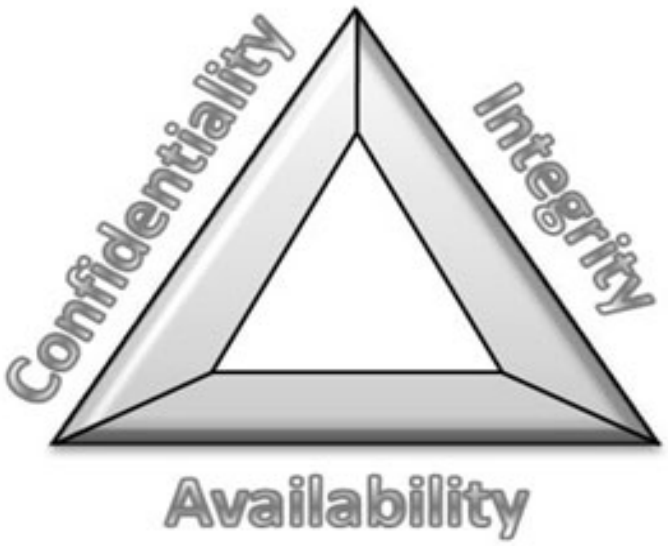


Figure1: The CIA Triad (Solomon, M. G., et al. 2005)

Integrity means that data cannot be modified without authorization (Sattarova, F., et al.2007). It means maintaining and assuring the accuracy and consistency of data over its entire life-cycle. This means that data cannot be altered by unauthorized people. The availability of the information must be available when it is needed. (Sattarova, F., et al.2007).

This thesis focused on one of the three main concepts of CIA Model, which is Integrity.

1.2 Problem Statement

The numerous improvements in the computing environment and the wide presence of computers in homes, educational institutes and organizations; there are different new attacks regularly performed against the confidentiality of data, the Integrity of systems or the copyrights of media providers.(Chandra, S.et al. 2013).

Therefore, Security experts have recognized that in most of the cases, attacks are due to poorly designed and developed software in many cases software is designed and developed without keeping security aspects in mind also users and organizations have a big need today to protect data against different threats, whether it is known or unknown threats, the following problems have been identified:

- There are no available criteria that measure integrity to a given source code.
- There is no software test dependent on a model to measure integrity

1.3 Contributions

The main contributions of this study are illustrated as follows:

- Provide empirical evidence whether there is a correlation CK metrics with Integrity.
- Can we use CK metrics to predict the integrity of software?

1.4 Objectives of the Research

This section describes the desired work in terms of properties; the result of this thesis shall remain the following:

- Define criteria which determine integrity's bugs in the implemented software.
- Calculate the integrity.
- To check if CK metrics are correlated with integrity.

1.5 Hypotheses of the Research

In this section, the researcher investigates how the likelihood of the CK metrics affects on the integrity at code level, to investigate whether CK metrics are affecting on the integrity at the code level or not. The researcher suggests two hypotheses:

Table 1.1: Hypotheses

H_0 : (Null Hypothesis)	CK metrics are not correlated to Integrity.
H_1:(Alternate Hypothesis)	CK metrics are correlated to Integrity.

To validate the hypotheses, the researcher will calculate CK metrics by using CKJM tools (Singer, .et al. 2008), In order to analyze the relationship between the integrity and the metrics.

In this study, it applies some open source programs, and then it calculates the integrity. Furthermore, it calculates the metrics for these programs and analyzes how these metrics are connected with the integrity.

1.6 Questions of the Research

Based on the problems stated in the above section we formulated the following Research Questions:

- In a given source code, how to classify the integrity problems?
- How to calculate the software integrity?
- Does the given Object Oriented Metrics(RFC,CBO,NOC) affect on the integrity of the implemented software as if it is done during the design level?

Chapter Two

Literature Survey

and Related Works

Chapter Two

Nowadays, several software quality models are proposed to evaluate software quality products, and they were developed based on well-known models. Although there are a number of research papers addressing model-based security (David Basin, et al. 2006), (Jan J¨urjens 2005) and model based testing (Paul Baker, et al. 2007). So far, there is little work on Model-based Security Testing (MBST).

This section describes the previous work related to measuring software integrity as well as previous attempt to design a model based test established for security.

Livshits and M. Lam (2005) proposed a static analysis technique to detect many recently discovered application vulnerabilities such as SQL injections, cross-site scripting, and HTTP splitting attacks. They designed a user-provided specifications system of vulnerabilities which are automatically translated into static analyzers and they found in their static analysis that two of 29 security vulnerabilities are residing in widely-used Java libraries. Then, they showed in their approach that all vulnerabilities match the specification within the statically analyzed code. They formulated many vulnerabilities types including SQL injections, cross-site scripting, HTTP splitting attacks as tainted object propagation problems. The results of the study indicated that their analysis is an effective practical tool for finding security vulnerabilities. The analysis stated false

positives for only one application and they determined that the false warnings reported could be eliminated with improved object naming.

The authors used a static analysis technique through compiling 29 security vulnerabilities in 9 large programs, which have been written in C and C++ languages. In this work, the researcher collects 195 Bugs, which affect on the integrity, via using 3 tools static analysis to find these bugs, in addition to using 3 projects that's written in Java language.

According to the study of (Yue Jiang, et al 2008) the objective of their study is to compare the performance of predictive model which use the design-level metrics with code-level metrics. The study analyzes 13 datasets of the "NASA Metrics Data Program" which provide code metrics and design. Through applying a domain of statistical important tests and modeling techniques, it assured that the models which have been built using code metrics usually perform better than the models which based on design metrics. The study explains the result from foretelling the errors from; design metrics, "static code metrics", and integration of both of them. And it concludes that the models which consist of an integration of the code and design level metrics performs better than models that used only design level or code level.

The authors present in their paper a comparison between the performance of predictive models which use design-level metrics with the

models which use code-level metrics and the models that use both of them. In addition, the authors used the design metrics that include node_count, dge_count, and Mc-Cabe Cyclomatic complexity measure, the static code metrics, such as num_operators, num_operands, and Halstead metrics are calculated from program statements.

According to the authors (Al-Badareen, A. et al. 2011), a complet comparison has been done between popular models of software quality like “ISO IEC 9126 Model (Dromey, R. G. 1995)”, “McCall Model (James, M.1977)”, “FURPS Model”, “Boehm Model (Boehm, B. et al. 1978)”, and “Dromey Model (Dromey, R. G. 1995)”. The result displays the weaknesses and strength of those standards in measuring security and other quality sides.

Chowdhury (2011) provided empirical evidence that complex, coupled, and non-cohesive software entities are generally less secure. He explored that Complexity, Coupling, and lack of Cohesion (CCC) metrics positively correlated to the number of vulnerabilities at a statistically significant level over five major releases of Mozilla Firefox. The correlation is on average 0.5 with a p-value less than 0.001. The code-level CCC metrics are generally more strong correlated to vulnerabilities than the design-level CCC metrics. However, design-level metrics such as NOC can be good

indicators of vulnerabilities. The authors explain that the metrics of CCC are systematically related to weaknesses via five versions of “Mozila Firefox”. The steady relation patterns mean that, once standardization for a certain project, it can use these metrics to show the weaknesses of the new versions reliably. The research compares the prediction effectiveness of the “Random Forests”, “Naive-Bayers”, and “C4.5 Decision Tree” techniques in the process of predicting the weaknesses of the entities that based on “CCC metrics”. The research concluded with these results; the “CCC metrics” are practical and beneficial addition to the automatic weaknesses prediction framework, which give the software practitioners the ability of taking precautionary actions against primary weaknesses through the software lifecycle early.

The authors (Khan, S. et al. 2012) give an Idea about “Integrity Quantification Model” (IQM) for Object Oriented Design. They estimate integrity, security attributes in term of complexity factors: Coupling Function (CP), which is points to corresponding Coupling between Object (CBO) metrics-, Total Supporting Services (TSS), which points to Response for Class (RFC) metrics- and Higher Level of Abstraction (HLA), which points to Number of Children (NOC) metrics. Figure2 discuss the relation of integrity, security attributes with complexity factors and corresponding metrics for design complexity.

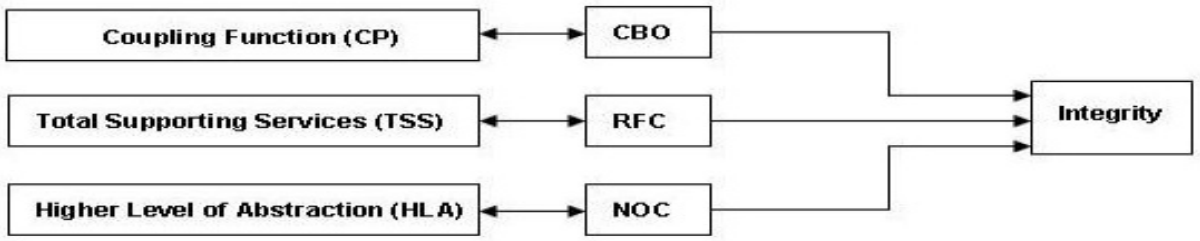


Figure (2.1) Relation Diagram, (Khan, S. et al. 2012)

A multiple linear regression technique has been used to evaluate the coefficients. This technique establishes a relationship between a dependent variable and multiple independent variables.

The authors offered a definition of measuring Integrity. Then proposed algorithms were calculated manually. In addition the authors addressed the design phase of software by measuring the quality metrics of a UML diagram. In this work we will generate an mechanized software testing, measuring software quality derived from a source code.

The authors (Schieferdecker, I.,et al. 2012), discuss an idea about security testing called “Model-based security testing”. It is a new field devoted to the effectiveness and the systematic specification of the objectives of the security test, cases of security test, and to their semi-automated or automated generation. The MBST technique includes “model-based fuzzing”, “security functional tests”, and impendence oriented testing and using of the patterns of the security test.

This work provided a survey on MBST techniques and the related models as well as samples of new methods and tools that are under development in the European ITEA2-Project DIAMONDS.

The authors (Chandra, S.et al. 2013), illustrated a methodology to scrutinize class hierarchy against security. Two security metrics and integrity state transition model have been developed for quantitative assessment of Integrity Risk. The methodology checks the integrity of the class hierarchy which implemented on online music store case study with experimental validation. The work measures and ranks security of software at the design stage of software development and concentrate on only integrity, security attribute which is one of the essential security requirements. The main aim of the methodology is quantification of integrity risk in the design stage. It explored set of sensitive classes and risky classes may use for further analysis or for improvements during the design stage.

Chapter Three

Background Information

Chapter Three

This chapter, focus a light on the tools that have been used to detect bugs which affect on the integrity, such as; FindBugs, Jtest, and VCG. Besides, the chapter also describes the tools that calculate the applying metrics in the study which is called CKJM. In addition, there is an overview about the integrity, metrics tools, static analysis, integrity, software quality factors, factor definitions.

3.1 Introduction

In the last decade, researchers have often tried to improve the usability, portability, integrity and other aspects of software in order for it to be more user-friendly and gain user trust. Several methods, techniques and tools have been proposed to reduce the negative effects of software size and complexity as well as detect vulnerabilities. Moreover, several software quality models were proposed to evaluate general and specific type of software products. In order to customize the closed model to the intended scope, some of the studies and researchers went to comparisons between the well-known models (ISO IEC9126, McCall and Boehm). These comparisons are leak of criteria that is conducted based on different perspectives and understanding. (Al-Badareen, A. B.,et al, 2011). On the other hand some, researchers also headed toward giving an idea about Integrity

Quantification Model (IQM) for Object Oriented Design, which has been developed using OOD construction at design time. For that it estimates integrity security attribute in terms of complexity factors such as CBO, RFC metrics and NOC metrics, these factors used for design complexity. (Khan, S. A., et al, 2012).

3.2 Metrics Tools

The metrics for the object-oriented software system focused on the structure of the source code, like encapsulation, inheritance, polymorphism, and object abstraction (Shaik, A., et,al. 2010).

Object Oriented (OO) metrics are widely used in the life cycle of a software product for assessing various attributes such as quality of the design, complexity of the code, find faults by detecting vulnerabilities, bugs and estimate the effort involved for maintenance which impact on the integrity as well as the bugs description exist in the **Appendix A**. As for the popularity of OO metrics have been increased, the large number of metrics proposed for detecting the various aspects of OO programming. Metrics can be collected manually or by an automated tool.

Nowadays, there are existing tool suffers mainly from various problems such as, (Shaik, A., et al. 2010):

1. Most of them are commercial tools where extensibility becomes a bad issue.
2. The researcher will not have access to the source code for adding his own metrics to the list of existing metrics.
3. Free tools are available, but they are specific to a language “like JAVA” or not easy to adapt to other metrics.
4. All tools are the interpretation of the metrics; most metrics definitions are more ambiguous and hence more than one alternative of the same metric which have been proposed by different researchers.

3.2.1 Chidamber and Kemerer Java Metrics (CKJM)

The automated tool that had proposed in this study used the Java programs, the purpose of using CKJM tool is to calculate for each class the following eight metrics proposed by Chidamber and Kemerer:

- WMC: Weighted Methods per Class
- DIT: Depth of Inheritance Tree
- NOC: Number of Children
- CBO: Coupling Between Object classes
- RFC: Response for a Class
- LCOM: Lack of Cohesion in Methods

Moreover, it also calculates for each class Ca: Afferent couplings and NPM: Number of public methods[2], in addition to use 4 metrics from CK metrics in this thesis. For example, the results of CKJM tools are represented as shown in figure3.1

```
<class>
    <name>org.argouml.uml.ui.behavior.common_behavior.PropPanelCallAction$UMLCallActionOperationComboBoxModel</name>
    <noc>0</noc>
    <cbo>9</cbo>
    <rfc>34</rfc>
    <loc>137</loc>
</class>
```

Figure 3.4: CKJM Results

In this table 3.1 description (NOC, CBO, and CBO) metrics that have been used in the thesis.

Table 3.1 CK metrics

Metrics Name	Define
Number of Children (NOC)	Represents the number of immediate subclasses (Children) subordinated to the class (parent) in the class hierarchy. NOC measures how many methods or field inherited directly by classes from a super class. .Classes with a large number of children have to provide more generic service to all the children in various contexts and should be more flexible, a

[2] http://www.spinellis.gr/sw/ckjm/. *viewed at: 10 dec. 2014.*

	constraint that can introduce more complexity into the parent class.[3]
Coupling between Objects (CBO)	**CBO** Is the number of other classes that a class is coupled to. It is only applicable to object-oriented systems. For example methods of one use methods or instance variables of another. Since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. Excessive coupling indicates weakness of class encapsulation and may inhibit reuse. So parts that have a high (outgoing) efferent coupling may be inversely related to security, since they can be affected by security problems in other parts of the system.[4]
Response for a Class (RFC)	Can be defined as set of methods that can be potentially executed in response to a message received by an object of that class. So RFC is only applicable to object-oriented systems. If a large number of methods can be invoked in response to a message, then the testing and debugging of the

3 http://www.arisa.se/compendium/node102.html#metrics:NOC *viewed at: 10 dec. 2014*

4 http://www.arisa.se/compendium/node105.html#metric:CBO viewed at: 10 dec. 2014

	class becomes more complicated since it requires a greater level of understanding required on the part of the tester.[5]
Lines of code (LOC)	Lines of code simply counts the lines of source code (line break characters) of a certain software entity, it is simple yet powerful to assess the complexity of software methods and entities, it is critical to use it in generated codes since it may lack of line breaks because it is depending on code conventions and format. Additionally it can only be measured in the source code itself from the front-end and is therefore a front-end side metric, currently only the Eclipse Java Front-end supports this as well as the UML Front-end can of course not calculate this metric, since there is no source code in UML. LOC to security requires to be able to locate the parts of a system responsible for security. The size of these parts might refer to higher security. Security might be increased with increasing LOC.[6]

5 http://www.arisa.se/compendium/node98.html#metric : RFC viewed at: 10 dec. 2014

6 http://www.arisa.se/compendium/node91.html#metric:LOC viewed at: 10 dec. 2014

3.3 Static Analysis Tools

The process of detecting the errors in the source code without any process of execution for it, is known as the “static analysis”, (Al Mamun, M. et al. 2010). Today, the analysis tools that used to detect the errors in the software have become a common topic according to the researchers. However, the available information about the value and accuracy of the experimental estimation for these tools are little. The present analysis conducted via many open source and commercial tools. The price of the commercial tools is extremely expensive compared to the research and open source tools, and it requires a license, which prevents the researcher of any evaluative and experimental data (Ayewah, N., et al. 2007). In order, to achieve the goal of detecting different bugs which impact on the integrity, the researcher in this study explored three techniques of the static analysis; VCG, FindGugs, and Jtest.

There exist different cases of error detection using static analysis. One of them is within the execution code review for a new module. In these cases, the developers will be concerned with reviewing the warnings in the code, and will want to correct the misleading and confusing code. Another case is through looking for errors in a big code. In this case, in the changing the code process the threshold is high responding to any warning, (Ayewah, N., et al.2007).

3.3.1 Find Bugs (V3.0) Tool

FindBugs is an open source static analysis tool that analyzes finds potential problems in Java class files to detect occurrences of bug patterns, (Ayewah, N., et al. 2007).

The analysis tool reports nearly 300 different bug patterns (Ayewah, N., et al. 2007). These patterns are defined in plug-in architecture in which users can select the patterns they want to analyze the source; the patterns are categorized as follows:

Table 3.2: patterns

Malicious code vulnerability	Code that can be maliciously altered by other code.
Dodgy	Code that can lead to errors.
Bad practice	Code that violates the recommended coding practices.
Correctness	Code that might give different results than the developer intended.
Internationalization	Code that can inhibit the use of international characters.
Performance	Code that could be written differently to improve performance.
Security	Code that can cause possible security problems.
Multithreaded correctness	Code that could cause problems in multi-threaded environment.
Experimental	Code that could miss cleanup of steams, database objects, or other objects that require cleanup operation.

Figure3.2 shows the result of using Findbugs tool; the description can be found in Appendix A. As shown in figure 3.2 the bug type is “E1_EXPOSE_REP” so via “Appendix A” it’s clear that the bug catagory is “MALUIOUS_CODE”.

Figure 3.5: FindBugs Tool result

3.3.2 Visual Code Grepper Tool

VCG is a tool that used for reviewing the automated code security for PHP, PL/SQL, C++, VB, C#, and Java, which are proposed to significantly speed up the process of reviewing the code via determination insecure and bad code. Figure 3.3 displays the result of using VCG, the description of them can be found in Appendix A. It's clear from the figure that the bug type is “SQL Injection”.

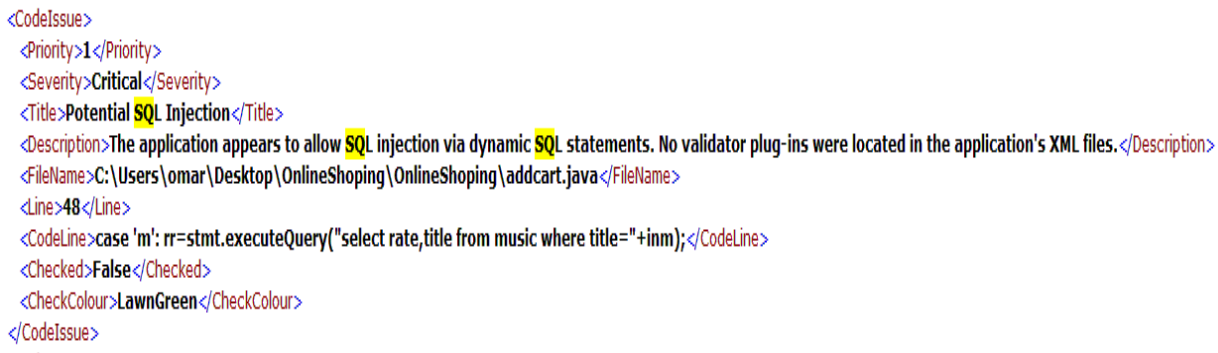


Figure 3.3:VCG Tool result

Shown in Figure (3.4) example, explain bugs, The following Java servlet code, used to perform a login function, illustrates the vulnerability by accepting user input without performing adequate input validation or escaping meta-characters:

```
conn = pool.getConnection( );
String sql = "select * from user where username='" + username +"' and password='" +
password + "'";
stmt = conn.createStatement();
rs = stmt.executeQuery(sql);
if (rs.next()) {
loggedIn = true;
        out.println("Successfully logged in");
} else {
        out.println("Username and/or password not recognized");
}
```

Figure 3.4:Example SQL Injection Bugs [7]

3.3.3 Parasoft Jtest (V9.5) Tools

Parasoft Jtest is a comprehensive Java testing product for development teams building Java EE, SOA, Web, and other Java applications. For Java development teams building SOA, Web services, or Web applications, Jtest works with Parasoft SOA test and Parasoft Web King to provide a comprehensive, integrated testing solution. The purpose of jtest is to help in increasing the using Java software's reliability while dramatically reducing

[7] https://www.owasp.org/index.php/Preventing_SQL_Injection_in_Java#Example_of_SQL_injection : viewed at: 25 feb. 2015

the amount of time that spend in testing. (Momotaz, S. 2010), Jtest automatically performs the following:

- White-box testing, Black-box testing, and Regression testing, and Static analysis (coding standard enforcement) of Java code.

Figure3.4 shows the result of using Jtest tool with the description as the description can be found in Appendix A, as shown in figure 3.4 the bug type is “Avoid string literals except in constant declarations and calls to System.out or System.err's 'print' or 'println' methods [SECURITY.WSC.SL-4]” so by using the “Appendix A in page (138)” this bug find in line “446”.

```
<weakness id="2755" tool_specific_id="2755">
    <name>SECURITY.WSC.SL</name>
    <location line="446" path="C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-app\src\org\argouml\uml\cognitive\checklist\Init.java" />
    <grade probability="1.0" severity="4" tool_specific_rank="4" />
    <output>
        <textoutput>The String literal "checklist.state.structure.merged-with-other" is used</textoutput>
    </output>
    <!--
                    <evaluation correctness="" falsepositive="" severity="">
                        <comments />
                    </evaluation>
                -->
</weakness>
```

Figure 3.5:Jtest Tool Result

shown in Figure (3.6) example explain bugs:

```
public class SL {
    public static void main(String args[]) {
        String msg= "Welcome to Hello World!"; //VIOLATION
        System.out.println(msg);
    }}
```

Figure 3.6: :Example SECURITY.WSC.SL-4 Bugs

3.3.4 Integrity

There are three main properties that concern the systems of information security; availability, integrity, and confidentiality. These characteristics represent the main concerns in the military and commercial industry. Historically, the privacy has obtained more attention because of its importance for the army. The main goal of the army environment is to hinder exposure of information. On the other hand, the main objective of the systems of commercial security is guaranteed that the "integrity" of its data is conserved from incorrect changes and unsuitable actions which can be done by forbidden users. The importance of the "confidentiality" is equally in the environment of commercial. But, David D. Clark and David R. Wilson explored that the confidentiality of information is less important than the integrity in most of the commercial systems, (Blake, S. Q. 2000).

According to Kreitzberg (1982), he identifies integrity is the ability's measure of a program to perform correctly on different sets of input. Other hand, the integrity is a measure of how a well program has been tested.

Integrity is the extent to which the system will perform without failure due to unauthorized access to the system or system information. Thus, high system integrity implies high software integrity. Furthermore, and in most applications, system integrity based on the software and continued software functioning. Whereas these applications the software survivability would also affect the system integrity.

The authors explored that the integrity gives an ability of concluding how impervious the software, which is based on find out the weakness, where a simple error in the program can make the application susceptible to forbidden data access, data deletion, or data update, and the crashes for the application makes it forbidden for “denial service attacks”, (Basili, V. R. 1993).

3.4 Software Quality Factors

There are certain factors of the quality which are grouped under three procuration concerns: adaptation, performance, and design. Thus, procuration manager specifies the requirements of the software. As for “Department of Defense Software Development Standard” (DOD-STD-SDS) the format areas in includes the software performance feature, the design of software and production, enhances the adaption of software reusability, as well as the software quality assurance including metrics.

Additionally, VanSuetendael, N.el at. (1991) state that the similarity of concerns of both areas and acquisition enables the acquisition manager in order to identify and select easily the quality factor categories and specific factors of interest.

3.6 Factor Definitions and Rating Formulas

Quality factor definitions and factor rating formula for integrity is previewed as below:

$$\text{Integrity} = 1 - \frac{\text{Errors}}{\textbf{\textit{Lines}} \text{ of Code}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 1$$

This formula quantifies user concerns for the final product. Besides, there are three types of measurements that the formula is used:

1- Number of errors per lines of code.

2- Effort to perform an action.

3- Utilization of resources.

The values of the rating must fall with values from 0 to 1. The integrity characteristic concerns with the failures of the software security which are because the forbidden access. The formula of the integrity is produced from the number of the errors of the integrity-related software that happen within an assumed time (e.g., 51+ during operational testing and evaluation) and the summation of the source code' lines, (VanSuetendael, N. el at, 1991).

This formula is comparable to the formula for reliability; the variation between them is that reliability is concerned with all errors of the software, and integrity is concerned only with the subset of errors that impact on the integrity. For example, if three integrity-related errors per

10,000 lines of code happen during operational testing and evaluation, the rating formula shows an integrity level of 0.9997 [1 –(3/10,000 0.9997)], (VanSuetendael, N. el at, 1991).

Integrity associates with metrics that measures attributes of software that bear on the ability in order to prevent unauthorized access, whether accidental or deliberate to the programs or data.

This thesis will take consideration of the NOC, CBO and RFC to take a look how these metrics impact on the software Integrity, Depending on related work.(Khan, S. A., et al, 2012).

Chapter Four

Proposed Model

Chapter Four

Proposed Model

In this research, the researcher uses two methods; the first method is based on defining and combining all the criteria which determine integrity bugs in software implementation. While the second one includes, determine the integrity bugs by using static analysis tools, calculate metrics by using CKJM tool. The final phase is to calculate the value of integrity by using equation number 1. After that, we analyze the final results in order to know whether the metrics have a relative relationship with integrity.

4.1 The Proposed Model (1)

This Proposd is based on combining the criteria that determine integrity bugs in software implementation. Thus, this contains 5 phases:

A) Collect all sources of Bugs & warnings in a source code.

B) After collecting Bugs & warnings, we define and classify them into different categories.

C) Select Bugs & warnings that have relationship with the integrity, we almost collected 205 Bugs. The descriptions of these Bugs are indicated in appendix (A).

D) The researcher searched for the available tools that explore these Bugs. There are three tools being used in this study, two of them are free (VCG, FindBugs), while the third one is commercial (Jtest).

E) Apply these Static Analysis tools (VCG, FindBugs, Jtest) on selected software projects.

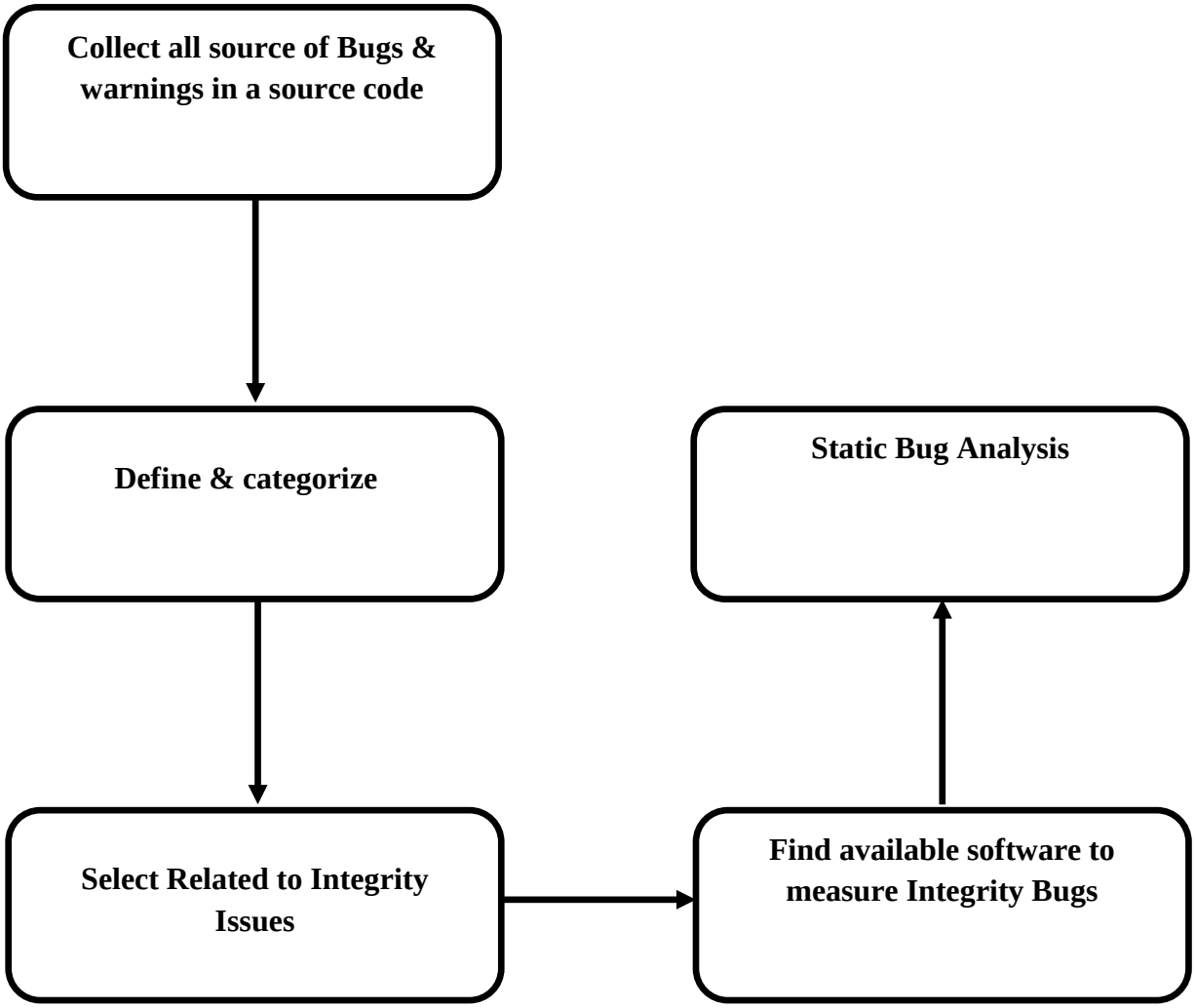
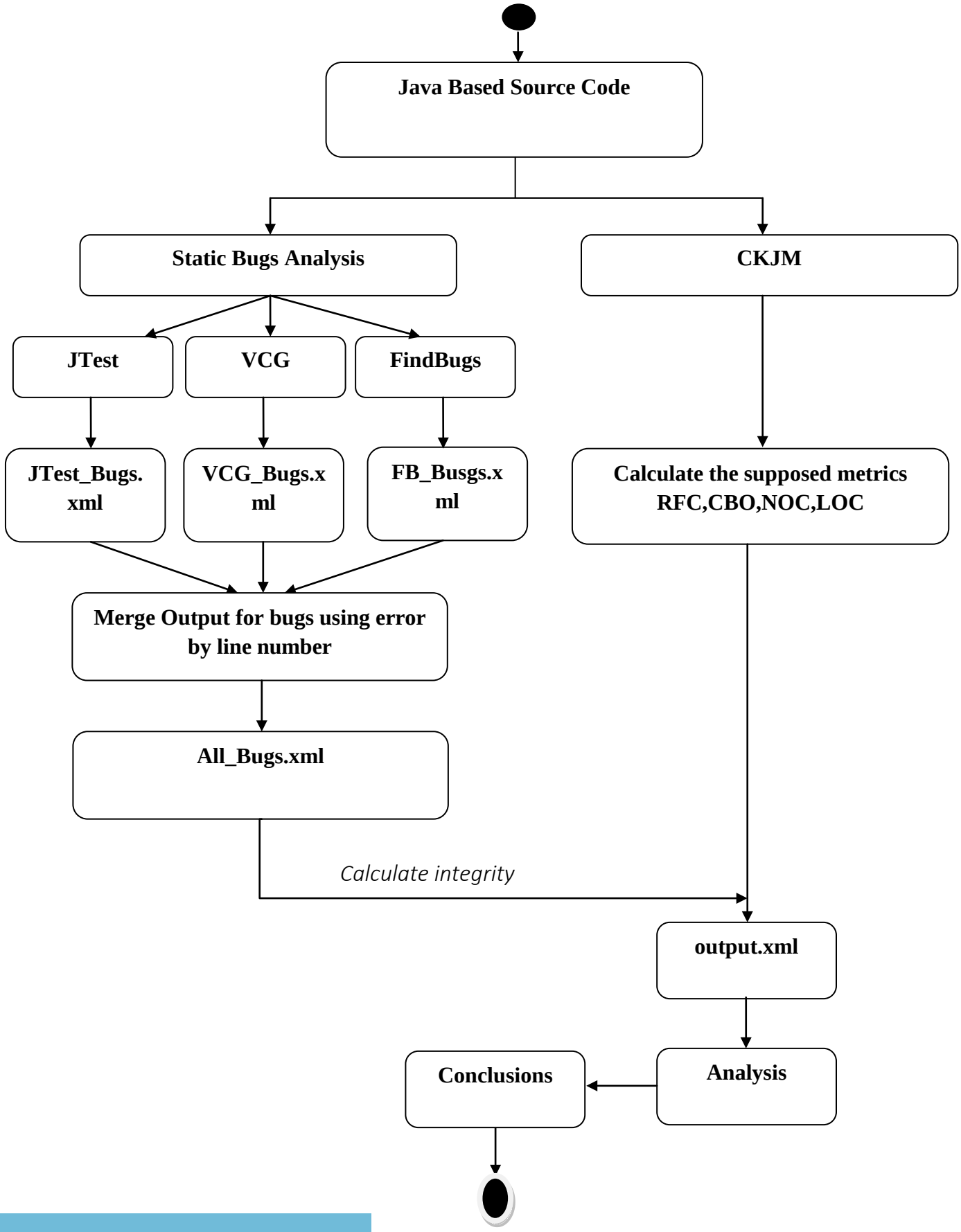


Figure 4.1 : proposed model



`**Figure 4.2 : proposed model 2**

4.2 The Proposed Model (2)

After download the projects, these programs are inserted into static analysis tools. These tools are used to discover the Bugs that affect on the integrity at code level, a brief explanation of these software projects will be cited in chapter Five

First step

We parsed the software project's source files through Jtest tool (which is a commercial tool from parasoft that is used to find errors in source code files, we used it in order to find the bugs that affects the integrity). After analyzing the project's source code through Jtest tool we got the following output shown in Figure 4.3.

```
<weakness id="1" tool_specific_id="1">
    <name>SECURITY.WSC.INIVF</name>
    <location line="59" path="c:\\argouml\application\ArgoVersion.java"/>
    <grade probability="1.0" severity="5" tool_specific_rank="5"/>
    <output>
      <textoutput>Flag not present</textoutput>
    </output>
```

Figure 4.3: Jtest Results

And in the same step we parsed the software project's source files through VCG tools which it is a static analysis tool, in order to defect fault sin source code files. After the analyzing process, we got the following output, shown in Figure 4.4

```
<CodeIssue>
    <Priority>2</Priority>
    <Severity>High</Severity>
    <Title>java.lang.Runtime.exec Gets Path from Variable</Title>
    <Description>The pathname used in the call appears to be loaded from a variable. Check the code manually to ensure that malicious filenames cannot be submitted by an attacker.</Description>
    <FileName>C:\ArgoUML-0.30.2-src\argouml\src\argouml-app\src\org\argouml\util\osdep\StartBrowser.java</FileName>
    <Line>63</Line>
    <CodeLine>Runtime.getRuntime().exec(</CodeLine>
    <Checked>False</Checked>
    <CheckColour>LawnGreen</CheckColour>
```

Figure 4.4 : VCG Results

Synchronously, we parsed the software project's source files via FindBugs, which it is a static analysis tool. After analyzing the project's source code via the FindBugs tool , we got the following output, shown in Figure 4.5

```
<BugInstance type="EI_EXPOSE_REP2"
priority="2"
 rank="18"
abbrev="EI2"
 category="MALICIOUS_CODE">
    <Class classname="antlr.ANTLRHashString">
      <SourceLine classname="antlr.ANTLRHashString" start="23" end="106" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Class>
    <Method classname="antlr.ANTLRHashString" name="setBuffer" signature="([CI)V" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" start="98" end="101" startBytecode="0" endBytecode="51" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Method>
    <Field classname="antlr.ANTLRHashString" name="buf" signature="[C" isStatic="false">
      <SourceLine classname="antlr.ANTLRHashString" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
    </Field>
    <LocalVariable name="?" register="1" pc="2" role="LOCAL_VARIABLE_UNKNOWN"/>
    <SourceLine classname="antlr.ANTLRHashString" start="98" end="98" startBytecode="2" endBytecode="2" sourcefile="ANTLRHashString.java" sourcepath="antlr/ANTLRHashString.java"/>
  </BugInstance>
```

Figure 4.5: Output results tool FindBugs

Synchronously, we parsed the software project's source files through CKJM tool, this tool is used to calculate the supposed metrics in our methodology (NOC, RFC, CBO, and LOC). The output will be as follows: figure 4.6

```
<class>
        <name>org.argouml.ui.PredicateMType</name>
        <noc>0</noc>
        <cbo>2</cbo>
        <rfc>11</rfc>
        <loc>92</loc>
</class>
```

figure 4.6: Output results tool CKJM

Second Step

Take the resulted outputs from the mentioned tools (which are in xml format from Figure 4.3, Figure 4.4, and Figure 4.5), then we merge these outputs into a single output file, throughout this merging process any duplicated report of an error/warning for the same bug on the same source code file (same class/same line) will be deleted, leaving one instance within the merged output file, the resulted output file is shown in the next figure: figure 4.7 ,4.8 and 4.9 display the XML output after parsing.

```
        <severity>5</severity>
        <type>SECURITY.WSC.INIVF</type>
        <description>Flag not present</description>
        <classname>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-
src\argouml\src\argouml-app\src\org\argouml\application\ArgoVersion</classname>
        <sourcepath>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-
src\argouml\src\argouml-app\src\org\argouml\application\ArgoVersion.java</sourcepath>
```

```
<line>59</line>
<tool>JTest</tool>
```

figure 2.7: Output Jtest after unified Output

```
<bug>
<severity>High</severity>
<type>java.lang.Runtime.exec Gets Path from Variable</type>
<description>The pathname used in the call appears to be loaded from a variable. Check the code manually to ensure that malicious filenames cannot be submitted by an attacker.</description>
<classname>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-app\src\org\argouml\util\osdep\StartBrowser</classname>
<sourcepath>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-app\src\org\argouml\util\osdep\StartBrowser.java</sourcepath>
<line>63</line>
<tool>VCG</tool>
```

figure 4.8: Output VCG after unified Output

```
<severity>1</severity>
<type>MS_SHOULD_BE_FINAL</type>
<description>MALICIOUS_CODE</description>
<line>78</line>
<classname>tudresden.ocl.lib.Ocl</classname>
<sourcefile>tudresden/ocl/lib/Ocl.java</sourcefile>
<tool>FindBugs</tool>
```

figure 4.9: Output FindBugs after unified Output

Third step

Figure 4.10 shows the output of merging of the resulted outputs from Jtest, FindBugs and VCG and CKJM The outputs will be as follows:

```
<class>
<name>org.argouml.uml.ui.behavior.common_behavior.PropPanelLink</name>
        <noc>0</noc>
        <cbo>12</cbo>
        <rfc>20</rfc>
        <loc>93</loc>
</class>
<bug>
<severity>4</severity>
<type>SECURITY.WSC.SL</type>
<description>The String literal "button.new-pseudostate" is used</description>
<classname>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-
app\src\org\argouml\uml\ui\behavior\state_machines\UMLCompositeStateSubvertexList</classname>
<sourcepath>C:\Users\omar\Desktop\TestProject\ArgoUML-0.30.2-src\argouml\src\argouml-
app\src\org\argouml\uml\ui\behavior\state_machines\UMLCompositeStateSubvertexList.java</sourcepath>
<line>65</line>
<tool>JTest</tool>
</bug>
```

Figure4.10: Output CKJM with Error number

After that, we take (classname , Errornum, NOC, RFC, CBO, LOC) then the outputs will be as shown in figure 4.11.

```
<class>
        <name>com.salesmanager.web.files.FilesController</name>
        <noc>0</noc>
          <cbo>7</cbo>
          <rfc>16</rfc>
          <loc>113</loc>
       <errorNum>10</errorNum>
</class>
```

Figure 4.11: Output CKJM with Integrity

Fourth step

We will use Figure 4.11 to calculate the effect of (NOC, RFC, CBO) metrics on integrity throughout employ A Multiple Linear Regression (MLR) technique which is used to get the coefficients. This technique establishes a relationship between a dependent variable and multiple independent variables. The Multiple Regression equation is in the following

The Multiple Linear Regression (MLR) technique was selected and used for the software development and software testing processes., (Fedotova, O., el at. 2013)

Multiple Linear regression tries to form a relationship between two or more interpretations variables and the response variable via appropriate linear equation to look at the data. Each one of the values of the independent variable x is connected with a value of the dependent variable y. The model is defined as in Eq.(2):

$$Y = \alpha_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots\cdots\cdots \beta_n X_n \ldots\ldots\ldots\ldots\ldots 2$$

Where:

Y: is the value of the Dependent variable (Y), what is being predicted or explained

α_0: (Alpha) is the Constant or intercept.

β_1: Is the Slope (Beta coefficient) for X.

X: First independent variable that is explaining the variance in Y.[8]

8 http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm. *viewed at: 10 DEC. 2014.*

Chapter Five

Evaluation and Results

Chapter Five

Evaluation and Results

In this chapter, we will present subject evaluation and the results that are analysis by using the Multiple Linear Regression (MLR). Section 5.1 subject evaluation 5.2 contains of Interpretation of regression results to projects. Section 5.3 contains of ArgoUML results. Section 5.4 contains of the Shopizer-Ecommerce project results. Section 5.5 contains of Payment4j Project results, section 5.6 limitation.

5.1 Subject Evaluation

The researcher uses three projects in the evaluation process for the results, which are ArgUML, Payment, and Ecommerical, where the Payment and Ecommerical are project interested in the security, and the ArgUML not interested in the the security and these projects are open source Java programs which were downloaded via (Githut, Sourceforge) while these two sites are considered as internet pages included sharing the open source projects with all people, and with the capability of downloaded the needed projects. Thus, three projects were downloaded which are explained in Table (5.1) below:

Table (5.1) Open source testing project

Name	Version	Location	# of Class
ArgoUML	0.30.2	http://argouml-downloads.tigris.org/source/browse/argouml-downloads/	1914
payments4j	0.0	https://github.com/CarlosZ/payments4j	194
shopizer-ecommerce	2.0.0	https://github.com/shopizer-ecommerce/shopizer	861

These projects were applied following the proposed model 2 which previously had been explained in Chapter 4.

5.2 Interpretation of Regression Results

In this section we will present Interpretation of results project that used in this thesis, (ArgoUML , Shopizer-Ecommerce , Payment4j)

5.3 ArgoUML Project

Table 5.2 : Variables Entered/Removed to ArgoUML

Variables Entered/Removed[b]			
Model	Variables Entered	Variables Removed	Method
1	RFC, CBO, NOC		Enter
a. All requested variables entered.			
b. Dependent Variable: integrity.			

Variables Entered :- SPSS allows you to enter variables in a regression in blocks, Hence, you need to know which variables were entered into the current regression.

Variables Removed:- This column listed the variables that were removed from the current regression. Usually, this column will be empty unless you did a stepwise regression.

Method:- This column tells you the method that SPSS used to run the regression. "Enter" means that each independent variable was entered in the usual fashion. If you did a stepwise regression, the entry in this column would tell you that.

Table 5.3 : Summary of the Model To ArgoUml project

Model Summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.134a	.018	.016	.09415
a. Predictors: (Constant), RFC, CBO, NOC				

R:- ((Pearson Correlation Coefficient)) is the correlation between the observed and predicted values of the dependent variable.

R-Square :- is the square root of R-Squared, this is the proportion of variance in the dependent variable (science) which can be explained by the independent variables. This is an overall measure of the strength of association and does not reflect the extent to which any particular independent variable is associated with the dependent variable.

Adjusted R-square :- This is an adjustment of the R-squared.

Std. Error of the Estimate :- This is also referred to as the root mean squared error. It is the standard deviation of the error term and the square root of the Mean Square for the Residuals in the ANOVA table.

Table 5.3 shows that Pearson Correlation is 0.13, which means that there is a relationship between RFC, NOC, CBO on integrity, however, R^2 is 0.018 which indicates that 1.8% of the variation in dependent variable (integrity) can be explained by the independent variables (RFC, NOC and CBO).

Table 5.4 : Results of ANOVA Test To ArgoUml project

ANOVA[b]						
Model		Sum of Squares	Df	Mean Square	F	Sig.
1	Regression	.311	3	.104	11.677	.000a
	Residual	16.939	1911	.009		
	Total	17.249	1914			

a. Predictors: (Constant), RFC, CBO, NOC

b. Dependent Variable: integrity.

Model :- SPSS allows you to specify multiple models in a single regression command. This tells you the number of the model being reported.

- **Regression, Residual, Total** - Looking at the breakdown of variance in the outcome variable, these are the categories we will examine: Regression, Residual, and Total. The Total variance is partitioned into the variance which can be explained by the independent

variables (Model) and the variance which is not explained by the independent variables (Error).

Sum of Squares :- These are the Sum of Squares associated with the three sources of variance, Total, Model and Residual. The Total variance is partitioned into the variance which can be explained by the independent variables (Regression) and the variance which is not explained by the independent variables (Residual).

df :- These are the degrees of freedom associated with the sources of variance.

Mean Square :- These are the Mean Squares, the Sum of Squares divided by their respective DF.

F and Sig.:- This is the F-statistic the p-value associated with it. The F-statistic is the Mean Square (Regression) divided by the Mean Square (Residual). The p-value is compared to some alpha level in testing the null hypothesis that all of the model coefficients are 0.

Table 5.4 shows that Sig= 0.000 less than 5%, where the statistical relationship used is 5%, and it means that there is a statistically significant difference between RFC, NOC, CBO and integrity, further statistical analysis will indicate which of these independent variables has a significant impact.

Table 5.5 : Individual Regression Coefficients To ArgoUml project

Coefficients[a]						
Model		Unstandardized Coefficients		Standardized Coefficients	T	Sig.
		B	Std. Error	Beta		
1	(Constant)	.943	.003		338.469	.000
	NOC	8.596E-5	.000	.004	.185	.853
	CBO	-3.848E-6	.000	-.001	-.029	.977
	RFC	.002	.000	.134	5.328	.000
a. Dependent Variable: integrity						

B - These are the values for the regression equation for predicting the dependent variable from the independent variable. The regression equation is presented in many different ways, for example:

$$\mathbf{Y_{predicted} = b0 + b1*x1 + b2*x2 + b3*x3 \quad \ldots (2)}$$

Std. Error - These are the standaJrd errors associated with the coefficients.

Beta - These are the standardized coefficients. These are the coefficients that you would obtain if you standardized all of the variables in the regression, including the dependent and all of the independent variables, and ran the regression. By standardizing the variables before running the regression, you have put all of the variables on the same scale, and you can compare the magnitude of the coefficients to see which one has more of an effect. You

will also notice that the larger betas are associated with the larger t-values and lower p-values.

t and Sig. - These are the t-statistics and their associated 2-tailed p-values used in testing whether a given coefficient is significantly different from zero. Using an alpha of 0.05:

The coefficient for math (0.389) is significantly different from 0 because its p-value is 0.000, which is smaller than 0.05.

According to the above mentioned table , we can suggest the following equation :

Integrity = 0.943 + 8.596E-5 NOC -3.848E-6 CBO+ 0.002 RFC

According to the above mentioned equation which is inferred from Table 5.4 , The analysis found that RFC (from the t-values in Table 5.5 for the individual regression coefficients) is the only metric that has a significant effect on Integrity. The remaining variables contribute insignificantly.

5.4 Shopizer-Ecommerce Project

Table 5.6 : Variables Entered/Removed to Shopizer-Ecommerce

Variables Entered/Removed[b]			
Model	Variables Entered	Variables Removed	Method
1	RFC,CBO,NOC	.	Enter

a. All requested variables entered.

b. Dependent Variable: integrity

Table 5.7 : Summary of the Model To Shopizer-Ecommerce project

Model Summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.021[a]	.005	-.003	1.24428
a. Predictors: (Constant), RFC,CBO,NOC.				

Table 5.7 shows that Pearson Correlation is 0.021 , which means that there is a weak relationship between RFC, NOC, CBO on integrity, however , R^2 is 0.005 which indicates that 0.5% of the variation in dependent variable (integrity) can be explained by the independent variables (RFC, NOC and CBO).

Table 5.8 : Results of ANOVA Test To Shopizer-Ecommerce project

ANOVA[b]						
Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	.606	3	.202	.130	.942[a]
	Residual	1328.389	858	1.548		
	Total	1328.995	861			
a. Predictors: (Constant), RFC,CBO,NOC.						
b. Dependent Variable: integrity						

Table 5.8 shows that Sig= 0.942 more than 5% , which means that there is a no statistically significant differences between RFC, NOC, CBO and integrity.

The above mentioned table shows that Sig= 0.942 more than 5% , which means that there is a no statistically significant differences between RFC, NOC, CBO and integrity and that means a no statistically significant relationship between RFC, NOC, CBO and integrity

Table 5.9 : Individual Regression Coefficients To Shopizer-Ecommerce project

Coefficients[a]						
Model		Unstandardized Coefficients		Standardized Coefficients	T	Sig.
		B	Std. Error	Beta		
1	(Constant)	.671	.055		12.246	.000
	NOC	.006	.015	.015	.423	.672
	CBO	.001	.003	-.004	-.112	.911
	RFC	.001	.002	.018	.464	.643
a. Dependent Variable: integrity						

According to the above mentioned table , we can suggest the following equation :

Integrity = 0.671 + 0.006 NOC + 0.001 CBO+ 0.001 RFC

According to the above mentioned equation which is inferred from Table 5.9 , The analysis found (from the t-values in Table 5.8 for the individual regression coefficients) that there isn't any metric that has a significant effect on Integrity. All variables contribute insignificantly.

5.5 Payment4j Project

Table 5.10 : Variables Entered/Removed to Payment4j project

Variables Entered/Removed[b]			
Model	Variables Entered	Variables Removed	Method
1	RFC,CBO,NOC.[a]	.	Enter

a. All requested variables entered.

b. Dependent Variable: integrity

Table 5.11 : Summary of the Model To Payment4j project

Model Summary				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.122[a]	.015	.000	.22804

a. Predictors: (Constant), RFC, CBO, NOC

Table 5.11 shows that Pearson Correlation is 0.0122, which means that there is a relationship between RFC, NOC, CBO on integrity, however, R^2 is 0.015 which indicates that 1.5 % of the variation in dependent variable (integrity) can be explained by the independent variables (RFC, NOC and CBO).

Table 5.12 : Results of ANOVA Test To Payment4j project

ANOVA[b]						
Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	.151	3	.050	.968	.04 [a]
	Residual	9.933	191	.052		
	Total	10.084	194			

a. Predictors: (Constant), RFC,CBO,NOC.

b. Dependent Variable: integrity

Table 5.12 shows that Sig= 0.04 less than 5% , which means that there is a statistically significant differences between RFC, NOC, CBO and integrity, further statistical analysis will indicate which of these independent variables has a significant impact.

Table 5.13 : Individual Regression Coefficients To Payment4j project

Coefficients[a]						
Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	.826	.021		40.153	.000
	NOC	.028	.028	.072	.998	.320
	CBO	-.001	.002	-.061	-.536	.592
	RFC	.001	.001	.142	1.253	.212

a. Dependent Variable: integrity

According to the above mentioned table , we can suggest the following equation :

Integrity = 0.826 + 0.028 NOC -0.001 CBO+ 0.001 RFC

According to the above mentioned equation which is inferred from Table 5.13 , The analysis found that RFC (from the t-values in Table 5.13 for the individual regression coefficients) is the only metric that has a significant effect on Integrity. The remaining variables contribute insignificantly.

5.6 Limitations

We recognize that there are certain limitations to the results and conclusions we have presented in this thesis, and we discuss several of them in the following paragraphs.

First, our research relies on Bugs which have already been discovered and reported. The bugs that have not been discovered or publicly announced yet are not used in our study even though such information might contribute to a more precise analysis.

Second, we are aware of the fact that there are many other factors that can affect on software Integrity . Therefore, by no means, we imply that NOC,CBO, and RFC metrics should be the sole consideration when trying to measure Integrity early in the software lifecycle.

Third, The researcher measured the integrity of software projects written in java language only, in addition to this we should mention that there is a lack of tools which discover Bugs that are affecting software integrity

Finally, we acknowledge that one case study is not sufficient to draw a completely general and concrete conclusions. Some conclusions drawn from studying 3 project may not apply to other software in different domains. Nevertheless, we have substantiated our findings, provided supportive evidence about how NOC, CBO, and RFC metrics are related to Integrity.

Chapter Six

Conclusion

6.1 Conclusion

The definition of the software quality may be easy if it's divided into three aspects; process quality, functional quality, and structural quality. The process quality dramatically impacts on the value which is received to the users, sponsors, and development teams, so that these three groups have an advantage of improving such aspect. Where, the functional quality defined as the software that correctly does its tasks and the structural quality is harder than the previous ones, where it includes the code maintainability, Code testability, Code understandability, Code understandability, Code efficiency .

It is important to evaluate the quality of software through the level of after implementation. Thus, this methodology defines criteria to determine integrity's bugs in the implemented software, and Calculate the integrity, then the researcher uses CK metrics to help to evaluate the software's quality. In this research, the researcher uses two methods; the first method is based on defining and combining all the criteria which determine integrity bugs in software implementation. While the second one includes determine the integrity bugs by using static analysis tools, calculate metrics by using CKJM tool, calculate the value of integrity by using equation number 1. After that, we analyze the final results in order to know whether the metrics have a relative relationship with integrity.

The regression analysis showed that the independent variable RFC has a significant effect on Integrity and NOC has insignificant effect (low effect) while CBO is the only factor that doesn't have a significant effect on Integrity.

RFC has the most effect, then NOC to a much less degree and CBO has the lowest effect on integrity, hence the null hypothesis is rejected and the alternative hypothesis is accepted.

6.2 Future Works

Our future work will focus on the following:

The choose of (NOC, CBO, RFC) metrics based on related research . It would be interesting to explore the relationship among other metrics, Complete the research toward the full CIA Triangle tested, and Re-Evaluate the measured integrity on the other projects, Commercial and with different programming languages.

References

- Al-Badareen, A. B., Selamat, M. H., Jabar, M. A., Din, J., & Turaev, S. (2011). Software Quality Models: A Comparative Study. ***In Software Engineering and Computer Systems.*** 1(179),46-55.

- Al Mamun, M., Khanam, A., Grahn, H., & Feldt, R. (2010). "***Comparing four static analysis tools for java concurrency bugs"***. In Third Swedish Workshop on Multi-Core Computing (MCC-10) 18-19 nevamber. university of Gothenburg. Gothenburg . swedish.

- Ayewah, N., Pugh, W., Morgenthaler, J. D., Penix, J., & Zhou, Y. (2007). "***Evaluating static analysis defect warnings on production software***". In Proceedings of the 7th ACM SIGPLAN-SIGSOFT**.** workshop on Program analysis for software tools and engineering .13-14 june. San diego. USA.

- Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. ***Software Engineering IEEE Transactions*** .28(1), 4-17.

- Basili, V. R. (1993). Applying the Goal/Question/Metric paradigm in the experience factory. Software Quality Assurance and Measurement: A Worldwide Perspective, ***Journal of the National Cancer Institute.*** 21-44.

- Bishop, M. A. (2002). ***The art and science of computer security***. Addison-Wesley Longman Publishing Co. Boston: USA.

- Blake, S. Q. (2000). "***The Clark-Wilson Security Model"***. (Online), Available: http://www. lib. iup. edu/comscisec/SANSpapers/blake. htm.

- Boehm, B. W., Brown, J. R., & Kaspar, H. (1978). ***Characteristics of software quality,*** Amsterdam : North-Holland.

- Chandra, S., & Khan, R. A. (2009). A Methodology to Check Integrity of a Class Hierarchy. ***International Journal of Recent Trends in Engineering***, 2(4), 83-85.

- Chandra, S., & Khan, R. A. (2013). An Empirical Validation of Integrity Risk Factor Metric: An Object-Oriented Design Perspective. ***International Journal of Advanced Research in Computer Science and Software Engineering***, 3(8), 528-537.

- Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. ***Software Engineering, IEEE Transactions.*** 20(6), 476-493.

- Chowdhury, I. (2011). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. ***Journal of Systems Architecture***, *57*(3), 294-313.

- ***Coupling Between Objects*** .(2014). (Online), Available: http://www.arisa.se/compendium/node105.html#metric:CBO .

- David Basin, J¨urgenDoser&TorstenLodderstedt (2006): ***Model driven security: From UML models to access control infrastructures***. ACM Trans. Software. Engineering Method. 15, (39–91) . (online), Available: http: //doi.acm.org/10.1145/1125808.1125810 .

- De Castro, V., Musicante, M. A., da Costa, U. S., de Souza Neto, P. A., & Vargas-Solar, G. (2014). Supporting Non-functional Requirements in Services Software Development Process: An MDD Approach. In SOFSEM 2014: Theory and Practice

of Computer Science (199-210). ***Springer International Publishing.*** London : UK.

- Dromey, R. G. (1995). A model for software product quality. ***Software Engineering, IEEE Transactions***, 21(2), 146-162.

- Fedotova, O., Teixeira, L., & Alvelos, H. (2013). Software Effort Estimation with Multiple Linear Regression: Review and Practical Application. ***Journal of Information Science and Engineering***, 29(5), 925-945.

- ***FindBugs - find bugs in java programs***.(2014). (Online), Available: http://FindBugs.sourceforge.net/ .

- Idre, ***SPSS Annotated Output Regression Analysis***.(2014). (Online), Available: http://www.ats.ucla.edu/stat/spss/output/reg_spss.htm.

- James, M. (1977). ***Factor in software quality***, NY: USA.

- Jiang, Y., Cuki, B., Menzies, T., & Bartlow, N. (2008***). Comparing design and code metrics for software quality prediction.*** In Proceedings of the 4th international workshop on Predictor models in software engineering (11-18 may).ACM. San diego. USA.

- Jureczko, M., & Spinellis, D. (2010). ***Using object-oriented design metrics to predict software defects. Models and Methods of System Dependability***, (69-81) .(Online), Available: http://www.dmst.aueb.gr/dds/pubs/conf/2010-DepCoS-RELCOMEX-ckjm-defects/html/JS10.pdf.

- Kayarvizhy, N., & Kanmani, S. (2011). An Automated Tool for Computing Object Oriented Metrics Using XML. In Advances in Computing and Communications (69-79). ***Springer Berlin Heidelberg.*** London: UK.

- Khan, S. A., & Khan, R. A. (2012). Integrity quantification model for object oriented design. ***ACM SIGSOFT Software Engineering Notes***, 37(2), 1-3.

- Livshits, V. B., & Lam, M. S. (2005, August). ***Finding Security Vulnerabilities in Java Applications with Static Analysis***. (18-18). (Online), Available: https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits_html/.

- ***Lines of Code***.(2014). (Online), Available: http://www.arisa.se/compendium/node91.html#metric:LOC.

- Martin, R. (1994). ***OO design quality metrics***. An analysis of dependencies.(1-8). (Online), Available: http://www.cin.ufpe.br/~alt/mestrado/oodmetrc.pdf.

- ***Microsoft "Security Threats***" .(2014). (Online), Available: http://technet.microsoft.com/en-us/library/cc723507. aspx#mainSection .

- Momotaz, S. (2010). ***Tool Support for Testing Java Generics***,*(Unpublished d*octoral dissertation), Texas Tech University.Texas: USA.

- ***Multiple Linear Regression*** . (2014) . (Online), Available: http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm .

- ***Number Of Children***. (2014). (Online), Available: http://www.arisa.se/compendium/node102.html#metrics:NOC .

- ***ParaSoft Jtest, Parasoft Corporation***, (2014). (Online), Available: http://www.testingfaqs.org/t-static.htm#Jtest .

- Paul B, Zhen R D, Jens G, Ystein H,(2007). **Ina Schieferdecker& Clay Williams Model-Driven Testing**: Using the UML Testing Profile. (Online), Available: http://dx.doi.org/10.1007/978-3-540-72563-3 .

- ***Preventing SQL Injection in java***. (2015) . (Online), Available: https://www.owasp.org/index.php/Preventing_SQL_Injection_in_Java #Example_of_SQL_injection.

- ***Response For a Class***. (2014). (Online), Available: http://www.arisa.se/compendium/node98.html#metric:RFC.

- Sattarova, F. Y., & Kim, T. H. (2007). IT security review: Privacy, protection, access control, assurance and system security. International ***Journal of Multimedia and Ubiquitous Engineering***, 2(2), 17-32.

- Schieferdecker, I., Grossmann, J., & Schneider, M. (2012). Model-based security testing. ***Electronic Proceedings in Theoretical Computer Science***,((MBT 2012). 1-12 Cornell University. NY, USA.

- Senthil, R., Kushwaha, D. S., & Misra, A. K. (2008). An Extended Component Model and its evaluation for Reliability & Quality**.** ***Journal of Object Technology***,7(7), 109-129.

- Shaik, A., Reddy, C. R. K., Manda, B., Prakashini, C., & Deepthi, K. (2010). Metrics for object oriented design softw\re systems: a survey. ***Journal of Emerging Trends in Engineering and Applied Sciences***, 1(2), 190-198.

- Singer, J., Marion, S., Brown, G., Jones, R., Luján, M., Ryder, C., & Watson, I. (2008). ***an information theoretic evaluation of software metrics for object lifetime***

prediction. in 2nd workshop on statistical and machine learning approaches to ARchitectures and compilaTion (SMART'08). (Online), Available: https://kar.kent.ac.uk/23960/1/InfoRider.pdf

- ***Software Quality Characteristics***. (2014). (Online), Available: http://www.sqa.net/iso9126.html.

- Spathoulas,(2014) . ***Assessing Tools for Finding Bugs in Concurrent Java***. (Online), Available: http://homepages.inf.ed.ac.uk/dts/students/spathoulas/spathoulas.pdf.

- Spinellis, D.: ***CKJM—Chidamber and Kemerer Java metrics.*** (2005) (Online), Available: http://www.spinellis.gr/sw/CKJM /.

- VanSuetendael, N., & Elwell, D. (1991). ***Software Quality Metrics***. Computer Recourse Management INC Pleasantville, NJ:USA.

- ***VisualCodeGrepper,sourceforge***.(2014).(Online), Available : http://sourceforge.net/projects/visualcodegrepp/files/ .

- ***Visibleimpact***, (2014):).(Online), Available : .http://www.visibleimpact.com/docs/Parasoft-CiscoCaseStudy .

Appendix

Appendix(A)

Criteria of defining Integrity Bugs:

In this chapter we will define the criteria in which in depend to find and integrity error . as main in the related work of khan the selection of the criteria of his algorithm was manual . In the following table we description the calculated in relevant bugs define the description in an implemented source code and how this bugs are relevant to Integrity issues as define in the chapter 3 section in addition to the available tools . more details on how to find the bugs allowing with related reference were moved to appendix ( ) for please refers for their model.

Number	Integrity Relevant Bugs	Description	Relevance	Reference
1.	Prevent security vulnerability (custom rule)	This is a template rule that, if customized, will look for places in code where possibly tainted data (as defined by the rule parameters) is used without first being checked for benignity and validated before being used.	category: Input-Based Attacks In general, in order to prevent application security flaws, any data passed into dangerous methods should be checked for benignity and validated before being used. If this requirement is not met, then a malicious user can potentially use such flaws for various malicious actions (for example, actions that allow him to gain control of a database or of resources from the system the application is running on).	N/A
2.	Prevent exposure of sensitive data	This rule detects cases when sensitive internal data is made available to the end-user. This makes the application less secure because the attacker may obtain program information that allows him to construct a request that breaks the normal program flow and/or provides him with higher access privileges than he should have. Sources of sensitive data: * toString(): Often toString() is implemented to display information internal to the object. This should not be made available to the end-user. * Reflection: Information about the internal structure of the program should be hidden from the user since it may	category: Exposing Sensitive Data 1. Enforces 'A6 - Information Leakage and Improper Error Handling', #6 from the OWASP Top 10 2007 list. Sensitive information leakage may be a critical problem for applications that need to be secure. Many security attacks occur as a result of an attacker gaining insight into the structure of an application, then using this insight to devise input data that makes the application behave in an abnormal way and ends up granting the attacker higher access privileges than he should have. Thus, for security critical applications, it's important to keep information about the program's internal structure hidden-- so that potential attackers cannot access it. In particular, exceptions, stack traces, method and class names	OWASP Top 10 2007 (A6 - Information Leakage and Improper Error Handling): http://www.owasp.org/index.php/Top_10_2007 Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/information_leakage.shtml PCI DSS Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration:

		be used to guide attacks. Objects of the following classes are considered sensitive: * java.lang.Class * java.reflect.AccessibleObject * java.lang.reflect.Field * java.lang.reflect.Method * java.lang.reflect.Constructor * Exceptions: Thrown exceptions should be hidden from the user. Any instance of java.lang.Throwable is sensitive. * Environment: Environment variables may contain information which should be hidden from the normal user of the system. * System Properties The rule checks that sensitive data do not leak throw the following means of output/UI: * AWT * Swing * SWT * JFace * Servlets * Apache ECS	should never be made available to the end user. 2. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #3: "Protect stored cardholder data" and Requirement #6: "Develop and maintain secure systems and applications".	http://cwe.mitre.org/data/definitions/209.html http://cwe.mitre.org/data/definitions/497.html
3.	Protect against Command injection	This rule detects cases when data coming directly from the end-user can influence the code which is executed (for example, to form the name of the file to be executed). This rule triggers when tainted data are passed to the following methods: java.lang.Runtime * exec(String) * exec(String, String[]) * exec(String, String[], File) * exec(String[]) * exec(String[], String[]) * exec(String[], String[], File) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks 1. Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list. If some tainted data will appear in an executed file name without verification, it may allow the execution of custom malicious code which could damage the system. 2. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Cigital Java Security Rulepack # 58: http://www.cigital.com/securitypack/view/index.html CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') http://cwe.mitre.org/data/definitions/78.html
4.	Protect against Jakarta	This rule detects cases when data coming directly from the end-user is used in a Jakarta Digester query or evaluation.	category: Input-Based Attacks Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list.	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/ind

	Digester injection	This may result in exposure of confidential data and execution of dangerous methods. This rule triggers when tainted data is passed to the following methods: org.apache.commons.digester.Rules * add(String, Rule) * match(String, String) * setNamespaceURI(String) org.apache.commons.digester.Rule * setNamespaceURI(String) org.apache.commons.digester.RegexMatcher * match(String, String) org.apache.commons.digester.Substitutor * substitute(String) org.apache.commons.digester.SimpleRegexMatcher * match(String, String) org.apache.commons.digester.Digester * addBeanPropertySetter(String) * findNamespaceURI(String) * getFeature(String) * getProperty(String) * parse(String) * peek(String) * pop(String) * push(String) * pushParams(String) * register(String, String) * resolveEntity(String, String) * setProperty(String, Object) * setPublicId(String) * setRuleNamespaceURI(String) * setSchema(String) * setSchemaLanguage(String) Rules constructors * org.apache.commons.digester.BeanPropertySetterRule * org.apache.commons.digester.CallMethodRule * org.apache.commons.digester.CallParamRule * org.apache.commons.digester.FactoryCreateRule * org.apache.commons.digester.ObjectCreateRule * org.apache.commons.digester.ObjectParamRule * org.apache.commons.digester.PathCallParamRule * org.apache.commons.digester.SetNestedPropertiesRule *	The Digester component provides a common implementation for reading XML configuration files to provide initialization of various Java objects within the system. Basically, the Digester package lets you configure an XML -> Java object mapping module, which triggers certain actions (called rules) whenever a particular pattern of nested XML elements is recognized. A rich set of predefined rules is available for your use, or you can also create your own rules. Rule constructors, matchers and properties-setting methods here are very sensitive to tainted data because they can lead to real object creation or system properties modification. Only checked data should reach the Digester component. 2. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	ex.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml

		org.apache.commons.digester.SetNextRule * org.apache.commons.digester.SetPropertiesRule * org.apache.commons.digester.SetPropertyRule * org.apache.commons.digester.SetRootRule * org.apache.commons.digester.SetTopRule Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.		
5.	Protect against Environment injection	This rule detects cases when data coming directly from the end-user is used unchecked to define system properties. This rule triggers when tainted data is passed to the following methods: java.lang.System * String setProperty(String, String) * void setProperties(Properties) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks If unverified data reaches system properties, it could allow attackers to damage the system. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/78.html
6.	Protect against File contents injection	This rule detects cases when data coming directly from the end-user is put into a file without being checked. This rule triggers when tainted data is passed to the following methods: * FileOutputStream.write(...) * FileWriter.write(...) * File.write(...) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For	category: Input-Based Attacks Data that is written to a file can be used to form other requests (such as SQL or XML or XPath etc.). Consequently, an attacker can provide malicious data as file contents which can lead to executing dangerous requests and/or revealing information that was intended to be private/secure. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml

		details, see the PARAMETERS section.		
7.	Protect against File names injection	This rule detects cases when data coming directly from the end-user is used to form the name of a file, which is then accessed. This rule triggers when tainted data is passed to the following methods: * File.File(...) * FileInputStream.FileInputStream(...) * FileOutputStream.FileOutputStream(...) * FileReader.FileReader(...) * FileWriter.FileWriter(...) * Paths.get(...) * FileSystem.getPath(...) * Formatter.Formatter(...) * ZipFile.ZipFile(...) * JarFile.JarFile(...) * javax.activation.FileDataSource.FileDataSource(...) * javax.xml.parsers.DocumentBuilder.parse(...) * javax.servlet.ServletContext.getRequestDispatcher(...) * javax.servlet.ServletContext.getResource*(...) * org.jaxen.Navigator.getDocument(...) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks If unverified data appears in file names, then an attacker could potentially gain access to any file on the system by providing specially-prepared malicious data. Enforcing this rule will help to protect against: - the OWASP 2007 Top 10 application vulnerability "A3 - Malicious File Execution" - the path traversal vulnerability, which is part of CWE 2010 Top 25 list (CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')) This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications." Specifically, this rule tests for Issue 6.5.6: "Injection flaws."	OWASP Top 10 2007 (A3 - Malicious File Execution): http://www.owasp.org/index.php/Top_10_2007-A3 CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') http://cwe.mitre.org/data/definitions/22.html CWE-73: External Control of File Name or Path http://cwe.mitre.org/data/definitions/73.html PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml
8.	Protect against JXPath injection	This rule detects cases when data coming directly from the end-user is used in an XML query formed by JXpath (Jakarta XPath realization), which may result in exposure of confidential data. This rule triggers when tainted data is passed to the following methods: org.apache.commons.jxpath.CompiledExpression * setValue(JXPathContext, Object) org.apache.commons.jxpath.Container * setValue(Object) org.apache.commons.jxpath.DynamicPropertyHandler * getProperty(Object, String) org.apache.commons.jxpath.Function * invoke(ExpressionContext, Object)	category: Input-Based Attacks Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list. The org.apache.commons.jxpath package defines a simple interpreter of an expression language called XPath. JXPath applies XPath expressions to graphs of objects of all kinds: JavaBeans, Maps, Servlet contexts, DOM etc. (including mixtures thereof). Consider this example: Address address = (Address)JXPathContext.newContext(vendor).getValue("locations[address/zipCode='90210']/address"); This XPath expression is equivalent to the following Java code: Address address = null; Collection locations = vendor.getLocations();	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') https://cwe.mitre.org/data/definitions/643.html

		org.apache.commons.jxpath.Functions
		* getFunction(String, String, Object[])
		
		org.apache.commons.jxpath.IdentityManager
		* getPointerByID(JXPathContext, String)
		
		org.apache.commons.jxpath.JXPathBeanInfo
		* getPropertyDescriptor(String)
		
		org.apache.commons.jxpath.KeyManager
		* getPointerByKey(JXPathContext, String, String)
		
		org.apache.commons.jxpath.Variables
		* declareVariable(String, Object)
		* undeclareVariable(String)
		* getVariable(String)
		
		org.apache.commons.jxpath.JXPathContext
		* compile(String)
		* createPath(String)
		* createPathAndSetValue(String, Object)
		* getNamespaceURI(String)
		* getPointer(String)
		* getPointerByID(String)
		* getPointerByKey(String)
		* getValue(String)
		* iterate(String)
		* iteratePointers(String)
		* newContext(String)
		* registerNamespace(String, String)
		* removeAll(String)
		* removePath(String)
		* selectNodes(String)
		* selectSingleNode(String)
		* setValue(String, Object)
		
		org.apache.commons.jxpath.MapDynamicPropertyHandler
		* getProperty(Object, String)
		* getPropertyNames(Object)
		* setProperty(Object, String, Object)
		
		org.apache.commons.jxpath.PackageFunctions
		* getFunction(String, String, Object[])
		
		org.apache.commons.jxpath.ri.NamespaceResolver
		* getNamespaceURI(String)
		* getPrefix(String)
		* registerNamespace(String, String)
		
		org.apache.commons.jxpath.ri.Parser
		* parseExpression(String, Compiler)
		
		org.apache.commons.jxpath.ri.Compuler
		* all methods
		
		Data from the following data sources are considered tainted:
		* Parameters of remote methods and
			Iterator it = locations.iterator();
			while (it.hasNext()){
			Location location = (Location)it.next();
			String zipCode = location.getAddress().getZipCode();
			if (zipCode.equals("90210")){
			address = location.getAddress();
			break;
			}
			}
			If an application uses run-time JXPath query construction, embedding unsafe user input into the query, it may be possible for the attacker to inject data into the query so that the newly-formed query will be parsed in a way that the programmer did not intend. Consequently, it is important to prevent potentially tainted data from reaching JXPath methods that can be used for creating/modifying objects or modifying any properties.
			
			This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	

		entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.		
9.	Protect against LDAP injection	This rule detects cases of probable LDAP injection when possibly tainted data reaches methods that execute LDAP queries. When an application uses data provided by the user or by some unverified data source to construct LDAP search queries and does not verify/validate such data before its use, it is possible for an attacker to alter the construction of the LDAP statements in a way that the developer did not intend. This can cause serious security problems where the permissions grant the rights to query, modify, or remove anything inside the LDAP tree. This rule triggers when tainted data is passed to the following 'dangerous' methods: javax.naming.directory.DirContext * search(String name, String filter, SearchControls cons) * search(Name name, String filter, SearchControls cons) Dangerous parameter is 'filter'. Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list. If data can appear in an LDAP query without having been previously validated, it may allow a malicious user to take control of the database. This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 Understanding LDAP: http://www.redbooks.ibm.com/redbooks/SG244986.html Introduction to LDAP Security: http://www.severus.org/sacha/docperso/intro_to_ldap_tisc.htm Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml RFC 1960 - A String Representation of LDAP Search Filters: http://www.ietf.org/rfc/rfc1960.txt PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') http://cwe.mitre.org/data/definitions/90.html
10.	Protect against Library injection	This rule detects cases when data coming directly from the end-user is used to form the name of the library which is loaded. This rule triggers when tainted data is passed to the following methods: java.lang.System * void load(String) * void loadLibrary(String) java.lang.Runtime * void load(String) * void loadLibrary(String) Data from the following data sources are	category: Input-Based Attacks If tainted data appears in the loading library name without verification, it may permit execution of custom malicious code which could damage the system. Enforcing this rule will help to protect against the OWASP 2007 Top 10 application vulnerability "A3 - Malicious File Execution". This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	OWASP Top 10 2007 (A3 - Malicious File Execution): http://www.owasp.org/index.php/Top_10_2007-A3 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-114: Process Control http://cwe.mitre.org/data/definitions/114.html

		considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.		
11.	Protect against log forging	This rule detects cases of log forging when possibly tainted data is written to the log. Depending on the nature of the application, the log may be reviewed manually or with a tool that automatically culls the logs for important events or trending information. Tainted data written to the log without proper validation may change the log format, break the automated log parser, or cover an attacker's tracks. This rule supports the following logging APIs: * log4j (http://logging.apache.org/log4j/) * logback (http://logback.qos.ch/) * SLF4J (http://www.slf4j.org/) * java.util.logging (http://download.oracle.com/javase/7/docs/api/) * Commons Logging (http://commons.apache.org/logging/) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks Writing tainted user data to a log can allow an attacker to forge log entries or inject malicious content into the logs. Enforcing this rule will help protect against the OWASP 2013 Top 10 application vulnerability "A1-Injection". This rule also helps enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.1: "Injection flaws".	OWASP Top 10 2013-A1-Injection: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-117: Improper Output Neutralization for Logs http://cwe.mitre.org/data/definitions/117.html
12.	Protect against network resource injection	This rule detects cases of probable network resource injection. It reports a violation when possibly tainted data that represents a network resource property (such as host name, IP-address, port number, path or query string) is passed as a parameter to a method that can allocate a resource directly or can create a resource descriptor (like URL or URI) that is to be used for the allocation. When an application permits user input (or input from an unverified data source) to define a resource used by the application, and does not validate such data before its use, this data can be manipulated to execute or access different resources. This rule triggers when tainted data is passed to the	category: Input-Based Attacks If an unvalidated resource property can be used by an application, the attacker can gain direct access to the resources of the system that the application is running on (see EXAMPLE section), or force the application to execute other remote resources-- thus changing the behavior of the application in a way that the developer did not intend. Enforcing this rule will help to protect against the OWASP 2007 Top 10 application vulnerability "A3 - Malicious File Execution". This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	http://www.owasp.org/index.php/Resource_Injection OWASP Top 10 2007 (A3 - Malicious File Execution): http://www.owasp.org/index.php/Top_10_2007-A3 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/601.html

		following 'dangerous' methods: java.net.DatagramPacket * setPort(int port) * constructor(s) that accept port number java.net.DatagramSocket * connect(InetAddress address, int port) * constructor(s) that accept port number java.net.DatagramSocketImpl * connect(InetAddress address, int port) * bind(int port, InetAddress addr) java.net.InetSocketAddress * createUnresolved(String host, int port) * constructor(s) that accept port number java.net.MulticastSocket * constructor(s) that accept port number java.net.ServerSocket * constructor(s) that accept port number java.net.Socket * constructor(s) that accept port number java.net.SocketImpl * connect(String host, int port) * connect(InetAddress address, int port) * bind(InetAddress host, int port) java.net.URI * resolve(String str) * create(String str) * constructor(s) that accept port number java.net.URL * constructor(s) that accept port number java.net.URLStreamHandler * setURL(URL u, String protocol, String host, int port, String authority, String userInfo, String path, String query, String ref) * parseURL(URL u, String spec, int start, int limit) java.net.InetAddress * getAllByName(String host) * getByAddress(String host, byte[] addr) * getByAddress(byte[] addr) * getByName(String host) java.net.NetworkInterface * getByName(String name) java.rmi.activation.Activatable * all exportObject(...) methods that accept port number * constructor(s) that accept port number java.rmi.registry.LocateRegistry * all getRegistry(...) methods that accept port number * all createRegistry(...) methods that accept port number		

		java.rmi.server.RMIClientSocketFactory * createSocket(String host, int port) java.rmi.server.RMIServerSocketFactory * createServerSocket(int port) java.util.logging.SocketHandler * constructor(s) that accept port number javax.net.ServerSocketFactory * all createServerSocket(...) methods that accept port number javax.net.SocketFactory * all createSocket(...) methods that accept port number javax.net.ssl.SSLSocket * constructor(s) that accept port number javax.net.ssl.SSLServerSocket * constructor(s) that accept port number Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.		
13.	Protect against HTTP response splitting	This rule detects cases of probable HTTP response splitting vulnerabilities. This rule triggers when tainted data is passed to the following methods: javax.servlet.http.HttpServletResponse * void sendRedirect(...) methods * void addCookie(...) methods * void addIntHeader(...) methods * void addDateHeader(...) methods * void setHeader(...) methods * void setIntHeader(...) methods * void setDateHeader(...) methods * void setStatus(...) methods javax.faces.context.ExternalContext * void redirect(...) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network In order to protect from HTTP response splitting it must be ensured that tainted data cannot be passed to dangerous methods and any user input is properly encoded or cleaned. Such encoding can be performed by one of the following	category: Input-Based Attacks Helps to enforce 'A3-Cross-Site Scripting (XSS)' and 'A8-Cross-Site Request Forgery (CSRF)', #3 and #8 from the OWASP Top 10 2013 list. HTTP response splitting is a type of vulnerability which occurs when tainted data entering a web application through an untrusted source (for example from a HTTP request) is included in an HTTP response without being validated. A successful attack may be performed by including CR (%0d or \r) and LF (%0a or \n) characters into the data which gets into an HTTP header. These characters, followed by the specially crafted string, may be used by an attacker to include arbitrary headers in the HTTP response as well as to create additional HTTP responses with arbitrary content. HTTP response splitting vulnerabilities may be used to perform XSS attacks, cross-user defacement, Web cache poisoning, and page hijacking.	OWASP HTTP Response Splitting: http://www.owasp.org/index.php/HTTP_Response_Splitting Wikipedia HTTP Response Splitting: http://en.wikipedia.org/wiki/HTTP_response_splitting Introduction to HTTP Response Splitting: http://www.securiteam.com/securityreviews/5WP0E2KFGK.html HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics: http://www.cgisecurity.com/lib/whitepaper_httpresponse.pdf Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/79.html http://cwe.mitre.org/data/definitions/80.html http://cwe.mitre.org/data/de

		methods whose return values are safe even if the data passed to the methods in parameters are tainted: java.net.URLEncoder * String encode(...) methods javax.servlet.http.HttpServletResponse * String encodeURL(...) method * String encodeUrl(...) method * String encodeRedirectURL(...) method * String encodeRedirectUrl(...) method Additional sources of tainted data and validating methods (in addition to the standard encoding methods) can be defined by parameterizing the rule. For details, see the PARAMETERS section.		finitions/113.html http://cwe.mitre.org/data/definitions/180.html http://cwe.mitre.org/data/definitions/352.html http://cwe.mitre.org/data/definitions/601.html
14.	Protect against Reflection injection	This rule detects cases when data coming directly from the end-user can influence the code that is executed-- for instance, it is used to form the name of the class which is loaded or the method that is invoked. This rule triggers when tainted data is passed to the following methods: java.lang.ClassLoader * loadClass(String) java.lang.Class * forName(String) * getDeclaredField(String) * getDeclaredMethod(String, Class[]) * getField(String) * getMethod(String, Class[]) * getResource(String) * getResourceAsStream(String) java.lang.reflect.InvocationHandler * invoke(Object, Method, Object[]) java.lang.reflect.AccessibleObject * setAccessible(AccessibleObject[], boolean) * void setAccessible(boolean flag) java.lang.reflect.Field * all get* methods java.lang.reflect.Method * invoke(Object, Object[]) java.lang.reflect.ReflectPermission * ReflectPermission(String) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For	category: Input-Based Attacks If some tainted data appears in a loaded class name without verification, it may permit execution of custom malicious code which could damage the system.	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') http://cwe.mitre.org/data/definitions/470.html

		details, see the PARAMETERS section.		
15.	Protect against SQL injection	This rule detects cases of probable SQL injection when possibly tainted data reaches methods that execute or prepare SQL queries, retrieve connections, etc. When an application uses data provided by the user (or by some unverified data source) to construct SQL queries and does not verify/validate such data before its use, it is possible for an attacker to alter the SQL statements in a way that the developer did not intend. As a result, the attacker can take total control of the database or even execute commands on the system. This rule triggers when tainted data is passed to the following 'dangerous' methods: java.sql.DriverManager * getConnection(String url) java.sql.Connection * prepareCall(...) * prepareStatement(...) * setSavepoint(String) * nativeSQL(String) java.sql.Statement * addBatch(String) * execute(...) * executeQuery(String) * executeUpdate(...) * setCursorName(String) org.springframework.jdbc.datasource.AbstractDriverBasedDataSource * setUrl(String) org.springframework.jdbc.datasource.DriverManagerDataSource * DriverManagerDataSource(...) org.springframework.jdbc.datasource.SimpleDriverDataSource * SimpleDriverDataSource(...) org.springframework.jdbc.datasource.SingleConnectionDataSource * SingleConnectionDataSource(...) org.springframework.jdbc.core.JdbcOperations * execute(...) * query.*(...) * update(...) * batchUpdate(...) org.springframework.jdbc.core.CallableStatementCreatorFactory * CallableStatementCreatorFactory(...) org.springframework.jdbc.core.PreparedStatementCreatorFactory * PreparedStatementCreatorFactory(...)	category: Input-Based Attacks Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list. If data can appear in an SQL query without being validated, there is a chance for a malicious user to take control of the database. This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws (for example, structured query language (SQL) injection)".	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/sql_injection.shtml PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') http://cwe.mitre.org/data/definitions/89.html

		* newPreparedStatementCreator(...) org.springframework.jdbc.core.namedparam.NamedParameterJdbcOperations * execute(...) * query.*(...) * update(...) org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate * getParsedSql(...) * getPreparedStatementCreator(...) org.springframework.jdbc.core.support.JdbcBeanDefinitionReader * loadBeanDefinitions(...) org.springframework.jdbc.object.BatchSqlUpdate * BatchSqlUpdate(...) org.springframework.jdbc.object.MappingSqlQuery * MappingSqlQuery(...) org.springframework.jdbc.object.MappingSqlQueryWithParameters * MappingSqlQueryWithParameters(...) org.springframework.jdbc.object.SqlCall * SqlCall(...) org.springframework.jdbc.object.SqlFunction * SqlFunction(...) org.springframework.jdbc.object.SqlOperation * newPreparedStatementCreator(...) org.springframework.jdbc.object.SqlQuery * SqlQuery(...) org.springframework.jdbc.object.SqlUpdate * SqlUpdate(...) org.springframework.jdbc.object.UpdatableSqlQuery * UpdatableSqlQuery(...) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.		
16.	Protect against XML data injection	This rule detects cases when data coming directly from the end-user is used to compromise an XML document. This rule triggers when tainted data is passed	category: Input-Based Attacks Data stored to XML can be used for different purposes. XML can be used even as an alternate	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standard

		to the following methods: org.w3c.dom.Node * setNodeValue(String) org.w3c.dom.Attr * setValue(String) org.w3c.dom.Element * setAttribute(String, String) org.w3c.dom.CharacterData * appendData(String) * insertData(int, String) * replaceData(int, int, String) * setData(String) org.w3c.dom.Document * createAttribute(String) * createCDATASection(String) * createElement(String) * createentityReference(String) * createProcessingInstruction(String, String) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	database / storage system. Consequently, data from XML can be used to form some other requests (like SQL, XML, XPath, etc.) As a result, an attacker can provide malicious data and that can lead to executing dangerous requests, revealing some private or secure information. This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6: "Injection flaws".	s/pci_dss.shtml CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) http://cwe.mitre.org/data/definitions/80.html
17.	Protect against XPath injection	This rule detects cases when data coming directly from the end-user is used in an XML query which may result in exposing confidential data. This rule triggers when tainted data is passed to the following methods: W3C definition org.w3c.dom.xpath.XPathEvaluator * createExpression(String, XPathNSResolver) * evaluate(String, Node, XPathNSResolver, short, Object) org.w3c.dom.xpath.XPathExpression * evaluate(Node, short, Object) org.w3c.dom.xpath.XPathNSResolver * lookupNamespaceURI(String) JavaX definition javax.xml.xpath.XPath * compile(String) * evaluate(String) javax.xml.xpath.XPathExpression * evaluate(String)	category: Input-Based Attacks Enforces 'A1-Injection', #1 from the OWASP Top 10 2013 list. XPath is a language that is used to refer to parts of an XML document. The XPath language provides a simple, concise syntax for selecting nodes from an XML document. Consequently, an application can use it to browse an XML document using a provided query, or as part of some larger operation (like applying an XQuery to an XML document). The syntax of XPath bears some resemblance to an SQL query, and indeed, it is possible to form SQL-like queries on an XML document using XPath. It is definitely possible to create an application that will use run-time XPath query construction, with some user-provided data in the query; this allows the attacker to inject the query with data that will lead the application to execute in a way that the programmer did not intend. This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.6:	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection') https://cwe.mitre.org/data/definitions/643.html

		javax.xml.xpath.XPathFunction * evaluate(String) javax.xml.xpath.XPathFactory * newInstance(String) * setFeature(String) org.apache.xpath definition javax.xml.xpath.ExtensionsProvider * extFunction(String, String, Vector, Object) javax.xml.xpath.XPathFactory * create(String, SourceLocator, PrefixResolver, int) javax.xml.xpath.Arg * Arg(QName, String, boolean) * setExpression(String) javax.xml.xpath.CachedXPathAPI * eval(Node, String) * selectSingleNode(Node, String) javax.xml.xpath.SourceTreeManager * resolveURI(Node, String) * getSourceTree(String, String, SourceLocator, XPathContext) javax.xml.xpath.SourceTree * SourceTree(int, String) org.jaxen definition org.jaxen.Navigator * parseXPath(String) org.jaxen.dom4j.Dom4jXPath * Dom4jXPath(String) org.jaxen.dom.DOMXPath * DOMXPath(String) org.jaxen.javabean.JavaBeanXPath * JavaBeanXPath(String) org.jaxen.jdom.JDOMXPath * JDOMXPath(String) org.jaxen.xom.XOMXPath * XOMXPath(String) org.jaxen.saxpath.XPathReader * parse(String) Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For	"Injection flaws".	

		details, see the PARAMETERS section.		
18.	Protect against XSS vulnerabilities	This rule detects cases of probable XSS vulnerabilities. This rule triggers when tainted data is passed to the following methods: javax.servlet.ServletOutputStream * void print(...) methods * void println(...) methods * void write(...) methods java.io.PrintWriter * void print(...) methods * void println(...) methods * void write(...) methods Data from the following data sources are considered tainted: * Parameters of remote methods and entry point methods * Native methods * Non-validating Struts forms * Network Additional sources of tainted data can be defined by parameterizing the rule. For details, see the PARAMETERS section.	category: Input-Based Attacks Helps to enforce 'A3-Cross-Site Scripting (XSS)' and 'A8-Cross-Site Request Forgery (CSRF)', #3 and #8 from the OWASP Top 10 2013 list. Cross-site scripting (XSS) is a type of attack that makes a web site show attacker-provided executable code, which loads in a user's browser. Such code can be written in HTML/JavaScript or VBScript, ActiveX, Java, Flash and so on. When such code is executed in the user's browser, it is run within the security context (or zone) of the hosting web site. Consequently, this code will get privileges far beyond what it should have. With such privileges, this code will have the ability to read, modify, and transmit any sensitive data accessible by the browser. As a result, users attacked by XSS could have their browsers redirected to another location that will look exactly like the original one, but will be used to hijack (steal cookies from) their account or steal login / password information. There are two types of cross-site scripting attacks: non-persistent and persistent. Non-persistent attacks require a user to visit a specially-crafted link that is laced with malicious code. Upon visiting the link, the code embedded in the URL will be echoed and executed within the user's web browser. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to click on any link, just simply view the web page containing the code.	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 Web Application Security Consortium: http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') http://cwe.mitre.org/data/definitions/79.html
19.	Do not set custom security managers outside of the 'main' method	This rule identifies any invocation of 'System. Set Security Manager' outside 'main' if it loads a custom Security Manager. An error is reported for each occurrence.	category: Backdoor Vulnerabilities category: Code Quality Enforce code access control. Enforcing this rule will help to protect against the OWASP Top 10 2013 application vulnerability "A2-Broken Authentication and Session Management".	OWASP Top 10 2013 (A2-Broken Authentication and Session Management): https://www.owasp.org/index.php/Top_10_2013-Top_10
20.	Ensure all sensitive method invocations are logged	This rule identifies code that does not log sensitive method invocations. An error is reported if some sensitive method invocations-- for instance, 'login' and 'logout' from 'javax.security.auth.login.LoginContext'-- are not logged in the previous or next statements.	category: Backdoor Vulnerabilities This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #10: "Track and monitor all access to network resources and cardholder data". See BENEFITS section for more of the SECURITY RELEVANCE.	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml
21.		This rule identifies code that attempts to access or set System properties through	category: Backdoor Vulnerabilities category: Malicious Code	CWE-15: External Control

	Do not access or set System properties	"java.lang.System". An error is reported for each occurrence.	Code which accesses or sets System properties may be malicious. See BENEFITS for more information.	of System or Configuration Setting http://cwe.mitre.org/data/definitions/15.html
22.	Do not access the class loader in a web component	Web components should not access the class loader. This rule will flag any call to the method 'Class.getClassLoader()' in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Backdoor Vulnerabilities Accessing the class loader is a suspicious security pattern, as it is typically unnecessary for a web application. Use of the class loader may indicate malicious code.	N/A
23.	Do not call 'Socket. set SocketImpl Factory()' or 'URL.set URL Stream Handler Factory()' in a web component	Web components should not set network socket factories or URL stream handler factories. This rule will flag a violation for each call to 'Socket.setSocketImplFactory()' or 'URL.setURLStreamHandlerFactory()' in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Backdoor Vulnerabilities By setting default socket factories, a web application will interfere with the web-communication provided by the web server. This is likely to have unintended consequences and side-steps the security provided by the container.	N/A
24.	Avoid using dynamically loaded classes in "privileged" code blocks	This rule flags a violation if the 'run()' method of a "java.security.PrivilegedAction" or "java.security.PrivilegedExceptionAction" implementation calls a method commonly used for dynamic class loading. The following method calls are flagged: 1) Class.forName() 2) ClassLoader.loadClass() 3) ClassLoader.findClass()	category: Backdoor Vulnerabilities Implementations of 'PrivilegedAction' and 'PrivilegedExceptionAction' are used to interact with sensitive data or operations. Only "trusted" classes should be interacted with in privileged code blocks. By using dynamically loaded classes it becomes difficult to know whether the loaded class should be trusted or not.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware.
25.	Use "read-only" Access Mode for Castor queries	This rule identifies calls to the overloaded method "execute()" made by Castor queries (a class implementing "org.exolab.castor.jdo.Query"). A violation is flagged if the "execute ()" method called does not specify an Access Mode or the access mode is not "read-only".	category: Backdoor Vulnerabilities Calling "execute()" with no AccessMode uses "shared" mode by default. Use of a mode other than "read only" allows the query to be written to, which may leave the query open to malicious behavior.	1. Cigital Java Security Rulepack # 18 and # 19: http://www.cigital.com/securitypack/view/index.html 2. Query (Castor JavaDoc) http://castor.codehaus.org/javadoc/org/exolab/castor/jdo/Query.html 3. Access Mode (Castor JavaDoc) http://castor.codehaus.org/javadoc/org/exolab/castor/mapping/AccessMode.html
26.	Wrap "privileged" method invocations in "final" methods	This rule flags a violation if a call to a user-specified "privileged" method is made from a non-"final" method.	category: Backdoor Vulnerabilities "Privileged" method calls should be made only from "final" methods to prevent an attacker from attempting to override or bypass the code.	1. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware 2. Cigital Java Security Rulepack # 64: http://www.cigital.com/securitypack/view/index.html

27.	Wrap "privileged" method invocations in "private" methods	This rule flags a violation if a call to a user-specified "privileged" method is made from a non-"private" method.	category: Backdoor Vulnerabilities "Privileged" method calls should be made only from "private" methods to limit exposure of privileged or sensitive information. Declaring a method "private" may also aid in preventing an attacker from attempting to override or bypass the code.	1. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware: http://www.mikeware.us/thesis/ware-writingsecurejavacode-may08.pdf 2. Cigital Java Security Rulepack # 63: http://www.cigital.com/securitypack/view/index.html
28.	Inspect usage of 'Date', 'Time' objects and 'System. current Time Millis ()' method invocations	This rule identifies variable declarations that are 'java.util.Date', 'java.sql.Date' or 'java.sql.Time' objects, it also searches for method invocations of 'System. get Current Time Millis()'. An error is reported for each occurrence.	category: Backdoor Vulnerabilities category: Malicious Code Date and Time objects can be entry points for security attacks. System times might also indicate areas where malicious code has been placed. By identifying the usage of those elements, you can assess how are they being used, and determine whether access to important functionality is based on 'Date' and 'Time' objects or system times.	N/A
29.	Inspect usage of 'getName()' from 'java.lang.Class' object	This rule identifies code where 'getName()' gets the name of a class. A message is reported for each occurrence.	category: Backdoor Vulnerabilities category: Malicious Code Use of the 'getName()' method of "java.lang.Class" often indicates logic which relies on comparing classes by name. Code logic should not rely on comparing classes by name. If classes are compared by name, it may be possible for an attacker to add a malicious class with the same package name and class name as the expected class. This may allow for the execution of code in the malicious class instead of the execution of code in the expected class.	N/A
30.	Do not use threads in web components	Web components should not use threads. This rule will flag a violation for each instantiation of "java.lang.Thread" (or any class extending "Thread") in a Web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Deadlocks and Race Conditions Using threads in a web component is likely to lead to confusion and unintended consequences, as the web component is already being used in a web server which itself is a threaded environment. Avoid using threads in web components to prevent errors related to concurrency between the web application and the web server.	CWE-383: J2EE Bad Practices: Direct Use of Threads http://cwe.mitre.org/data/definitions/383.html
31.	Do not pass byte arrays to ObjectOutput Stream in the 'writeObject()' method	This rule identifies code that passes byte arrays to Object Output Stream in the 'write Object()' method. An error is reported if byte[] fields are passed to Object Output Stream. write(byte[]) in a custom serialization method (write Object). Fields of type byte[] should be cloned before being passed to the serialization output stream.	category: Erratic Application Behavior category: Data Security The class ObjectOutputStream may be subclassed by a malicious class that tries to modify object-internal data. When a direct reference to the byte[] field is passed to the Object Output Stream, this reference can be used to access and modify the array contents. Passing a clone of the original	Secure Programming for Linux and Unix HOWTO - Chap 10. Language-Specific Issues http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/java.html (rule #14)

			array makes this kind of attack impossible.	
32.	Do not compare Class objects by name	This rule identifies code that compares class objects with the 'getName ()' method. An error is reported for each occurrence.	Enforce code access control. See BENEFITS for more information. .	Statically Scanning Java Code: Finding Security Vulnerabilities. John Viega, Gary McGraw, Tom Mutdosch, and Edward W. Felten IEEE SOFTWARE September/October 2000 http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html CWE-486: Comparison of Classes by Name http://cwe.mitre.org/data/definitions/486.html
33.	Do not use AWT classes in Web components	Web components should not use AWT components. This rule will flag a violation for each method invocation from the "java.awt" package in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Erratic Application Behavior Web components which have AWT code in them are likely to have unanticipated functionality from an architectural point of view, which decreases the quality and security of the application. Additionally, it may represent an entry point for an attacker.	N/A
34.	Enforce returning a defensive copy in 'clone()' methods	This rule identifies classes that implement 'java.lang.Cloneable', but do not initialize the mutable fields of a clone object to defensive copies in the 'clone()' method. An error is reported for each occurrence.	category: Erratic Application Behavior If a mutable field in a Cloneable object is not initialized by a defensive copy, when users change the state of that mutable field on the Cloneable object, the field in the original object will be changed as well.	N/A
35.	Do not stop the JVM in a web component	Web components should not stop the JVM. This rule will flag any call to the method 'System.exit()' in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Erratic Application Behavior Stopping the JVM stops all applications running on the JVM, and could cause a widespread denial of service if exploitable by an attacker.	CWE-382: J2EE Bad Practices: Use of System.exit() http://cwe.mitre.org/data/definitions/382.html
36.	Do not pass user-given mutable objects directly to certain types	This rule checks that mutable objects, which are passed as parameters to methods, are not passed directly to certain constructors. Instead of passing these objects directly, a copy should be made. This way, the objects cannot be unexpectedly changed while in use. A violation is flagged for each occurrence.	category: Erratic Application Behavior Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A4-Insecure Direct Object References". See BENEFITS section. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.5.4: "Insecure direct object references".	1. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware 2. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 3. OWASP Top 10 2013 (A4-Insecure Direct Object References): https://www.owasp.org/index.php/Top_10_2013-

				Top_10
37.	Do not declare "static" fields in web components	Web components should not contain non-final static fields. This rule will flag a violation for each non-final static field in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package. N/A	category: Erratic Application Behavior Because web components are part of a multi-user, threaded environment (provided by the web server), static fields are likely to be modified in unanticipated ways and should not be used (in web components) as sources for data.	N/A
38.	Do not change the input streams of 'java.lang.System' in a web component	Web components should not change the input streams of 'java.Lang.System'. This rule will flag any call to the method 'System.setIn()' in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Erratic Application Behavior Changing the input streams of 'System' is a suspicious security pattern.	N/A
39.	Do not store user-given mutable objects directly into variables	This rule checks that mutable objects which are passed as parameters to methods are not stored directly into variables. Instead of storing these objects directly into variables, a copy should be made. This way, the objects cannot unexpectedly be changed in the calling method. A violation is flagged for each occurrence.	category: Erratic Application Behavior See BENEFITS section.	http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/java.html
40.	Avoid calling specified methods from web components and EJBs	There are certain methods which should not be called from web components (servlets) and EJBs. This parameterizable rule will flag a violation if a method specified in the parameters is called.	category: Erratic Application Behavior Use of certain methods may be considered unsafe in web components or EJBs. Furthermore, certain methods should be avoided as they are considered bad practice or not advised by the Java API. For example, Sockets should be avoided in web applications as they are prone to error and other alternatives exist. Using unexpected or inappropriate methods may result in unpredictable behavior which can lead to loss of security.	N/A
41.	Limit the number of "AccessController.doPrivileged" calls per class	This rule flags a violation if a class contains greater than a user-specified number of "AccessController.doPrivileged" calls.	category: Erratic Application Behavior Following this rule can minimize the amount and locations of security of code which can reduce code complexity and make it less prone to errors.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/250.html
42.	Limit the number of lines in "privileged" code blocks	This rule flags a violation if the "run()" method of a "java.security.PrivilegedAction" or "java.security.PrivilegedExceptionAction" implementation contains greater than the specified number of lines.	category: Erratic Application Behavior Privileged code should be kept minimized to improve clarity and reduce chances for errors.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/250.html
43.	Inspect 'static' fields which may have	This rule identifies "static" fields which may have been intended to be declared	category: Erratic Application Behavior A "static" field which is not also declared "final"	CWE-500: Public Static Field Not Marked Final http://cwe.mitre.org/data/de

	intended to be declared 'static final'	"static final" instead. This rule will check the following types: - primitive types - java.lang.Boolean - java.lang.Byte - java.lang.Character - java.lang.Double - java.lang.Float - java.lang.Integer - java.lang.Long - java.lang.Short - java.lang.String An error is reported for each occurrence.	may be changed by other classes and break the logic of the application.	finitions/500.html
44.	Implement 'readObject()' and 'writeObject()' for all 'Serializable' classes	This rule identifies 'Serializable' classes that do not implement the 'readObject()' and/or 'writeObject()' methods. By default, the rule checks that 'readObject()' is implemented in all classes implementing the 'java.io.Serializable' interface. The rule can be parameterized to also check for 'writeObject()'. An error is reported for each occurrence.	category: Erratic Application Behavior category: Input Validation Even if your class uses the default serialized form, you should still use 'readObject()' and 'writeObject()' and validate all serialized data to guarantee security and class invariants.	1. Joshua Bloch: "Effective Java - Programming Language Guide" Addison Wesley, 2001, pp. 218-219 2. Cigital Java Security Rulepack # 10: http://www.cigital.com/securitypack/view/index.html LOG @move-from PB.OROM(v5.1)
45.	Do not pass exception messages into output in order to prevent the application from leaking sensitive information	This rule identifies code that passes exception messages into output. An error is reported when a catch clause calls an output method and the exception being caught in the catch clause appears in the list of parameters or is used as the calling object.	category: Exposing Sensitive Data category: Error Handling Sensitive information might be leaked when exception messages are passed into output. Hackers trying to gain information about a server application could look at exception messages leaked from the server. The recommended way to obtain exception information is to deploy a logging system (instead of using print methods). Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure". This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.5.7: "Improper error handling".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/209.html http://cwe.mitre.org/data/definitions/311.html http://cwe.mitre.org/data/definitions/497.html
46.	Store sensitive data in mutable objects	This rule flags a violation if sensitive data is not stored in mutable objects. The following cases are flagged: 1) A "javax.swing.JPasswordField" calls "getSelectedText()" or "getText()", both of which return a String 2) A "password" method specified in the parameter table, which returns a char[] is converted to a String through use of either the "String(char[])" or "String(char[],int,int)" constructor	category: Exposing Sensitive Data Sensitive data should be stored as mutable objects so they can be overwritten after use. For example, a password is frequently stored as a "char[]". After use the password could be cleared by using "Arrays.fill()". However, if thepassword were instead stored as a String, then the value may persist in memory as the garbage collector is responsible for removing the String from memory.	
47.	Use 'post' instead of 'get' for	Web forms which contain password fields should use the 'post' method instead of the 'get' method to communicate this information to other	category: Exposing Sensitive Data See BENEFITS section.	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10

	credential transfers	pages. This rule will flag a violation for any 'form' tag which has 'get' for its 'method' attribute and which contains in its body an 'input' tag which has the type 'password'.		Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/352.html http://cwe.mitre.org/data/definitions/598.html
48.	Clear sensitive data after use	This rule flags a violation if sensitive data is not cleared after use. This rule checks "private" fields and local variables which are assigned the return value of "java.swing.JPasswordField#getPassword()" or "java.io.Console#readPassword()". If the return value of those methods are never cleared through use of "java.util.Arrays#fill()" or in a loop then a violation is flagged.	category: Exposing Sensitive Data Sensitive data should be cleared after it is no longer needed to ensure that it spends as little time in memory as possible.	1. Console (Java Platform SE 6): http://java.sun.com/javase/6/docs/api/java/io/Console.html 2. JPasswordField (Java Platform SE 6): http://java.sun.com/javase/6/docs/api/javax/swing/JPasswordField.html 3. Cigital Java Security Rulepack # 57: http://www.cigital.com/securitypack/view/index.html
49.	Avoid storing sensitive data in plaintext in a cookie	This rule finds sensitive data in the javax.servlet. http.Cookie constructor and detects dangerous situations such as 1) when a cookie stores passwords 2) when a cookie stores roles 3) when a cookie stores user identifiers These situations are recognized by cookie name and value. A violation is reported for each usage of a Cookie constructor where such data is passed.	category: Exposing Sensitive Data Enforcing this rule will help to protect against many of the OWASP Top 10 application vulnerabilities, such as: A6-Sensitive Data Exposure See BENEFITS.	1. CWE-315: Plaintext Storage in a Cookie http://cwe.mitre.org/data/definitions/315.html 2. CWE-807: Reliance on Untrusted Inputs in a Security Decision http://cwe.mitre.org/data/definitions/807.html 3. OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 Avoid methods that might expose internal representations
50.	by returning arrays or other mutable fields	This rule identifies methods that might expose internal representations by returning arrays or other mutable objects. An error is reported for each occurrence. If an array field or any other mutable field is declared as "private", "package-private" or "protected", and there is a return statement that returns the reference to the mutable object, then the caller would be able to modify the content of the field regardless of the accessibility modifiers on the field. Enforcing this rule will help to protect against the OWASP Top 10 application vulnerability "A4-Insecure Direct Object Reference".	category: Exposing Sensitive Data See BENEFITS section.	OWASP Top 10 2013 (A4-Insecure Direct Object Reference): https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-495: Private Array-Typed Field Returned From A Public Method http://cwe.mitre.org/data/definitions/495.html
51.	Inspect instance fields of serializable	This rule identifies non-transient, non-final instance fields within Serializable classes. An error is reported for each occurrence.	category: Exposing Sensitive Data Any non-transient, non-final instance field will be serialized. Those fields might carry confidential data which should either not be serialized or	CWE-499: Serializable Class Containing Sensitive Data http://cwe.mitre.org/data/de

	objects to make sure they will not expose sensitive information		should be encrypted before serialization.	finitions/499.html
52.	Do not expose data with a 'FileNotFound' exception	This rule identifies method calls that throw uncaught FileNotFound exceptions. A violation is reported for each occurrence.	If a method calls the java.io.FileInputStream constructor to read an underlying configuration file and that file is not present, a java.io.FileNotFoundException containing the file path is thrown. Propagating this exception back to the method caller exposes the layout of the file system. Many forms of attack require knowing or guessing locations of files.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#2-1
53.	Do not interrogate or modify security policy information in a web component	Web components should not interrogate or modify security policy information in the container. This rule will flag any call to the method 'getPolicy()' or the method 'setPolicy()' from the class 'java.security.Policy' in a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Exposing Sensitive Data Interrogating the security policy can create security exposures, particularly if the results of the interrogation were to be disseminated outside the container: for example, if they were sent to a Web client. This information would enable a malicious entity to better plan an attack against the container and/or enterprise. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10
54.	Declare "transient" fields "private"	This rule identifies "transient" fields which are not declared "private".A violation is flagged for each case.	category: Exposing Sensitive Data Fields marked "transient" frequently contain sensitive data. They should also be marked "private" to help prevent unintended access of the data. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware: http://www.mikeware.us/thesis/ware-writingsecurejavacode-may08.pdf
55.	Avoid "transient" fields in serialPersistentFields array	This rule identifies "transient" fields which are referenced by a "serialPersistentFields" array. This rule flags a violation if the following conditions are met: 1. The class is Serializable 2. "serialPersistentFields" has the proper format of "private static final ObjectStreamField[] serialPersistentFields" 3. The field matching the "name" parameter of any of the ObjectStreamField is "transient"	category: Exposing Sensitive Data Fields marked "transient" frequently contain sensitive data. Any field containing sensitive data should not be referenced in the "serialPersistentFields" array.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware DefiningSerializable Fields for a Class http://docs.oracle.com/javase/1.5.0/docs/guide/serialization/spec/serial-arch.html#6250
56.	Avoid writing to Consoles	This rule identifies calls to the 'flush', 'format', or 'printf' method of a 'java.io.Console'. This rule also identifies calls to an output method by the 'PrintWriter' returned from a call to	category: Exposing Sensitive Data Output to a Console may reveal system data or leftover debugging information which could then be used by an attacker.	CWE - CWE-497: Information Leak of System Data (1.5) http://cwe.mitre.org/data/definitions/497.html

		'java.io.Console#writer()'. A violation is reported for each occurrence.		
57.	Do not log confidential or sensitive information	This rule identifies usage of certain phrases in string literals used in specified logging methods. The literal will be flagged if it is in the definition of a logging method or passed in as a parameter. Custom methods and qualified names can be specified, as can the strings to flag.	Some information, such as Social Security numbers (SSNs) and passwords, are highly sensitive. This information should not be kept for longer than necessary,nor where it may be seen, even by administrators. For instance, it should not be sent to log files and its presence should not be detectable through searches. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 http://www.oracle.com/technetwork/java/seccodeguide-139067.html#2-2 Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/311.html http://cwe.mitre.org/data/definitions/533.html http://cwe.mitre.org/data/definitions/534.html
58.	Avoid calling print methods of 'System.err' or 'System.out'	This rule flags calls to the 'format()', 'print()', 'println()', and 'write()' methods of 'System.err' and System.out'. These methods are commonly used for debugging purposes.	category: Exposing Sensitive Data Debug statements can contain information about implementation details that should not be leaked to the user. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A2-Broken Authentication and Session Management".	OWASP Top 10 2013 (A2-Broken Authentication and Session Management): https://www.owasp.org/index.php/Top_10_2013-Top_10
59.	Encapsulate all redirect and forward URLs with a validation function	This rule identifies if redirect URLs or forward URLs are not encapsulated validation method invocations. An error is reported for each occurrence.	Enforcing this rule will help to protect against some of the OWASP Top 10 2013 application vulnerabilities, including: A10-Unvalidated Redirects and Forwards It concerns also CWE 2010 Top 25 - Insecure Interaction Between Components	OWASP Top 10 2013 (A10-Unvalidated Redirects and Forwards): https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-601: URL Redirection to Untrusted Site ('Open Redirect') http://cwe.mitre.org/data/definitions/601.html
60.	Avoid using "SELECT *" in SQL queries	The "splat" character (the wild-card character "*") should not be used in the "SELECT" portion of an SQL query. The specific fields needed should be specified instead. This rule will flag a violation for each String in which the pattern "SELECT *" appears.	category: Input-Based Attacks Allowing the use of "SELECT *" in code is a bad security practice because it is likely to grant the programmer more access to data from the database than is required. In general, only the data that is absolutely necessary should be accessed, and this data should be filtered based on the user and their role.	N/A
61.	Encapsulate all dangerous data returning methods with a validation function	This rule will flag the following cases: 1. method invocations which return dangerous data which are not encapsulated validation method invocations e.g HttpServletRequest.getParameter(String) 2. tainted data is passed to dangerous methods without having been previously validated e.g DatagramChannel.reveive(ByteBuffer) 3. method invocations which throws	category: Input-Based Attacks category: Code Quality Input validation needs to be performed on all dangerous data in order to prevent security vulnerabilities. Enforcing this rule will help to protect against many of the OWASP Top 10 application vulnerabilities, including: A1-Injection A3-Cross-Site Scripting (XSS)	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml

		RemoteException (if "Remote methods" is checked in the rule parameter) An error is reported for each occurrence. This sources of tainted data include thousands of methods that are too numerous to list here. These methods are taken from various APIs and libraries, including but not limited to: * Java Standard Library * Apache ECS * JDBC * Hibernate * Struts * Spring * XPath Data sources which are considered tainted by default: * Parameters of never called methods * Parameters of remote methods * Data defined by native methods * Data from non-validating Struts forms * Data retrieved from the net Additional data sources which may be considered tainted from rule parameters: * Servlet requests * Files * Pipes * Remote methods * Reflection methods * Environment variables and system properties * Database * Stream-oriented APIs (streams, readers and channels) * Console * GUI controls This rule will not flag the following cases (see EXAMPLE below): - a dangerous method is encapsulated by a validation method - the return value of a dangerous method is assigned to a variable and the variable is then encapsulated by a validation method	A4-Insecure Direct Object Reference A7-Missing Function Level Access Control A8-Cross-Site Request Forgery (CSRF) This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications".Specifically, this rule helps to test for Issue 6.5.1: "Unvalidated input".	Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/79.html http://cwe.mitre.org/data/definitions/352.html
62.	Avoid XPath injection when evaluating XPath queries	This rule will flag the following cases: 1. An XPath query is concatenated with variables (fields, parameters and local variables). 2. 'XPathVariableResolver' is not set on XPath variables (if the parameter is enabled). An error is reported for each occurrence.	category: Input-Based Attacks If an XPath query is concatenated with variables, it may allow attackers to inject the query with data that will lead the application to execute in a waythat programmers did not intend. For a similar rule, see 'BD.SECURITY.TDXPATH'	N/A
63.	Do not extend from the Struts classes 'ActionForm' and 'DynaActionForm'	This rule makes sure that classes do not extend from the class "org.apache.struts.action.ActionForm" or from the class "org.apache.struts.action.DynaActionForm". A violation will be flagged for each class	category: Input-Based Attacks If an ActionForm is not validated, the values passed to the system can cause various types of security errors based on unvalidated inputs such as SQL injection, cross-site scripting, and other injection vulnerabilities.	CWE - CWE-104: Struts: Form Bean Does Not Extend Validation Class (1.5) http://cwe.mitre.org/data/definitions/104.html

		extending either ActionForm or DynaActionForm.		
64.	Avoid temporary files	This rule identifies calls to the 'createTempFile' methods of 'java.io.File'. A violation is reported for each occurrence.	category: Input-Based Attacks Temporary files may be created with loose permissions that could allow an attacker to unexpectedly alter the file. This could disrupt the application or have farther reaching security consequences depending on what the file is used for.	N/A
65.	Canonicalize all data before validation	This rule identifies code that validates data without first canonicalizing the data. A violation will be flagged for each call to one of the user-specified validation methods which passes in a local variable which was never passed to one of the user-specified canonicalization methods. A violation will also be flagged for each call to a validation method which passes in the return value of another method call without first passing this return value to a canonicalization method.	category: Input-Based Attacks Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities, including: A1-Injection A3-Cross-Site Scripting (XSS)	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') http://cwe.mitre.org/data/definitions/79.html
66.	Always call 'super.validate()' from validation methods in 'ActionForm' classes	This rule checks to make sure that overridden validation methods in classes extending from "org.apache.struts.action.ActionForm" call "super.validate()". This enforces usage of the validation mechanism. A violation will be flagged for each "validate" method in a class extending from "ActionForm" which does not call "super.validate()".	category: Input-Based Attacks See BENEFITS section.	N/A
67.	Use wrapper methods to secure native methods	This rule will flag any native method that is called outside of a wrapper class. Native methods do not benefit from the security checks and buffer overflow protections that are standard in java methods. Any native method not declared private will be flagged, along with any private native method that is called more than one time in the file.	Java code is subject to runtime checks for type, array bounds, and library usage. Native code, on the other hand, is generally not. While pure Java code is effectively immune to traditional buffer overflow attacks, native methods are not. To offer some of these protections during the invocation of native code, do not declare a native method public. Instead, declare it private and expose the functionality through a public Java-based wrapper method. A wrapper can safely perform any necessary input validation prior to the invocation of the native method.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#5-3 CWE-111: Direct Use of Unsafe JNI http://cwe.mitre.org/data/definitions/111.html
68.	Use 'prepareCall' or 'prepareStatement' instead of 'createStatement'	This rule will flag any use of the method "createStatement()" from the interface "java.sql.Connection". Instead, "prepareCall" or "prepareStatement()" should be used. A violation is flagged for each occurrence.	category: Input-Based Attacks "java.sql.Statement" is typically the culprit in SQL Injection vulnerabilities in Java, as there is no way to parameterize the query without needing to escape SQL meta-characters. Using objects of type "prepareCall()" or "prepareStatement()" instead of "createStatement()" can help prevent SQL injection vulnerabilities. Enforcing this rule will help to protect against the	OWASP Top 10 2013 (A1-Injection): https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') http://cwe.mitre.org/data/de

			OWASP Top 10 2013 application vulnerability "A1-Injection"	finitions/89.html
69.	Avoid storing usernames and passwords in plain text in Castor 'jdo-conf.xml' files	This rule checks for Castor usernames and passwords which are stored in plain text in 'jdo-conf.xml' files. Castor usernames and passwords should always be encrypted if they are to be stored in XML files. Since there are many different kinds of encryption algorithms in the market, in this rule, we assume that an encrypted password is a combination of uppercase letters, lowercase letters, digits and symbols. A violation will be flagged for each Castor username or password which is stored in a 'jdo-conf.xml' file and which does not contain an uppercase letter, lowercase letter, digit and symbol.	category: Unsafe Environment Configuration If a user stores a plain text username or password in a Castor configuration file, the username or password can easily be stolen by anyone with access to that file. This makes it very easy for hackers to steal the username or password. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers". This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access." Specifically, this rule helps to test for Issue 8.4: "Render all passwords unreadable during transmission and storage on all system components using strong cryptography" and Issue 8.5.11: "Use passwords containing both numeric and alphabetic characters."	1. Cigital Java Security Rule Pack # 17: http://www.cigital.com/securitypack/view/index.html 2. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 3. CWE-798: Use of Hard-coded Credentials http://cwe.mitre.org/data/definitions/798.html
70.	Ensure that passwords are not stored in plain text and are sufficiently long	The rule identifies the following insecure password Strings in the configuration file: - empty password string - password which is too short - plain text password An error is reported for each occurrence. A plain text password is a password which is not encrypted. Since there are many different kinds of encryption algorithms in the market, in this rule, we assume that an encrypted password is a combination of uppercase letters, lowercase letters, digits and symbols.	category: Unsafe Environment Configuration If a user stores a plain text password in a XML configuration file or property file, his password can easily be stolen by someone (including hackers) who can access that file. If the empty String is used as a password, it can be guessed very easily. If the password is fewer than 6 characters(by default) in length, it is very easy to guess even using a brute force approach of trial and error. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers".	1. Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/258.html 2. Cigital Java Security Rulepack # 25: http://www.cigital.com/securitypack/view/index.html 3. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 4. Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/256.html http://cwe.mitre.org/data/definitions/798.html
71.	Ensure WS-Security is enabled in WebSphere 'ibm-webservicesclient-ext.xmi' files	This rule identifies if <securityRequestGeneratorServiceConfig> or <securityResponseConsumerServiceConfig> tags are missing in WebSphere 'ibm-webservicesclient-ext.xmi' files. Omitting these tags will cause the integrity and confidentiality of SOAP messages to not be guaranteed. An error is reported for each occurrence.	category: Unsafe Environment Configuration WS-Security enhances the security of SOAP messages at the message level. If WS-Security is not enabled, message integrity and confidentiality will depend on transport security.	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
72.	Avoid unencrypted passwords in	This rule identifies if the password in a <securityToken> tag is not encrypted in WebSphere 'ibm-webservicesclient-	category: Unsafe Environment Configuration If passwords are not encrypted, they may be stolen by attackers if the message is sent over an	1. http://www.redbooks.ibm.com/redbooks/pdfs/sg24725

	WebSphere 'ibm-webservicesclient-ext.xmi' files	ext.xmi' files. An unencrypted password will be exposed to attackers if the message is sent over an insecure channel. An error is reported for each occurrence.	insecure channel. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers". This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access." Specifically, this rule helps to test for Issue 8.4: "Render all passwords unreadable during transmission and storage on all system components using strong cryptography" and Issue 8.5.11: "Use passwords containing both numeric and alphabetic characters.". Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A6-Sensitive Data Exposure".	7.pdf 2. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 3. OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 4. Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/798.html
73.	Ensure all web content directories have a "welcome file"	This rule identifies if web content directories didn't have a "welcome file" which the "welcome files" are specified in 'web.xml'. An error is reported for each occurrence.	Enforcing this rule will help to protect against some of the OWASP Top 10 2013 application vulnerabilities, including: A5-Security Misconfiguration For example, if a web content directory, '/admin' didn't have a welcome file, hackers will be able to see and download all the files in '/admin' directory by typing url 'http://notsafe.com/admin'	OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10
74.	Ensure WS-Security is enabled in WebSphere 'ibm-webservices-ext.xmi' files	This rule identifies if <securityRequestConsumerServiceConfig> or <securityResponseGeneratorServiceConfig> tags are missing in WebSphere 'ibm-webservices-ext.xmi' files. Omitting these tags will cause the integrity and confidentiality of the SOAP messages to not be guaranteed. An error is reported for each occurrence.	category: Unsafe Environment Configuration WS-Security enhances the security of SOAP messages at the message level. If WS-Security is not enabled, message integrity and confidentiality will rely upon the security of the transport mechanism.	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
75.	Avoid unencrypted passwords in WebSphere 'ibm-webservices-ext.xmi' files	This rule identifies if passwords in <securityToken> tags are not encrypted in WebSphere 'ibm-webservices-ext.xmi' files. Unencrypted passwords will be exposed to attackers if messages are sent over an insecure channel. An error is reported for each occurrence.	category: Unsafe Environment Configuration If passwords are not encrypted, they may be stolen by attackers if the message is sent over an insecure channel. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers". This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access." Specifically, this rule helps to test for Issue 8.4: "Render all passwords unreadable during transmission and storage on all system components using strong	1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf 2. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 3. OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 4. CWE-798: Use of Hard-coded Credentials

			cryptography" and Issue 8.5.11: "Use passwords containing both numeric and alphabetic characters." Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A6-Sensitive Data Exposure".	http://cwe.mitre.org/data/definitions/798.html
76.	Avoid defining multiple security roles with the same name in 'web.xml' files	A common mistake in 'web.xml' files is to define multiple security roles with the same name. This rule will flag a violation for any case where there are multiple '<security-role>' elements within a 'web.xml' file which all specify the same role name.	category: Unsafe Environment Configuration Including multiple '<security-role>' elements with the same name can lead to unauthorized access to resources. The last security role with a given role name will most likely be chosen if there are multiple security roles with the same role name. This can cause unintended access to be granted. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability, including: A5-Security Misconfiguration A7-Missing Function Level Access Control	Cigital Java Security Rulepack # 31: http://www.cigital.com/securitypack/view/index.html OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-863: Incorrect Authorization http://cwe.mitre.org/data/definitions/863.html
77.	Ensure SOAP messages are encrypted in WebSphere 'ibm-webservicesclient-ext.xmi' files	This rule identifies if <requiredConfidentiality> and <confidentiality> tags are not configured correctly or are missing in WebSphere 'ibm-webservicesclient-ext.xmi' files. Omitting these tags will cause the SOAP messages to be unencrypted. An error is reported for each occurrence.	category: Unsafe Environment Configuration If SOAP messages are encrypted at the message level, the messages will be confidential, which means that you do not need to worry about the transport security. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability, including: A5-Security Misconfiguration A6-Sensitive Data Exposure	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10
78.	Ensure SOAP messages are digitally signed in WebSphere 'ibm-webservicesclient-ext.xmi' files	This rule identifies if <requiredIntegrity> and <integrity> tags are not configured correctly or are missing in WebSphere 'ibm-webservicesclient-ext.xmi' files. Omitting these tags will cause the integrity of SOAP message to not be guaranteed. An error is reported for each occurrence.	category: Unsafe Environment Configuration If the SOAP message is digitally signed, the receiver can use the signature to verify the integrity of the message in order to detect whether the message was modified by a third party or not. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10
79.	Ensure SOAP messages are encrypted in WebSphere 'ibm-webservices-ext.xmi' files	This rule identifies if <requiredConfidentiality> or <confidentiality> tags are missing or not configured properly in WebSphere 'ibm-webservices-ext.xmi' files. Omitting these tags will cause SOAP messages to be unencrypted. An error is reported for each occurrence.	category: Unsafe Environment Configuration If SOAP messages are encrypted at the message level, this ensures that the messages will be confidential. This means that you do not need to worry about the transport security. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability, including: A5-Security Misconfiguration A6-Sensitive Data Exposure	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10
80.	Ensure SOAP	This rule identifies if <requiredIntegrity> or <integrity> tags are missing or not configured properly in WebSphere 'ibm-webservices-ext.xmi' files. Omitting	category: Unsafe Environment Configuration If SOAP messages are digitally signed, the receiver can use the signature to verify the integrity of the message. That way, the receiver can detect	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf

	messages are digitally signed in WebSphere 'ibm-webservices-ext.xmi' files	these tags will cause the integrity of SOAP message to not be guaranteed. An error is reported for each occurrence.	whether the message was modified by a third party or not. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10
81.	Ensure that 'axis.development.system' is set to "false" in Axis 'server-config.wsdd' files	This rule identifies cases where the parameter 'axis.development.system' is set to "true" in an Apache Axis 'server-config.wsdd' file. An error is reported for each occurrence.	category: Unsafe Environment Configuration 'axis.development.system' should only be set to "true" during the development phase. Having this property set to "true" in production code will cause stack traces and other debugging information to be exposed to the user. This information is likely to include sensitive information and implementation details that should not be exposed to the user.	"Apache Axis Reference Guide", "Global Axis Configuration" section: http://ws.apache.org/axis/java/reference.html#GlobalAxisConfiguration
82.	Ensure that 'axis.enableListQuery' is set to "false" in Axis 'server-config.wsdd' files	This rule identifies if the parameter 'axis.enableListQuery' is set to "true" in an Apache Axis 'server-config.wsdd' file. An error is reported for each occurrence.	category: Unsafe Environment Configuration If 'axis.enableListQuery' is set to "true" in a 'server-config.wsdd' file, it will cause the current system config to be listed. This can expose sensitive information such as passwords.	"Apache Axis Reference Guide", "Global Axis Configuration" section: http://ws.apache.org/axis/java/reference.html#GlobalAxisConfiguration
83.	Ensure that 'axis.disableServiceList' is set to "true" in Axis 'server-config.wsdd' files	This rule identifies cases where the parameter 'axis.disableServiceList' is set to "false" in an Apache Axis 'server-config.wsdd' file. The 'axis.disableServiceList' parameter is set to "false" by default, so a violation will also be flagged if this parameter is not specified. An error is reported for each 'server-config.wsdd' file where 'axis.disableServiceList' is either not specified or explicitly set to "false".	category: Unsafe Environment Configuration 'axis.disableServiceList' should not be set to "false" because doing so will allow the available services to be listed when a "GET" request is performed on the servlet root. This can allow access to a list of features which should not be exposed to the public.	"Apache Axis Reference Guide", "Global Axis Configuration" section: http://ws.apache.org/axis/java/reference.html#GlobalAxisConfiguration
84.	Ensure that the 'Encrypt' directive is specified for each 'items' tag in Axis2 configuration files	This rule checks that the actions defined within the "InflowSecurity" and "OutflowSecurity" parameters of Apache Axis2 'services.xml' and 'axis2.xml' files specify the "Encrypt" directive. For each "action" defined within an "InflowSecurity" or "OutflowSecurity" parameter, the rule will check that the '<items>' tag specifies the "Encrypt" directive so that messages will be properly encrypted. The rule will flag a violation for each '<items>' tag which is missing the "Encrypt" directive. 'services.xml' and 'axis2.xml' files which do not use the Rampart security module (i.e., those which do not contain a '<module ref="rampart" />' tag) will be ignored.	category: Unsafe Environment Configuration For "items" tags contained within "InflowSecurity" and "OutflowSecurity" parameters, it is recommended that the "Encrypt" directive always be specified. Without this directive, messages will not have any encryption, and message security will rely on the security of the transport mechanism. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration	"Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10
85.	Ensure that each filter mapped in a	Each filter mapping in a 'web.xml' file should include a '<filter-name>' element with the name of a filter defined within	category: Unsafe Environment Configuration	Cigital Java Security Rulepack # 34: http://www.cigital.com/sec

	'web.xml' file has a corresponding definition	that 'web.xml' file. A common mistake is for a filter mapping to specify the name of a filter that is missing a definition. This rule will flag a violation for each '<filter-mapping>' element in a 'web.xml' file which contains a '<filter-name>' element which specifies the name of a filter for which there is no corresponding '<filter>' element in the 'web.xml' file.	Filters are often used to prevent common attacks such as URL injection and HTTP request injection. If you attempt to use a filter to prevent attacks but fail to define the filter correctly, the filter will not be effective in preventing attacks. To ensure security, you must make sure that there is a corresponding '<filter>' element for each '<filter-mapping>' element.	uritypack/view/index.html
86.	Ensure that 'InflowSecurity' and 'OutflowSecurity' parameters are specified in Axis2 configuration files	This rule checks that an "InflowSecurity" and "OutflowSecurity" parameter is specified for each <service> or <axisconfig> tag defined in an Apache Axis2 'services.xml' file or 'axis2.xml' file. By default, a violation will be flagged for each <service> or <axisconfig> tag defined within an Apache Axis2 'services.xml' or 'axis2.xml' file which does not contain either an "InflowSecurity" or an "OutflowSecurity" child tag. 'services.xml' and 'axis2.xml' files which do not use the Rampart security module (those which do not contain a '<module ref="rampart" />' tag) will be ignored.	category: Unsafe Environment Configuration If the "InflowSecurity" or "OutflowSecurity" parameter is missing, messages are not guaranteed to be secure. Their security will depend on the security of the mechanism used to transport the messages. You should always specify these parameters for each <service> and <axisconfig> tag to make sure that the messages are secure. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration".	"Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10
87.	Include an appropriate '<login-config>' element to specify the type of authentication to be performed in 'web.xml' files	When using 'auth-constraint' elements to control access to resources in a 'web.xml' file, you must also include a 'login-config' element to specify the type of user authentication to be performed. The four recognized types of user authentication are: - BASIC - FORM - DIGEST - CLIENT-CERT This rule will flag a violation for each 'web.xml' file which includes at least one 'auth-constraint' element but which either does not contain a 'login-config' element or contains a 'login-config' element which specifies an authentication method which is not one of the four recognized authentication methods.	category: Unsafe Environment Configuration If you omit the 'login-config' element from a 'web.xml' file which is supposed to perform access control or if you include an incorrect 'login-config' element, it may be possible for unauthorized users to access sensitive resources. To ensure that proper access control is performed, you should always include a 'login-config' element which specifies one of the four recognized types of user authentication listed in the "DESCRIPTION" section. Enforcing this rule will help to protect against some of the the OWASP 2013 Top 10 application vulnerability including: A2-Broken Authentication and Session Management A5-Security Misconfiguration	Cigital Java Security Rulepack # 33: http://www.cigital.com/securitypack/view/index.html OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/862.html
88.	Avoid using plain text passwords in Axis 'wsdd' files	This rule checks that plain text passwords are not used in Apache Axis '.wsdd' files. A violation will be flagged for each "passwordType" parameter in a '.wsdd' file which has the value "PasswordText" unless encryption is performed. Using the password type "PasswordText" without any encryption is dangerous because it may allow passwords to be transmitted in plain text. By default, "passwordType" parameters with value "PasswordText" will not be flagged if there is also an "action" parameter which specifies to use encryption on the password. By default,	category: Unsafe Environment Configuration Passwords of type "PasswordText" will be sent in plain text unless they are encrypted. This is dangerous from a security standpoint because the passwords can be easily stolen if they are sent over an insecure channel. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers".	1. "WSS4J": http://ws.apache.org/wss4j/package.html 2. CWE-522: Insufficiently Protected Credentials http://cwe.mitre.org/data/definitions/522.html 3. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 4. CWE-798: Use of Hard-

		a violation will also not be flagged if there is no "action" parameter which specifies to use the "UsernameToken" action.	This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access." Specifically, this rule helps to test for Issue 8.4: "Render all passwords unreadable during transmission and storage on all system components using strong cryptography" and Issue 8.5.11: "Use passwords containing both numeric and alphabetic characters."	coded Credentials http://cwe.mitre.org/data/definitions/798.html
89.	Restrict access to JSPs in 'web.xml' files by including a security constraint for '*.jsp' files	JSP files often contain Javascript and Java code which should not be exposed to all users. Thus, it is a good idea to always include a security constraint in all 'web.xml' files which restricts access to '*.jsp' files. This rule will flag a violation for each 'web.xml' file which does not contain a '<security-constraint>' element restricting access to '*.jsp' resources. It will also flag a violation for each 'web.xml' file which contains a '<security-constraint>' element which allows access to '*.jsp' resources to all users by specifying '*' for the role name.	category: Unsafe Environment Configuration JSP files should not be accessible to all users because they often contain business logic implemented in Javascript or Java code which should have restricted access. The best way to restrict access to JSP files is to define security roles and to limit access to '*.jsp' resources using a '<security-constraint>' element.	Cigital Java Security Rulepack # 35: http://www.cigital.com/securitypack/view/index.html
90.	Ensure that the 'Signature' directive is specified for each 'items' tag in Axis2 configuration files	This rule checks that the actions defined within the "InflowSecurity" and "OutflowSecurity" parameters of Apache Axis2 'services.xml' and 'axis2.xml' files specify the "Signature" directive. For each "action" defined within an "InflowSecurity" or "OutflowSecurity" parameter, the rule will check that the "items" tag specifies the "Signature" directive so that messages will contain a signature which allows the receiver to verify the authenticity of the message. The rule will flag a violation for each '<items>' tag which is missing the "Signature" directive. 'services.xml' and 'axis2.xml' files which do not use the Rampart security module (i.e., those which do not contain a '<module ref="rampart" />' tag) will be ignored.	category: Unsafe Environment Configuration For '<items>' tags contained within "InflowSecurity" and "OutflowSecurity" parameters, it is recommended that the "Signature" directive always be specified. Specifying the "Signature" directive will ensure that messages contain a signature which allows the receiver to verify the integrity of the message. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration".	"Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10 Parasoft Jtest Static Analysis Rules > Security [SECURITY] > Unsafe Environment Configuration [SECURITY.UEC]
91.	Always specify error pages in web.xml	This rule identifies 'web.xml' files which do not specify any 'error-page' elements. A violation is reported for each occurrence. This rule also has parameters which can be used to check the format of 'error-page' elements as well as if certain error codes or exception types are covered.	category: Unsafe Environment Configuration Failure to specify an 'error-page' for common situations may result in a default error page being displayed. In the case of exceptions, data may be displayed to the user which reveals inner workings of the application. An attacker could then use this information to manipulate or exploit the application.	N/A
92.	Session identifiers should be at least 128 bits long to prevent brute-force session	Session identifiers should be at least 128 bits long to prevent brute-force attacks. This rule check deployment descriptor files and identifies uses of Session-ID with a width less than that set in the parameters.	category: Unsafe Environment Configuration	1. CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length http://cwe.mitre.org/data/definitions/6.html

	guessing			2. Sun Java System Application Server Platform Edition 8 Developer's Guide. Chapter 5 Deployment Descriptor Files http://download.oracle.com/docs/cd/E19518-01/817-6087/dgdesc.html 3. OWASP Brute force attack https://www.owasp.org/index.php/Brute_force_attack 4. ASDR TOC Vulnerabilities OWASP Insufficient Session-ID Length https://www.owasp.org/index.php/Insufficient_Session-ID_Length
93.	Avoid using the SOAP Monitor module	Apache Axis 2 comes with a utility called the SOAP Monitor module. For security reasons, you should only enable this module for debugging purposes. This rule identifies cases where the SOAP Monitor module is enabled in an 'axis2.xml' file.	category: Unsafe Environment Configuration When the SOAP Monitor module is enabled, it is easy for attackers to eavesdrop on SOAP messages sent and received by a Web application. To protect sensitive information, this utility should always be disabled.	"Apache Axis Reference Guide": http://ws.apache.org/axis2/1_4_1/soapmonitor-module.html
94.	Ensure that each security role referenced in a 'web.xml' file has a corresponding definition	Each security role referenced within an '<auth-constraint>' element of a 'web.xml' file must have a corresponding '<security-role>' element with the same role name. The '<security-role>' element is the element which defines the security role. If this element is missing, any resource which is specified to only allow access to that security role will be inaccessible to users who are supposed to be in that security role. A violation will be flagged for each '<auth-constraint>' element which specifies a role name for which there is no corresponding '<security-role>' element.	category: Unsafe Environment Configuration Failure to properly define security roles will lead to denial of access to anyone who is supposed to be in the specified role. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration".	Cigital Java Security Rulepack # 30: http://www.cigital.com/securitypack/view/index.html OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-863: Incorrect Authorization http://cwe.mitre.org/data/definitions/863.html
95.	Ensure that sessions are configured to time out within a reasonable amount of time in 'web.xml' files	You should be careful to configure a reasonable session timeout within each 'web.xml' file. Configuring a reasonable session timeout will ensure that users can comfortably browse the Web application without leaving the session open too long in case they forget to log out. The standard way to configure a session timeout in a 'web.xml' file is with a '<session-timeout>' element like in the following example: <web-app>	category: Unsafe Environment Configuration A2-Broken Authentication and Session Management If a session is left open for too long, it may allow an unauthorized user to start using the session. People often forget to log out of Web applications, so it is best to protect them from malicious users by automatically logging them out after a certain amount of time. This is especially true for Web applications which are used at public terminals. If you do not set an explicit session timeout in the	Cigital Java Security Rulepack # 29: http://www.cigital.com/securitypack/view/index.html OWASP Top 10 for 2013 (A2-Broken Authentication and Session Management): https://www.owasp.org/index.php/Top_10_2013-Top_10 CWE-613: Insufficient

		... <session-config> <session-timeout>30</session-timeout> </session-config> </web-app>	'web.xml' file, it will be left up to the container to determine the length of the timeout. You should specify an explicit timeout to prevent the container from enforcing a timeout which is too long or no timeout at all.	Session Expiration http://cwe.mitre.org/data/definitions/613.html
96.	Ensure that the 'Timestamp' directive is specified for each 'items' tag in Axis2 configuration files	This rule checks that the actions defined within the "InflowSecurity" and "OutflowSecurity" parameters of Apache Axis2 'services.xml' and 'axis2.xml' files specify the "Timestamp" directive. For each "action" defined within an "InflowSecurity" or "OutflowSecurity" parameter, the rule will check that the '<items>' tag specifies the "Timestamp" directive so that messages will contain a timestamp which will help to prevent Replay attacks. The rule will flag a violation for each "items" tag which is missing the "Timestamp" directive. 'services.xml' and 'axis2.xml' files which do not use the Rampart security module (i.e., those which do not contain a '<module ref="rampart" />' tag) will be ignored.	category: Unsafe Environment Configuration For '<items>' tags contained within "InflowSecurity" and "OutflowSecurity" parameters, it is recommended that the "Timestamp" directive always be specified. Specifying the "Timestamp" directive will ensure that messages contain a timestamp. This will help to prevent replay attacks. A replay attack happens when an attacker intercepts a message and then sends it at a later time when the message is a security risk. If the message contains a timestamp, the receiver will be able to determine that the message is stale and reject it. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration".	"Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html OWASP Top 10 2013 (A5-Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10
97.	Ensure that all constrained resources are protected with a '<user-data-constraint>' element in 'web.xml' files	'<security-constraint>' elements in 'web.xml' files can contain a '<user-data-constraint>' element. While the '<user-data-constraint>' element is optional, it is recommended that you always include it for constrained resources because the '<user-data-constraint>' element allows you to specify how data sent between the client and the container will be protected. According to the Java EE 5 Tutorial (See the "REFERENCES" section), the following are legal values for the '<transport-guarantee>' element which must be defined within the '<user-data-constraint>' element: - "CONFIDENTIAL" when the application requires that data be transmitted so as to prevent other entities from observing the contents of the transmission - "INTEGRAL" when the application requires that the data be sent between client and server in such a way that it cannot be changed in transit - "NONE" to indicate that the container must accept the constrained requests on any connection, including an unprotected one You should always include a '<user-data-constraint>' element with a '<transport-guarantee>' element which specifies the appropriate level of security. This rule will flag a violation for each '<security-constraint>' element in a 'web.xml' file which contains a non-empty '<auth-constraint>' element and	category: Unsafe Environment Configuration Resources which contain an '<auth-constraint>' element are generally supposed to be protected. This means that communication between the client and the container should also be protected for these resources. The way to do this is to include a '<user-data-constraint>' element with either "INTEGRAL" or "CONFIDENTIAL" specified for the transport guarantee. Even in cases where the communication does not need to be protected, you should specify "NONE" for the transport guarantee to make this explicit.	1. The Java EE 5 Tutorial: http://java.sun.com/javaee/5/docs/tutorial/doc/bnbxw.html 2. Cigital Java Security Rulepack # 32: http://www.cigital.com/securitypack/view/index.html

		which does not contain a '<user-data-constraint>' element.		
98.	Avoid using plain text passwords in Axis2 configuration files	This rule checks that plain text passwords are not used in Apache Axis2 'services.xml' and 'axis2.xml' files. For each '<action>' defined within an "InflowSecurity" or "OutflowSecurity" parameter, the rule will check that the password type is not set to "PasswordText". A violation will be flagged for each '<action>' tag which contains a '<passwordType>PasswordText</passwordType>' child tag. By default, a violation will not be flagged if the '<items>' tag also specifies to use "Encrypt" to encrypt the plain text password. By default,a violation will also not be flagged if the '<items>' tag does not specify to use the "UsernameToken" directive. 'services.xml' and 'axis2.xml' files which do not use the Rampart security module (i.e., those which do not contain a '<module ref="rampart" />' tag) will be ignored.	category: Unsafe Environment Configuration Using the "UsernameToken" directive with a plain text password and no encryption is dangerous because the password may be stolen by attackers if the message is sent over an insecure channel. Passwords should always be encrypted before being sent over an insecure channel. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers". This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access." Specifically, this rule helps to test for Issue 8.4: "Render all passwords unreadable during transmission and storage on all system components using strong cryptography" and Issue 8.5.11: "Use passwords containing both numeric and alphabetic characters."	1. "Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html 2. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 3. Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/798.html
99.	Avoid misconfiguring timestamps in WebSphere 'ibm-webservicesclient-ext.xmi' files	This rule identifies if the <addTimestamp> tag is missing in WebSphere 'ibm-webservicesclient-ext.xmi' files or the expiration is missing in the <addTimestamp> tag. Omitting the <addTimestamp> tag or the expiration will cause the SOAP message to be vulnerable to replay attacks. An error is reported for each occurrence.	category: Unsafe Environment Configuration If the timestamp is missing or the timestamp never expires in the SOAP message, it may be vulnerable to replay attacks. A replay attack happens when an attacker intercepts a message and then sends it at a later time (when the message is harmful).	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
100.	Avoid unsigned timestamps in WebSphere 'ibm-webservicesclient-ext.xmi' files	This rule identifies if there is an <addTimestamp> tag in WebSphere 'ibm-webservicesclient-ext.xmi' files but there is no digital signature for the timestamp. Any unsigned timestamp is vulnerable to replay attacks. An error is reported for each occurrence.	category: Unsafe Environment Configuration If the timestamp is not signed, an attacker can intercept the SOAP message, modify the timestamp and send a malicious message to the receiver.	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
101.	Avoid misconfiguring timestamps in WebSphere 'ibm-webservices-ext.xmi' files	This rule identifies if an <addTimestamp> tag is missing in WebSphere 'ibm-webservices-ext.xmi' files or an expiration is missing in the <addTimestamp> tag. Omitting the <addTimestamp> tag or the expiration will cause the SOAP messages to be vulnerable to replay attacks. An error is reported for each occurrence.	category: Unsafe Environment Configuration If the timestamp is missing or the timestamp never expires in the SOAP message, the application may be vulnerable to replay attacks. A replay attack happens when an attacker intercepts a message and then sends it at a later time (when the message is harmful).	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
102.	Ensure that the Rampart WS-Security module is enabled in Axis2	Apache Axis2 comes with a security module called "Rampart" which provides WS-Security features. This module is enabled by adding a '<module ref="rampart"/>' tag to the 'services.xml' or 'axis2.xml' file. This rule will flag a violation for each 'services.xml' or	category: Unsafe Environment Configuration The Apache Rampart WS-Security module is provided with Apache Axis2 to ensure message integrity and confidentiality. If Rampart is disabled, it makes it difficult to guarantee message integrity and confidentiality.	"Securing SOAP Messages with Rampart": http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html OWASP Top 10 2013 (A5-

	configuration files	'axis2.xml' file which is missing the tag '<module ref="rampart"/>'. Without this tag, Rampart will be disabled.	Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A5-Security Misconfiguration".	Security Misconfiguration): https://www.owasp.org/index.php/Top_10_2013-Top_10
103.	Avoid unsigned timestamps in WebSphere 'ibm-webservices-ext.xmi' files	This rule identifies if there is an <addTimestamp> tag in a WebSphere 'ibm-webservices-ext.xmi' file but there is no digital signature for that timestamp. Any unsigned timestamp is vulnerable to replay attacks. An error is reported for each occurrence.	category: Unsafe Environment Configuration If a timestamp is not signed, an attacker can intercept the SOAP message, modify the timestamp, and send the message to the receiver at a later time (when the message is harmful). Since the timestamp has been altered, the receiver has no way of telling that the message is stale.	http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf
104.	Use 'https' instead of 'http' for the 'transportReceiver' and 'transportSender' in 'axis2.xml' configuration files	This rule identifies cases where the transport sender or transport receiver specified in the 'axis2.xml' configuration file uses "http" instead of "https". A violation will be flagged for each "transportSender" or "transportReceiver" tag where the name attribute is set to "http".	category: Unsafe Environment Configuration If sensitive data is being transmitted, you should use "https" (or some other secure protocol) instead of "http" since "https" uses SSL to ensure that data is secure. The "http" protocol does not provide any encryption. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #4: "Encrypt transmission of cardholder data across open, public networks". Specifically, this rule helps to test for Issue 4.1: "Use strong cryptography and security protocols such as SSL/TLS or IPSEC to safeguard sensitive cardholder data during transmission over open, public networks." Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	1. "HTTP Transport": http://ws.apache.org/axis2/1_4_1/http-transport.html 2. "CWE-311: Failure to Encrypt Sensitive Data": http://cwe.mitre.org/data/definitions/311.html 3. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 4. OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10
105.	Ensure that "REST" is disabled in 'axis2.xml' configuration files	This rule checks that "REST" is disabled in Apache Axis 2 'axis2.xml' configuration files. A violation will be flagged for each 'axis2.xml' file which sets the "disableREST" parameter to "false".	category: Unsafe Environment Configuration "REST" does not have any message security standards, so it is dangerous to use "REST". Some people prefer to always have "REST" disabled.	N/A
106.	Ensure all exceptions are either logged with a standard logger or rethrown	This rule identifies code that does not log caught exceptions with a standard logger or rethrow caught exceptions. If the exception could be handled without using a standard logger or rethrowing the exception, then a comment should be added.	category: Unsafe Error Handling and Logging Using a logging mechanism to track exceptions caught could provide a clearer and more secure overview of the possible security vulnerabilities. A prompt and accurate fix could be made based on the information. Enforcing this rule will help to protect against the OWASP 2007 Top 10 application vulnerability "A6 - Information Leakage and Improper Error Handling". This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to check for Issue 6.5.7:"Improper error handling".	OWASP Top 10 2007 (A6 - Information Leakage and Improper Error Handling): http://www.owasp.org/index.php/Top_10_2007-A6 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-390: Detection of Error Condition Without Action http://cwe.mitre.org/data/definitions/390.html
107.	Do not	This rule identifies custom 'ClassLoader'	category: Weak Security Controls	

	define custom class loaders	declarations. A message is reported for each occurrence.	category: Malicious Code Creating a custom class loader that is not secure can allow the loading of malicious classes.	http://www.unix.org.ua/orelly/java-ent/security/ch03_04.htm
108.	Do not pass mutable objects to 'ObjectOutputStream' in the 'writeObject()' method	This rule identifies code that directly passes mutable fields to ObjectOutputStream in the 'writeObject()' method. An error is reported if these fields are passed to 'ObjectOutputStream.writeObject(Object)' in a custom serialization method (writeObject). Certain types of fields should be cloned before being passed to the serialization output stream.	category: Weak Security Controls category: Data Security The class ObjectOutputStream may be subclassed by a malicious class that tries to modify object-internal data. When a direct reference to a mutable field is passed to the ObjectOutputStream, this reference can be used to access and modify the field's content. Passing a copy of the field to the 'writeObject(Object)' method makes this kind of attack impossible.	N/A
109.	Ensure arguments passed to certain methods come from predefined methods list	There are certain methods whose arguments should always come from a valid source--otherwise, passing such arguments creates a security risk. For instance, the seed used for generating random values must be different each time and should not be predictable. If the seed can be guessed or known, then the pseudo-random numbers can be determined. This rule will flag a violation for any case where the method specified in the parameter table is not called with an argument from a valid source (also specified in parameter table).	category: Weak Security Controls Passing constant or predictable values to certain methods creates a security risk. Use values from valid/verified sources only.	CWE-329: Not Using a Random IV with CBC Mode http://cwe.mitre.org/data/definitions/329.html CWE-336: Same Seed in PRNG http://cwe.mitre.org/data/definitions/336.html CWE-337: Predictable Seed in PRNG http://cwe.mitre.org/data/definitions/337.html
110.	Avoid passing hardcoded usernames/passwords/URLs to database connection methods	Passing constant values into a database connection method is a security risk. These constant values can include the username, password, and URL for accessing the database. Instead of using constant Strings to connect to a database, the programmer should get database connection Strings from a secure source. By default, this rule will check the following things: 1. this rule will look for any calls to the "getConnection()" method of "java.sql.DriverManager" where all the arguments are constant Strings. 2. this rule will check for the password is hard-coded or not when callin the "getConnection()" method of "java.sql.DriverManager". 3. this rule will check indirect calls to "getConnection().	category: Weak Security Controls category: Data Security If constant database connection Strings are found in a method, they probably were accidentally left in the code. They should be taken out of all production code because leaving these Strings in the code can grant unauthorized users access to data. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers". This rule also helps to enforce the PCI DSS Requirement #8: "Assign a unique ID to each person with computer access". Specifically, this rule helps to test for Issue 8.5.8: "Do not use group, shared, or generic accounts and passwords".	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-259: Use of Hard-coded Password http://cwe.mitre.org/data/definitions/259.html CWE-798: Use of Hard-coded Credentials http://cwe.mitre.org/data/definitions/798.html

111.	Avoid using insecure algorithms for cryptography	Some methods in the Java SDK and the Java Cryptographic Extension library require the specification of an algorithm to be used for data encryption. Examples of such methods which take the name of an algorithm as a parameter are: - the 'getInstance()' methods of "java.security.MessageDigest" - the 'getInstance()' methods of "javax.crypto.Cipher" - the 'getInstance()' methods of "javax.crypto.KeyGenerator" There are a wide variety of algorithms which can be passed into these methods, but some algorithms are insecure and should not be used. To ensure that none of these insecure algorithms are used, you should select algorithms for each of these methods from an approved list of algorithms. See the "PARAMETERS" section for more details about which algorithms should be used with each type. This rule will flag a violation for each case where an algorithm which is not in the approved list of algorithms is passed to the 'getInstance()' method of a type. The types which are checked and the appropriate algorithms to use with each type are specified in the parameter lists. The rule also checks that padding is used with the "RSA" algorithm when calling the 'getInstance()' method of 'javax.crypto.Cipher'. Cigital (See link # 6 in the "REFERENCES" section) recommends that you always use an OAEP (Optimal Asymmetric Encryption Padding) padding mode with the "RSA" algorithm. Without padding, the algorithm is considered insecure. A violation will be flagged for each case where the 'getInstance()' method of 'javax.crypto.Cipher' is called with "RSA" as an argument (with no padding). See the example shown below for more details.	`category: Weak Security Controls category: Cryptography Use of an insecure cryptographic algorithm may result in data being easy to decrypt. You should use a strong algorithm to ensure that sensitive data cannot be easily decrypted. Selecting an algorithm from a list of algorithms known to be secure will help to achieve this goal. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #3: "Protect stored cardholder data". Specifically, this rule helps to check for the requirement to use "strong cryptography with associated key management processes and procedures". Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability A6-Sensitive Data Exposure	1. the OWASP "Guide to Cryptography": http://www.owasp.org/index.php/Guide_to_Cryptography 2. the OWASP page "Using the Java Cryptographic Extensions": http://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions 3. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml 4. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware 5. "JavaTM Cryptography Architecture API Specification & Reference": http://docs.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.html 6. Cigital Java Security Rule Pack # 12 & # 13: http://www.cigital.com/securitypack/view/index.html 7. Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/327.html http://cwe.mitre.org/data/definitions/328.html http://cwe.mitre.org/data/definitions/780.html
112.	Make all member classes and interfaces "private"	This rule identifies member classes and interfaces that are not "private". An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Enforce code access control. See BENEFITS for more information.	Statically Scanning Java Code: Finding Security Vulnerabilities. John Viega, Gary McGraw, Tom Mutdosch, and Edward W. Felten IEEE SOFTWARE September/October 2000 Nigel Warren, Philip Bishop: "Java in Practice - Design Styles and Idioms for Effective Java". Addison-Wesley, 1999. pp.10 - 11.

				The Extension Mechanism (Optional Package Sealing) http://docs.oracle.com/javase/1.5.0/docs/guide/extensions/spec.html#sealing Twelve Rules for developing more secure Java code http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html LOG @deprecated OOP.AIC @severity-from 3(v6.0)
113.	Call authenticati on methods to enforce consistency	This rule identifies method declarations which do not contain required authentication method invocations. An error is reported for each declaration matching one of the user-specified regular expressions which does not contain a call to a centralized authentication method (Centralized authentication methods are specified in the parameters).	category: Weak Security Controls category: Code Quality Authentication should be consistently applied in user management methods. Enforcing this rule will help to protect against many of the OWASP Top 10 application vulnerabilities, such as: A4-Insecure Direct Object Reference A7-Missing Function Level Access Control This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications" and Requirement #8: "Assign a unique ID to each person with computer access". Specifically, this rule helps to test for Issue 6.5.3: "Broken authentication and session management" and Issue 8.5: "Ensure proper user authentication and password management for non-consumer users and administrators on all system components as follows:".	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/306.html http://cwe.mitre.org/data/definitions/352.html LOG @severity-from 4(v8.0)
114.	Call access control methods to enforce consistency	This rule identifies method declarations which do not contain required access control method invocations. An error is reported for each declaration matching one of the user-specified regular expressions which does not contain a call to a centralized access control method (Centralized access control methods are specified in the parameters).	category: Weak Security Controls category: Code Quality Access control should be consistently applied in user management methods. Enforcing this rule will help to protect against many of the OWASP Top 10 application vulnerabilities, such as: A4-Insecure Direct Object Reference A5-Security Misconfiguration A7-Missing Function Level Access Control This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications" and Requirement #8: "Assign a unique ID to each person with computer access". Specifically, this rule helps to test for Issue 6.5.2: "Broken access control" and Issue 8.5.16: "Authenticate all access to any database containing cardholder data. This includes access by applications, administrators,	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/352.html http://cwe.mitre.org/data/definitions/732.html LOG @severity-from 4(v8.0)

			and all other users".	
115.	Do not allow password fields to be autocompleted	The "autocomplete" attribute for password fields in web pages should be set to "false" or "off". This rule will flag a violation for any input tag where the type attribute is "password" and the "autocomplete" attribute is either not specified or set to something other than "false" or "off".	category: Weak Security Controls Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities, including: A2-Broken Authentication and Session Management A8-Cross-Site Request Forgery (CSRF) This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #8: "Assign a unique ID to each person with computer access". Specifically, this rule helps to test for Issue 8.5.8: "Do not use group, shared, or generic accounts and passwords". See the BENEFITS section for more of the SECURITY RELEVANCE.	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/306.html http://cwe.mitre.org/data/definitions/352.html
116.	Enforce 'SecurityManager' checks before setting or getting fields	This rule checks for the following cases: 1. 'SecurityManager' is not checked before setting a field in 'setter' methods. 2. 'SecurityManager' is not checked before getting a field in 'getter' methods. 3. 'SecurityManager' is not checked in non-"final" and non-"private methods. An error is reported for each occurrence.	category: Weak Security Controls 'SecurityManager.checkXXX()' is used to check for permission before performing an unsafe or sensitive operation. Therefore, it should be called before performing any potentially sensitive operations. Also, if a non-"private" method contains sensitive operations, it should be declared as "final" in order to avoid security holes as the method may be overridden by hackers.	N/A
117.	Enforce 'SecurityManager' checks in methods of 'Cloneable' classes	This rule flags the following cases in Cloneable classes: 1. 'SecurityManager' is checked in constructors, but not in the 'clone()' method. 2. 'SecurityManager' is checked in the 'clone()' method, but not in the non-"private" constructors. An error is reported for each occurrence.	category: Weak Security Controls 'SecurityManager.checkXXX()' is used to check for permission before performing an unsafe or sensitive operation. If a constructor of a Cloneable class has a SecurityManager check, it means that the class contains sensitive data and needs to check permission before class creation. Therefore, similar SecurityManager checks should be made in 'clone()' methods as this is another way to create an instance of a class.	N/A
118.	Enforce 'SecurityManager' checks in methods of 'Serializable' classes	This rule checks for the following cases in Serializable classes: 1. 'SecurityManager' is checked in constructors, but not in the 'readObject()' and 'readObjectNoData()' methods. 2. 'SecurityManager' is checked in setter methods, but not in the 'readObject()' method. 3. 'SecurityManager' is checked in getter methods, but not in the 'writeObject()' method. 4. 'SecurityManager' is checked in the 'readObject()' or 'readObjectNoData()' methods, but not in the non-"private" constructors.	category: Weak Security Controls 'SecurityManager.checkXXX()' is used to check for permission before performing an unsafe or sensitive operation. If a constructor, setter, or getter method of a Serializable class has a Security Manager check, it means that certain methods are performing potentially sensitive operations and need to check for permission. Therefore, similar Security Manager checks should be made in methods when doing serialization.	N/A

		An error is reported for each occurrence.		
119.	Use 'java.security.SecureRandom' instead of 'java.util.Random' or 'Math.random()'	This rule identifies declarations of the 'java.util.Random' object or method invocations of 'Math.random()'. An error is reported for each occurrence.	category: Weak Security Controls category: Cryptography Enforcing this rule will help to protect against the OWASP Top 10 2013 application vulnerability "A6-Sensitive Data Exposure". Avoid weak random number generators. See BENEFITS for more information.	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 http://java.sun.com/j2se/1.5.0/docs/api/java/security/SecureRandom.html Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/327.html http://cwe.mitre.org/data/definitions/330.html http://cwe.mitre.org/data/definitions/338.html http://cwe.mitre.org/data/definitions/676.html
120.	Use the SSL-enabled version of classes when possible	SSL connections should be used when possible. This rule will flag variable declarations where the type of the variable is one of the non-SSL-enabled types in the list below. non-SSL-enabled type equivalent SSL-enabled type ---------- ---------- javax.net.ServerSocketFactory javax.net.ssl.SSLServerSocketFactory javax.net.SocketFactory javax.net.ssl.SSLSocketFactory java.net.HttpURLConnection javax.net.ssl.HttpsURLConnection java.net.ServerSocket javax.net.ssl.SSLServerSocket java.net.Socket javax.net.ssl.SSLSocket java.rmi.server.RMIClientSocketFactory javax.rmi.ssl.SslRMIClientSocketFactory java.rmi.server.RMIServerSocketFactory javax.rmi.ssl.SslRMIServerSocketFactory java.security.Permission javax.net.ssl.SSLPermission	category: Weak Security Controls Regular network connections can be sniffed by anyone on the network, which could lead to information disclosure. SSL connections are encrypted, and thus are more secure in regards to casual inspection. Enforcing this rule will help to protect against the OWASP Top 10 application vulnerability "A6-Sensitive Data Exposure". This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #4: "Encrypt transmission of cardholder data across open, public networks" and Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to check for Issue 4.1: "Use strong cryptography and security protocols such as SSL/TLS or IPSEC to safeguard sensitive cardholder data during transmission over open, public networks" and Issue 6.5.7: "Improper error handling".	OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/306.html http://cwe.mitre.org/data/definitions/311.html http://cwe.mitre.org/data/definitions/494.html
121.	Avoid hard-coding the arguments to certain methods	There are certain methods for which the arguments should never be hard-coded because hard-coding these arguments is a security risk. For instance, any method which takes a username or password as an argument should require the user to enter his username or password. These methods should not accept hard-coded constant values because using these values will allow anyone access to sensitive data. This rule will flag a violation for any case where the methods specified in the parameter table are passed hard-coded constant values.	category: Weak Security Controls Passing constant values to certain methods represents a security risk. For instance, for the method 'hash()' of the interface "org.owasp.esapi.Encryptor" which is included in the parameter list by default, passing a hard-coded constant value as the second argument (the "salt") is a security risk because the salt is used to encrypt data and should not be exposed to users. Passing in a constant value for the salt will allow all the developers who work on the code to see the salt. Even worse, anyone with access to the	PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml

			byte code can use "javap -c" to find out the value of the salt. This method is just one example of a method which should not accept hard-coded constant values. You should customize the parameter table to list all the methods in your application which should not be passed hard-coded constant values. This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.6: "Removal of custom application accounts, user IDs, and passwords before applications become active or are released to customers".	
122.	Avoid constructors and overriding methods which are more accessible than those of their super classes	This rule identifies constructors and overriding methods which are more accessible than the constructors and overridden methods in their super classes. An error is reported for each occurrence.	category: Weak Security Controls If constructors and overriding methods are more accessible than the constructors and overridden methods in their super classes, attackers could override the methods and gain unintended access to the super class methods.	N/A
123.	Inspect usage of standard API calls that bypass security	This rule identifies method calls that have the potential to bypass SecurityManager checks. Certain standard APIs in the core libraries of the Java runtime enforce SecurityManager checks, but allow those checks to be bypassed depending on the immediate caller's class loader. Other calls use the immediate caller's class loader to find and load the specified library, and calling these methods using untrusted objects is dangerous. It can be dangerous to allow untrusted code to have access to any return value that comes from one of these methods. These methods can be used safely, but the objects that call them, the objects that are used as parameters, and the objects that they return need to be evaluated and prevented from interacting with untrusted code.	The Java SecurityManager allows applications to implement security policies. Bypassing the securityManager is an attack vector that should be prevented, and can be prevented if these methods are evaluated carefully or avoided completely. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A9-Using Components with Known Vulnerabilities".	OWASP Top 10 2013 (A9-Using Components with Known Vulnerabilities): https://www.owasp.org/index.php/Top_10_2013-Top_10 http://www.oracle.com/technetwork/java/seccodeguide-139067.html#4-5 http://www.oracle.com/technetwork/java/seccodeguide-139067.html#4-6 CWE-545: Use of Dynamic Class Loading http://cwe.mitre.org/data/definitions/545.html
124.	Avoid turning raw text into xml	This rule identifies possible instances of XML or HTML creation from raw text. Any time a string literal is concatenated, assigned to a string, or appended to a string, StringBuffer, or StringBuilder, this rule will flag. It is recommended that XML or HTML creation libraries be utilized, so that security checks may be done.	Running improperly formed code can affect security.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#3-1 CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) http://cwe.mitre.org/data/definitions/80.html

125.	Do not use inner classes	This rule identifies classes that contain inner classes. An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Enforce code access control. See BENEFITS for more information	http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html
126.	Allow only certain providers to be specified for the 'Security.addProvider()' method	This rule enforces that only certain providers are passed to the 'addProvider()' method of 'java.security.Security'. By default, the only acceptable providers are the following: - sun.security.pkcs11.SunPKCS11 - com.sun.net.ssl.internal.ssl.Provider - com.sun.security.sasl.Provider - com.sun.rsajca.Provider - sun.security.provider.Sun - com.sun.crypto.provider.SunJCE - sun.security.jgss.SunProvider - sun.security.rsa.SunRsaSign It is recommended that only the cryptography providers which are provided by Sun be used.	category: Weak Security Controls Category: Cryptography If a custom cryptography provider is used, it may implement the cryptography incorrectly, leaving the application vulnerable. Only trusted cryptography providers (like the Sun providers listed above) should be used.	N/A
127.	Keep all access control methods centralized to enforce consistency	This rule identifies access control methods which are called outside of their centralized locations. An error is reported for each occurrence. Instead of calling access control methods directly, a wrapper method should be used which calls the method that does the actual access control. The methods which do the actual access control are specified in the "Access control method names" parameter. The wrapper methods which are allowed to call these methods directly are specified in the "Centralized access control method names" parameter. If a method not specified in this parameter calls one of the methods specified in the "Access control method names" parameter, an error is flagged.	category: Weak Security Controls category: Access Control Access control should be centralized so that it can be validated as consistent. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability, including: A5-Security Misconfiguration A7-Missing Function Level Access Control This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.5.2: "Broken access control" .	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml
128.	Keep all authentication methods centralized to enforce consistency	This rule identifies authentication methods which are called outside of their centralized location. An error is reported for each occurrence. Instead of calling authentication methods directly, a wrapper method should be used which calls the method that does the actual authentication. The methods which do the actual authentication are specified in the "Authentication method names" parameter. The wrapper methods which are allowed to call these methods directly are specified in the "Centralized authentication method names" parameter. If a method not specified in this parameter calls one of the methods specified in the "Authentication method names" parameter, an error is flagged.	category: Weak Security Controls category: Authentication Authentication should be centralized so that it can be validated as consistent. Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A7-Missing Function Level Access Control". This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications" and Requirement #8: "Assign a unique ID to each person with computer access". Specifically, this rule helps to test for Issue 6.5.3: "Broken authentication and session management" and Issue 8.5: "Ensure	OWASP Top 10 2013 (A7-Missing Function Level Access Control): https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-306: Missing Authentication for Critical Function http://cwe.mitre.org/data/definitions/306.html

			proper user authentication and password management for non-consumer users and administrators on all system components as follows:".	
129.	Always clone array parameters which are stored to fields	This rule flags a violation if an array parameter is not cloned before being stored in a field.	category: Weak Security Controls If an array is not cloned before being stored in a field then any external changes to the array contents will be reflected in the stored array and may affect functionality.	CWE-496: Public Data Assigned to Private Array-Typed Field http://cwe.mitre.org/data/definitions/496.html PMD Rulesets index: ArrayIsStoredDirectly http://pmd.sourceforge.net/rules/index.html
130.	Only call "final" methods from specified code blocks in non-"final" classes	This rule flags a violation if a non-"final" method is called from a specified method or "synchronized" statement of a non-"final" class. This rule can be used to perform security checks of code blocks which run privileged or sensitive code. See PARAMETERS.	category: Weak Security Controls Non-"final" classes which perform privileged or sensitive actions should avoid calling non-"final" methods from certain code blocks. Only "trusted" methods should be called from privileged code and by allowing only calls to "final" methods there are extra protections that the method being called is the expected one.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
131.	Only "clone()" instances of "final" classes	This rule flags a violation if a call to "clone()" is made by an instance of a non-"final" class.	category: Weak Security Controls If a class is not declared "final" it may be possible to construct a malicious subclass. For security purposes, calls to "clone()" should be assumed that they may operate on malicious data. Therefore, to reduce the likelihood of this occurring classes which may be cloned should be declared "final".	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
132.	Avoid using cryptographic keys which are too short	In order to be secure, cryptographic keys should be sufficiently long. This rule will check for code which uses one of the cryptography methods provided by Java but specifies a key length which is too short. The recommended minimum key length differs for each algorithm. By default, the rule will flag a violation for each case where the "AES" algorithm is used with a key shorter than 128 bits or the "RSA" algorithm is used with a key shorter than 1024 bits. In order for this rule to find a violation, the code must use either the 'init()' method of "javax.crypto.KeyGenerator" or the 'initialize()' method of "java.security.KeyPairGenerator" to set the size of the cryptographic key. The code must also use the 'getInstance()' method of either "javax.crypto.KeyGenerator" or "java.security.KeyPairGenerator" to	category: Weak Security Controls According to the NIST (the National Institute of Standards and Technology), using a key size of at least 128 bits with the "AES" algorithm will guarantee that the data remains secure until 2030. Using a large key makes the data more difficult to decrypt. For the "RSA" algorithm, experts recommend using a key that is at least 1024 bits long to ensure that data is secure. This rule helps to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure".	1. "Recommendation for Key Management - Part 1: General (Revised)" by NIST: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf 2. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware 3. Cigital Java Security Rulepack # 11 and # 59: http://www.cigital.com/securitypack/view/index.html 4. OWASP Top 10 2013 (A6-Sensitive Data

		specify which algorithm is being used.		Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10
133.	Inspect instantiations of 'ClassLoader' objects	This rule identifies instantiations of 'ClassLoader' object. A message is reported for each occurrence.	category: Weak Security Controls category: Malicious Code This rule prevents insecure class loaders. See BENEFITS for more information. Ensure that the class loader is secure	N/A
134.	Do not override any 'ClassLoader' method except 'findClass()'	This rule identifies any overridden methods of 'java.lang.ClassLoader' except 'findClass'. An error is reported for each occurrence.	category: Weak Security Controls category: Malicious Code Prevent insecure class loader. See BENEFITS for more information.	ClassLoader (Java 2 Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/
135.	Make your 'clone()' method "final" for security	This rule identifies classes that implement the Cloneable "interface", but do not have a "final" 'clone()' method. An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Enforce code access control. Seee BENEFITS for more information.	Statically Scanning Java Code: Finding Security Vulnerabilities. John Viega, Gary McGraw, Tom Mutdosch, and Edward W. Felten IEEE SOFTWARE September/October 2000 Joshua Bloch : "Effective Java - Programming Language Guide" Addison Wesley, 2001, pp. 45-52 http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html CWE-491: Public cloneable() Method Without Final ('Object Hijack') http://cwe.mitre.org/data/definitions/491.html
136.	Do not define custom 'SecurityManager's	This rule identifies code that tries to define a custom security manager that extends 'java.lang.SecurityManager'. A message is reported for each occurrence.	category: Weak Security Controls category: Code Quality An insecure custom security manager can grant unauthorized classes access to privileged methods and cause access control problems.	Cigital Java Security Rulepack # 61 and # 62: http://www.cigital.com/securitypack/view/index.html
137.	Avoid using hard-coded cryptographic keys	This rule identifies hard-coded cryptographic keys. A violation is reported for each occurrence.	category: Weak Security Controls If a hard-coded cryptographic key is used the application becomes more vulnerable to brute force attacks. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities including "A6-Sensitive Data Exposure".	CWE - CWE-321: Use of Hard-coded Cryptographic key (1.5) http://cwe.mitre.org/data/definitions/321.html OWASP Top 10 2013 (A6-Sensitive Data Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10

138.	Declare subclasses of 'PrivilegedAction', 'PrivilegedExceptionAction', and 'PrivilegedActionException' "final"	This rule flags a violation if a class is non-"final" and satisfies one of the following: 1. The class extends "java.security.PrivilegedActionException" 2. The class implements "java.security.PrivilegedAction" 3. The class implements "java.security.PrivilegedExceptionAction"	category: Weak Security Controls "Privileged" classes and interfaces are used to interact with sensitive data and operations. A subclass or implementation should be declared "final" to prevent an attacker from creating a subclass and gaining access to sensitive information.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
139.	Declare subclasses of 'Permission' and 'BasicPermission' "final"	This rule flags a violation if a class is non-"final" and satisfies one of the following: 1. The class extends "java.security.BasicPermission" 2. The class extends "java.security.Permission"	category: Weak Security Controls "Permission" classes are used to interact with sensitive data and operations. A subclass should be declared "final" to prevent an attacker from creating a subclass and gaining access to sensitive information.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
140.	Ensure that all Permissions, PrivilegedActions, and PrivilegedActionExceptions are declared in the same package	This rule enforces that all 'java.security.Permission', 'java.security.PrivilegedAction', 'java.security.PrivilegedExceptionAction', and 'java.security.PrivilegedActionException' classes are grouped together in the same package. An error is reported for each occurrence.	category: Weak Security Controls 'java.security.Permission' and 'java.security.PrivilegedAction' are used to check for permissions and perform dangerous operations. It is a good practice to separate these security-critical modules from other modules.	N/A
141.	Declare the 'run()' method of 'PrivilegedAction' and 'PrivilegedExceptionAction' implementations "final"	This rule flags a violation if the 'run()' method of a "java.security.PrivilegedAction" or "java.security.PrivilegedExceptionAction" implementation is not declared "final".	category: Weak Security Controls Implementations of 'PrivilegedAction' and 'PrivilegedExceptionAction' are used to to interact with sensitive data or operations. If the 'run()' method is not declared "final" an attacker could subclass the implementation and override the 'run()' method. This may allow the privileged code to be bypassed or otherwise interfered with.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
142.	Do not declare fields as "public" "static" "final" 'Collection' or 'Map' objects	This rule identifies "public" "static" "final" fields that implement the "java.util.Collection" or "java.util.Map" interfaces. An error is reported for each occurrence.	category: Weak Security Controls See BENEFITS section.	Collections (Java 2 Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collections.html
143.	Ensure 'SecurityManager' check in constructor of "public" non-"final" sensitive	This rule checks "public" non-"final" sensitive classes which do not perform a SecurityManager "check" method in their constructor. A class is considered sensitive if it disallows cloning by overriding the "clone()" method to throw an exception.	category: Weak Security Controls If a class prevents cloning, it usually implies that there are intended restrictions on when new instances can be created. Adding a SecurityManager check in a constructor can add additional security in limiting when a subclass is allowed.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware http://www.oracle.com/technetwork/java/seccodeguide-139067.html#7-2

	type			
144.	Classloaders should only be created inside doPrivileged block	This code creates a classloader, which needs permission if a security manage is installed. If this code might be invoked by code that does not have security permissions, then the classloader creation needs to occur inside a doPrivileged block.	category: Weak Security Controls See BENEFITS section.	Collections (Java 2 Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collections.html
145.	Do not use sockets in web components	Web components should avoid usinge sockets. This rule will flag a violation for each instantiation of "java.net.Socket", "java.net.ServerSocket", or any of their subclasses from a web component. A class or interface is considered a web component if it extends or implements a type from the "javax.servlet" package.	category: Weak Security Controls See BENEFITS section.	Collections (Java 2 Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collections.html
146.	Ensure that an appropriate security manager is set	This rule identifies code where a security manager is not set through 'setSecurityManager' within 'main'. An error is reported for each occurrence.	category: Weak Security Controls category: Security Inspection A missing security manager might result in unauthorized access to privileged code. Different applications have different security requirements and concerns; consequently, appropriate security managers should be assigned. For example, standalone applications should have weaker permissions than applications that are on a server. Alternatively, the security manager can be set through the command line. Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities, including: A2 - Broken Authentication and Session Management	OWASP Top 10 2013 (A2 - Broken Authentication and Session Management): https://www.owasp.org/index.php/Top_10_2013-Top_10 Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/180.html http://cwe.mitre.org/data/definitions/306.html
147.	Do not call 'System.setProperty()' in a web component	Web components (a class or interface that extends or implements a type from the "javax.servlet" package) should not call 'System.setProperty()'. A violation is flagged for each occurrence.	category: Weak Security Controls Calling setProperty in a web component ties the implementation of the component to the system it is running on, which is generally bad for portability. Additionally, it could allow an attack to set system properties, which has serious security ramifications.	N/A
148.	Use the "getSecure()" and "setSecure()" methods to enforce the use of secure cookies	When using variables of the type 'javax.servlet.http.Cookie', it is important to make sure that all of the 'Cookie' variables are secure. To do this, you should use the 'setSecure()' and 'getSecure()' methods of 'Cookie'. This rule will flag a violation for each declaration of a local 'Cookie' object where neither 'getSecure()' nor 'setSecure()' is called on the object.	category: Weak Security Controls category: Session Management If a 'Cookie' variable is not secure, there is no guarantee that it will be sent using a secure protocol (such as HTTPS or SSL). Enforcing this rule will help to protect against some of the OWASP Top 10 application vulnerabilities, including: A2-Broken Authentication and Session Management A8-Cross-Site Request Forgery (CSRF)	OWASP Top 10 2013: https://www.owasp.org/index.php/Top_10_2013-Top_10 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml Common Weakness Enumeration: http://cwe.mitre.org/data/definitions/614.html

			This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.5.3: "Broken authentication and session management".	http://cwe.mitre.org/data/definitions/306.html http://cwe.mitre.org/data/definitions/352.html http://cwe.mitre.org/data/definitions/807.html
149.	Use wrapper methods instead of calling dangerous or problematic methods directly (custom rule)	This rule checks that the methods specified in the parameters as "dangerous" are not called directly by the user but are instead called using a wrapper method. With the default setting of the parameters, this rule will check nothing. It must be customized. Once methods are added to the parameter list, a violation will be flagged for each case where one of the "dangerous/problematic" methods is called outside of the corresponding wrapper method.	category: Weak Security Controls Enforcing this rule will help to protect against the OWASP 2013 Top 10 application vulnerability "A9-Using Components with Known Vulnerabilities". See BENEFITS section.	OWASP Top 10 2013 (A9-Using Components with Known Vulnerabilities): https://www.owasp.org/index.php/Top_10_2013-Top_10
150.	Always verify JarFile signatures	This rule identifies calls to 'java.util.jar.JarFile' constructors which do not have verification enabled.	category: Weak Security Controls Verification should be enabled for JarFile objects when possible so that if a signature is present then the JarFile is verified against it. This reduces the chances of using a JarFile that has been tampered with or that may contain tainted data.	CWE - CWE-347: Improper Verification of CryptographicSignature (1.5) http://cwe.mitre.org/data/definitions/347.html JarFile (Java Platform SE 6) http://java.sun.com/javase/6/docs/api/java/util/jar/JarFile.html
151.	Inspect usage of scripting API	This rule identifies calls to ObjectInputStream.defaultReadObject. It recommends that they be replaced with calls to ObjectInputStream.readFields.	defaultReadObject can assign arbitrary objects to non-transient fields and does not necessarily return. readFields does not have these problems.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#8-3
152.	Avoid using anonymous "privileged" classes when invoking "AccessController.doPrivileged()"	This rule flags a violation if a call to "AccessController.doPrivileged()" is made using an anonymous subclass of either "PrivilegedAction" or "PrivilegedExceptionAction".	category: Weak Security Controls Use of anonymous classes often increases complexity and lowers readability. Furthermore, security related code should be isolated when possible. Creating an external subclass and using it instead of an anonymous one allows for the code to be kept with other security related code. If a security policy or implementation is then changed, the code can be more easily updated in one location and there are less chances for "old" code to be left behind.	"Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware
153.	Avoid DNS lookups for decision making	This rule checks if the return value of the 'java.net.InetAddress' method 'getHostName()' or 'getCanonicalHostName()' is checked against a String for validation. A violation is reported for each occurrence. See NOTES and SECURITY	category: Weak Security Controls Host names are vulnerable to DNS cache poisoning. Therefore, decisions should not be based on host names. Although IP addresses may be spoofed, it is generally safer to make decisions based on an IP address rather than host name.	CWE - CWE-247: Reliance on DNS Lookups in a Security Decision (1.5) http://cwe.mitre.org/data/definitions/247.html

		RELEVANCE for more information.	When possible, multiple forms of verification should be used.	
154.	Always call 'HttpSession.invalidate()' before 'LoginContext.login()'	This rule identifies calls to 'LoginContext.login()' which are not preceded by a call to 'HttpSession.invalidate()'. A violation is reported for each occurrence.	category: Weak Security Controls If a session is not invalidated before a new login is performed it may use the same session as the previous logged in user. On public computers it may be possible to record session information and then gain access to a user's account if they log into the same application. However, if a session is invalidated before a login then this will not be possible.	CWE - CWE-384: Session Fixation (1.5) http://cwe.mitre.org/data/definitions/384.html
155.	Avoid non-random "byte[]" when using IvParameterSpec	This rule checks that IvParameterSpec is initialized using a random byte array. A byte array is considered to be random if it was passed to SecureRandom#nextBytes().	category: Weak Security Controls Use of a non-random initialization vector may result in unintended disclosure of information which could be used by an attacker. Use of "java.util.Random" is discouraged in favor of "java.security.SecureRandom" as SecureRandom is considered cryptographically strong, while Random is not.	1. Cigital Java Security Rulepack # 14: http://www.cigital.com/securitypack/view/index.html 2. IvParameterSpec (Java 2 Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/javax/crypto/spec/IvParameterSpec.html 3. SecureRandom (Java 2. Platform SE 5.0) http://java.sun.com/j2se/1.5.0/docs/api/java/security/SecureRandom.html 4. Initialization vector - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Initialization_vector 5. CWE-329: Not Using a Random IV with CBC Mode http://cwe.mitre.org/data/definitions/329.html
156.	Avoid string literals except in constant declarations and calls to System.out or System.err's 'print' or 'println' methods	This rule identifies the use of string literals outside of string constant declaration statements and calls to "System.err.print", "System.err.println", "System.out.print", and "System.out.println". An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Hard-coded constant strings in the code are error-prone and hard to maintain.	N/A
157.	Ensure 'SecurityManager' checks before 'Socket' transfers or retrievals	This rule flags a violation if a call to "Socket.getInputStream()" or "Socket.getOutputStream()" is not preceded by a call to a "SecurityManager" "check" method. An error is reported for each occurrence.	category: Weak Security Controls "SecurityManager" is used to check for permissions before performing unsafe or sensitive operations. To improve security, transfers and retrievals from sockets should be preceded by a SecurityManager check.	1. "Writing Secure Java Code: A Taxonomy of Heuristics and an Evaluation of Static Analysis Tools" by Michael Ware 2. OWASP Top 10 2013 (A6-Sensitive Data

			This rule helps to protect against the OWASP 2013 Top 10 application vulnerability "A6-Sensitive Data Exposure". This rule also helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule helps to test for Issue 6.3.1.4: "Validation of secure communications".	Exposure): https://www.owasp.org/index.php/Top_10_2013-Top_10 3. PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml
158.	Do not call the 'printStackTrace()' method of "Throwable" objects	This identifies code that calls the no-argument 'printStackTrace()' method of "Throwable" objects. An error is reported for each occurrence.	category: Exposing Sensitive Data category: Error Handling The application's internal information might be revealed to the user in the error message. Avoid using 'printStackTrace()' in production code unless the output is redirected to a log file. Enforcing this rule will help to protect against the OWASP 2007 Top 10 application vulnerability "A6 - Information Leakage and Improper Error Handling". This rule helps to enforce the PCI DSS (Payment Card Industry Data Security Standard) Requirement #6: "Develop and maintain secure systems and applications". Specifically, this rule tests for Issue 6.5.7: "Improper error handling".	OWASP Top 10 2007 (A6 - Information Leakage and Improper Error Handling): http://www.owasp.org/index.php/Top_10_2007-A6 PCI Data Security Standard: https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml CWE-209: Information Exposure Through an Error Message http://cwe.mitre.org/data/definitions/209.html
159.	Field isn't final but should be	This static field public but not final, and could be changed by malicious code or by accident from another package. The field could be made final to avoid this vulnerability.		http://www.oracle.com/technetwork/java/seccodeguide-139067.html#3-8
160.	Inspect usage of scripting API	This rule identifies instantiations of the scripting API engine, because improper usage of this object can result in execution of untrusted code.	Running an untrusted script can result in unexpected results, including security breaches.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#3-8
161.	Make your classes nondeserializeable	This rule identifies classes that do not have a "final" readObject() method. An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Enforce code access control. See BENEFITS for more information.	http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html
162.	Make immutable classes final	This rule identifies immutable classes that are not declared final. It can also identify fields of immutable classes that are not declared final.	If a class that was intended to be immutable could be changed, the security of any code using that class could be compromised.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#6-1

163.	May expose internal representation by incorporating reference to mutable object	**May expose internal representation by incorporating reference to mutable object**		http://www.oracle.com/technetwork/java/seccodeguide-139067.html#1-1 http://www.oracle.com/technetwork/java/seccodeguide-139067.html#3-8
164.	Ensure that Secure Processing is used	This rule identifies instantiations of certain xml text builders that do not explicitly set the XMLConstants.FEATURE_SECURE_PROCESSING property to true. The specified classes can be changed in the parameters.	See benefits section.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#1-1 http://www.oracle.com/technetwork/java/seccodeguide-139067.html#3-8
165.	Defend against partially initialized instances of non-final classes	This rule identifies mutable classes that do not have initializer flags at the end of their constructors. It can also identify initializer flags that are not volatile.	If an instance was only partially initialized, it should not be able to be used or operated on. Errors can be created which could expose secure data.	http://www.oracle.com/technetwork/java/seccodeguide-139067.html#7-3
166.	Make your classes noncloneable	This rule identifies classes without a "final" clone () method. An error is reported for each occurrence.	category: Weak Security Controls category: Code Quality Enforce code access control. See BENEFITS for more information.	http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html
167.	Inspect 'Random' objects or 'Math.random()' methods that could indicate areas where malicious code has been placed	This rule identifies 'Random' objects or 'Math.random()' methods. A message is reported for each occurrence.	category: Weak Security Controls category: Malicious Code Random number generators might indicate areas where malicious code has been placed. Inspect the code around random number generators for security issues related to unusual behavior.	CWE-511: Logic/Time Bomb http://cwe.mitre.org/data/definitions/511.html
168.	Make your classes nonserializeable	This rule identifies classes that do not have a "final" 'writeObject()' method. An error is reported for each occurrence.	category: Weak Security Controls category: Data Security Avoid serialization of confidential data. See BENEFITS for more information.	Statically Scanning Java Code: Finding Security Vulnerabilities. John Viega, Gary McGraw, Tom Mutdosch, and Edward W. Felten IEEE SOFTWARE September/October 2000 Joshua Bloch: "Effective Java - Programming Language Guide". Addison-Wesley, 2001, pp. 213-217 http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules_p.html CWE-499: Serializable Class Containing Sensitive Data http://cwe.mitre.org/data/de

				finitions/499.html
169.	Use library methods for string replacements of special characters in HTML and XML	This rule identifies calls to the replace method in the string class that use certain strings as parameters. Relevant parameters are listed in the parameters section.	Running improperly formed code can affect security	http://www.oracle.com/technetwork/java/seccodeguide-139067.html
170.	Malicious code vulnerability Warnings	- All programming errors, but also errors in system design or specification, which cannot be classified in another security category are called logic errors. - Thus, these errors are not typical programming errors. - Moreover, it is usually not possible to test for resulting security flaws.	FLO	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
171.	Avoid 'main()' methods because they may allow unauthorized access to classes	This rule identifies each main method declared. An error is reported for each occurrence.	category: Weak Security Controls If a main method is accidentally left in the code, it might allow unauthorized access to a class. See BENEFITS for more information.	N/A
172.	Code Injection (COD)	- The injection of system and script commands into a web application or an application's server. - This kind of attack mostly applies to server side script languages like PHP or Perl.	INP PAT RES	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
173.	Cookie Security (COO)	- This category includes several security vulnerabilities based on cookies, e.g., unfiltered cookie content, cookie poisoning, and flow injection via cookies. - In a broader sense, this section is related to session management.	INP	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
174.	Cross Site Scripting (XSS)	- Here, the attacker inserts code into a URL or link. - The malicious URL must be executed by a web application's user to have an effect. - Misleading users to execute such URLs is supported by the URL itself which looks like a trustworthy URL to the application.	INP	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.

		- This only works when the application is vulnerable to XSS. - The result can be, e.g., the execution of malicious script (e.g., JavaScript) commands on the client side.		
175.	Directory Browsing	Path Traversal		Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
176.	Directory Traversal	Path Traversal		Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0. .
177.	Flow Injection (FLO)	- It is a special case of logic errors and is usually not detectable by security scanners. - This vulnerability is based on setting application states which depend on untrustworthy user data. - Thus, the control flow of an application's code could be influenced by an attacker.	LOG	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
178.	Information Disclosure (INF)	- An information disclosure security flaw can be defined as the emission of data or information which is not intended to become available to the public. - This can be internal or private data. - There are several issues in this category which are not only programming errors, like the wrong or public storage of sensitive data.	Information Disclosure (INF)	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
179.	Contain Password **(INP)**	- Usually any input/external data – not only from users – of an application has to be checked to see whether it conforms to intended formats or properties. - Such procedures usually also involve data filtering (sanitization) and adequate output encoding. If input validation, filtering, and output encoding are missing or incomplete, this can enable a variety of attacks.	COD COO RES SQL XSS	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
180.	Logic Errors (LOG)	All programming errors, but also errors in system design or specification, which cannot be classified in another security category are called logic errors.	FLO	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.

		Thus, these errors are not typical programming errors. Moreover, it is usually not possible to test for resulting security flaws.		
181.	Path Browsing	see Path Traversal		Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
182.	Path Traversal (PAT)	Can be generally defined as unintended access to application files or directories by injecting (sub) paths and filenames. The injection, for instance, can take place into application URLs.	COD INP RES	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
183.	Category Vulnerability	Definition for the area of IT security	Related to (attacks	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
184.	Resource Injection (RES)	Resource injection flaws can be defined as a category of security vulnerability related to unintentional access to system resources via the application layer, like in the case of path traversal.	COD INP PAT	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
185.	SQL Code Injection (SQL)	Results of successful attacks of this category are the execution of arbitrary SQL statements and commands on the application's database backend(s).	INP	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
186.	Unreleased Resources (UNR)	Some program resources, which are, e.g., variables and class instances (objects), have to be explicitly unloaded for freeing application memory. If they are not released properly and not caught by the Java garbage collector, they might lead to increased memory consumption. Thus, in a broader sense, unreleased resources can enable Denial of Service| attacks and are a concern for an application's security.	Unreleased Resources (UNR)	Secologic, Java Web Application Security, Best Practice Guide, Document Version 2.0.
187.	SQL injection	input containing SQL commands to a database server for execution.		Security Code Guidelines, Sun Microsystems, Inc. http://java.sun.com/security/seccodeguide.html
188.	Cross-site scripting	exploit applications that output unchecked input, this tricks the user to execute malicious scripts.		Security Code Guidelines, Sun Microsystems, Inc. http://java.sun.com/security/seccodeguide.html
189.	HTTP response splitting	- exploit applications that output input verbatim to perform Web page defacements or Web cache poisoning attacks.		Security Code Guidelines, Sun Microsystems, Inc. http://java.sun.com/security/seccodeguide.html

190.	Path traversal	exploit unchecked user input to control which files are accessed on the server.		Security Code Guidelines, Sun Microsystems, Inc. http://java.sun.com/security/seccodeguide.html
191.	Command injection	exploit user input to execute shell commands.		Security Code Guidelines, Sun Microsystems, Inc. http://java.sun.com/security/seccodeguide.html