



jamk.fi

Analyzing Java EE application security with SonarQube

Author(s) Paananen Timo

Master's thesis
April 2016
Cyber Security
Degree Programme in Information Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Author(s) Paananen, Timo	Type of publication Master's thesis	Date 4.4.2016
	Number of pages 90	Language of publication: English
		Permission for web publication: x
Title of publication Analyzing Java EE application security with SonarQube		
Degree programme Information Technology Cyber security		
Supervisor(s) Häkkinen, Antti; Rantonen Mika		
Assigned by Kela, Social Insurance Institute of Finland		
<p>Abstract</p> <p>The master's thesis studied SonarQube's capabilities to find security weaknesses in a Java EE application.</p> <p>Firstly, the master's thesis aims to recognize possible points where a developer could cause security weakness to application, by misusing the technologies used by the assigner. Secondly the master's thesis presents a static application analysis and SonarQube. In the last part , the master's thesis creates a knowledge base of different security weaknesses and vulnerability scoring system.</p> <p>The test application was developed for the master's thesis implementation phase and it contained recognized weaknesses. The test application was analyzed with SonarQube using two different rule sets. The first rule set was collected from the SonarQube vanilla installation, and the second set consisted of enriched rules added from the plugins installed separately.</p> <p>The SonarQube was able to find security related issues with both of these rule sets. The latter one was able to find 40% more issues. Neither of the rule set produced any false positive issues. Also, the way how the SonarQube presents issues supports the developers learning by showing non-compliant and compliant code for each rule.</p>		
Keywords/tags (subjects) Java EE, SAST, static application security testing, SonarQube, security		
Miscellaneous		

Tekijä(t) Paananen, Timo	Julkaisun laji Opinnäytetyö, YAMK	Päivämäärä 4.4.2016
	Sivumäärä 90	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty: x
Työn nimi Analyzing Java EE application security with SonarQube		
Tutkinto-ohjelma Information Technology Cyber security		
Työn ohjaaja(t) Häkkinen, Antti; Rantonen Mika		
Toimeksiantaja(t) Kela		
<p>Tiivistelmä</p> <p>Opinnäytetyössä tutkittiin staattisen lähdekoodianalysointityökalun SonarQube kykyä tunnistaa Java EE -sovelluksista tietoturvaheikkouksia.</p> <p>Aluksi tunnistettiin toimeksiantajan käyttämistä Java EE teknologioista kohdat, joissa sovelluskehittäjät voisivat mahdollisesti aiheuttaa sovellukseen tietoturvaheikkouksia. Opinnäytetyössä esitellään erilaiset tietoturvaheikkouksien ja -haavoittuvuuksien pisteytysjärjestelmät.</p> <p>Opinnäytetyön toteutusosaa varten toteutettiin testisovellus, joka sisälsi tunnistettuja tietoturvaheikkouksia. Tätä sovellusta analysointiin SonarQubella käyttämällä kahta eri sääntökokoelmaa, joista toinen oli kasattu SonarQuben perusasennuksesta ja toiseen oli lisätty sääntöjä jälkikäteen asennetuista laajennoksista.</p> <p>SonarQube kykeni löytämään molemmilla sääntökokoelmilla tietoturvaheikkouksia Java EE -sovelluksista. Laajempi sääntökokoelma löysi kuitenkin 40% enemmän heikkouksia. Kumpikaan sääntökokoelmista ei tuottanut virheellisiä havaintoja. SonarQuben tapa esittää löydetyt virheet tukee sovelluskehittäjien oppimista, esittämällä sekä virheellisen tavan että oikeaoppisen tavan toteuttaa säännön tarkastelema kohta.</p>		
Avainsanat (asiasanat) Java EE, SAST, static application security testing, SonarQube, security		
Muut tiedot		

Contents

1 Introduction.....	5
1.1 Reasearch methods.....	5
1.2 Research limitations.....	6
1.3 Assigner.....	6
2 Research background.....	8
2.1 Java EE.....	8
2.1.1 Java EE containers.....	9
2.1.2 Technologies and their security perspectives.....	12
2.1.2.1 JSR-224 Java API for XML-Based Web Services (JAX-WS) 2.2.....	12
2.1.2.2 JSR-236 Concurrency Utilities for Java EE 1.0.....	14
2.1.2.3 JSR-250 Common Annotations for the Java Platform 1.2.....	15
2.1.2.4 JSR-318 Interceptors 1.2.....	17
2.1.2.5 JSR- 338 Java Persistence API 2.1.....	18
2.1.2.6 JSR-339 Java API for ReSTful Web Services (JAX-RS) 2.0.....	19
2.1.2.7 JSR-340 Java Servlet 3.1.....	21
2.1.2.8 JSR-341 Expression Language 3.0.....	23
2.1.2.9 JSR-343 Java Message Service API 2.0.....	23
2.1.2.10 JSR-344 JavaServer Faces 2.2.....	24
2.1.2.11 JSR-345 Enterprise JavaBean 3.2.....	26
2.1.2.12 JSR-346 Context and Dependency Injection for Java 1.1.....	27
2.1.2.13 JSR-349 Bean Validation 1.1.....	28
2.1.2.14 JSR-352 Batch Application for the Java Platform.....	29
2.1.2.15 JSR-907 Java Transaction API (JTA) 1.2.....	30
2.1.3 Security layers.....	31
2.2 Static Application Testing.....	32
2.2.1 SonarQube.....	33
2.3 Application vulnerabilities and how to measure them....	35
2.3.1 Common Weaknesses Scoring System.....	36
2.3.2 Common Vulnerability Scoring System.....	37
2.4 Theoretical framework.....	38
3 Methodology.....	40
3.1 Data collection.....	40
3.2 Static Application Testing environment.....	40

3.2.1 SonarQube setup.....	41
3.2.2 Target of analysis.....	42
4 Results.....	43
4.1 Creating quality profiles.....	43
4.2 Results of the vanilla installations.....	44
4.3 Security tuned installations results.....	47
4.4 Analysis of results.....	50
5 Conclusions.....	53
Appendices.....	59
Appendix A: CWSS submetrics.....	59
Appendix B: CVSS metric vectors.....	61
Appendix C: CVSS score formulas.....	62
Appendix D: Identified weaknesses in Java EE technologies and their implementation references.....	63
Appendix E: Java plugins security rules profile.....	68
Appendix F: Security rules profile from multiple plugins.....	74
Appendix G: Security issues with Java plugin.....	85
Appendix H: Security issues with multiple plugins.....	88

Tables

Table 1: Mandatory binding provider context properties.....	13
Table 2: Common annotations.....	16
Table 3: Test environment component versions.....	41
Table 4: Java EE security rules in Java Plugin.....	45
Table 5: Java EE related issues found with Java Plugin's security rules....	47
Table 6: Java EE related security rules in second quality profile.....	49
Table 7: Java EE related issues found with security rules from multiple plugins.....	50
Table 8: Issues mapped to identified weaknesses.....	52

Figures

Figure 1: Java EE Containers (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D. 1-12).....	10
Figure 2: Java EE APIs in Application Client Container (Jendrock, Cervera- navarro, Evans, Haase & Markito N.D,1-15).....	10
Figure 3: Java EE APIs in Web Container (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D,. 1-13).....	11
Figure 4: Java EE APIs in EJB Container (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D, 1-14).....	12
Figure 5: JSF execute and render lifecycle (see org. Burns 2013, 55)....	25

Figure 6: Batch Applications components (See org. Vignola 2013, 5).....	29
Figure 7: SonarSource architecture (See org. Gignoux 2015).....	34
Figure 8: The three tenets of information system security (See org. Kim; Solomon. 10).....	35
Figure 9: CWSS Score Formulas (Coley; Martin. 2014).....	37
Figure 10: CVSS information example.....	38
Figure 11: Test applications structure.....	42
Figure 12: Create new quality profile dialog.....	43
Figure 13: Analyze time with Java plugin security rules profile.....	47
Figure 14: Analyze time with extended security rules profile.....	50
Figure 15: Issue explanation.....	52

Terminology

Term	Explanation
API	Application programming interface
Compiler	Software that translates human readable code to lower level language which computers can understand
HTTP	HyperText Transport Protocol
Java	Programming language introduced by Sun Microsystems
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
JAXB	Java Architecture for XML Binding
JNDI	Java Naming and Directory Interface
JPQL	Java Persistence Query Language
OWASP	Open Web Application Security Project
POJO	Plain Old Java Object
REST	Representational State Transfer
RMI-IIOP	Remote Method Invocation (RMI) interface over the Internet Inter-Orb protocol (IIOP)
SAST	Static Application Security Testing
SOAP	Simple Object Access Protocol
SQL	sequel query language used in databases
WS-I	Web Service interoperability organization
WSDL	Web Service Description Language which is used to describe web service interfaces to clients

1 Introduction

This master's thesis researches how Java EE application security can be improved by using static application security testing tools. For that it first presents what security elements Java programming language has and what extra security elements and techniques Java Enterprise edition offers.

After presenting what is considered as security elements in Java EE applications master's thesis introduces what a Static Application Testing is and how the static application testing tools work in the technical perspective. From here the master's thesis moves towards to static application security testing and tries to explain what static application security testing is. In the end of this part is small preview to the common used static application analysis tools.

Last part of background information of this master's thesis defines how applications security is measured. So it is possible to define if static application security analysis help to build better software.

1.1 Reasearch methods

The master's thesis applies Static Application Security Testing to a Java EE application and analyses the results. Analysis tries to resolve how well Java EE security models and mechanisms are covered by the chosen tool set and if they provide useful information for developers. In the master's thesis research background chapter relevant Java EE technologies are studied and charted for possible points that developers could misuse thus exposing the application to security weaknesses. Empirical research is based on the following research questions.

1. How does static application security testing improve Java EE

applications security?

2. Which elements forms Java EE applications security?
3. How does SonarQube work?

1.2 Research limitations

This master's thesis does not study application development processes which uses static application security testing nor does it not contain how to apply static application security testing results to application development. Java SE security models and mechanisms are also left outside of this master's thesis with possible weaknesses that are caused by misusing the Java language itself. In the static application security part SonarSource's SonarQube product is used to produce analysis data, this limitation is based on the assigner's needs.

The master's thesis does not express any opinions to design patters that could increase or decrease application security. Additionally, Java EE technologies are limited to those technologies that master's thesis assigner uses and are presented through chapters 2.1.1.1 to 2.1.1.15.

1.3 Assigner

The assigner of the master's thesis is Kela, the Social Insurance Institute of Finland that operates directly under the supervision of Finnish parliament. Kela's mission is to secure the income and promote the health of the entire nation, and to support the capacity of individual citizens to care for themselves. Kela is a reliable, efficient and socially responsible actor. It has an active role in developing social security and its implementation. The social security provided by Kela is clearly understandable, reasonable in amount and

delivered with a good standard of quality. Kela's service is the best in the public sector. (Operations 2014.)

Kela has its own ICT department, that employees about 500 persons and it develops all its own benefit systems used to make a benefit decision. About 100 persons from all of Kela ICT department staff are Java developers. There is currently an on-going project called ARKKI, that aims to renew all the benefit systems from the mainframe to Java EE.

2 Research background

This chapter presents all background knowledge for this master's thesis. The first subchapter explains what Java EE and its security aspects are and how Java EE technologies are used. After that static application testing is introduced and SonarSource's SonarQube tool. The end of this chapter explains how applications security can be measured and which are the factors are affecting application's security in code level.

2.1 Java EE

Java EE means basically Java's Enterprise Edition (Java EE) which uses Java Standard Edition (Java SE) specification as its base. Java EE contains two sections, Java EE platform specification and a set of specifications for technologies.

Java community (Idemichiel 2014) writes that the Java EE Platform specification is an umbrella specification that does not directly define Java EE APIs. The Java EE platform specification only references to other specifications and defines how they work together. Java community continues to tell that beside being an umbrella specification Java EE platform specification defines other attributes of the platform such as security, deployment, transactions and interoperability.

As earlier mentioned The Java EE platform specification only refers to the Java APIs specification and Java EE contains totally 33 different specification, including platform specification. From these 33 specifications 24 specify purely technologies and APIs such as JAVA API for RESTful web services (JAX-RS) 2.0. The rest of the specifications does not directly define technologies but defines how something should be implemented from architectural point of view, how to use a specific pattern, or container behaviour. These nine

specifications are listed in the following list. The technology specifications are presented in chapter 2.1.2.

- JSR 45: Debugging support for other languages
- JSR 52: Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JSR 77: J2EE Management 1.1
- JSR 88: Java EE application deployment
- JSR 109: Implementing Enterprise Web Services 1.3
- JSR 115: Java Authorization Contract for Containers
- JSR 181: Web Service Metadata for the java Platform
- JSR 322: Java EE connector Architecture 1.7
- JSR 342: Java Platform, Enterprise Edition 7

All these different technologies run on top of Java SE and communicate to each other through containers. As Völter, Schmid and Wolff (2002, 44.) explains that containers are an execution environment that provides a federated view to the underlying Java EE API's for the application components. Figure 1 shows how Java EE containers communicate together.

2.1.1 Java EE containers

There are three different containers in Java EE: Application client container, Web container and EJB container. All these containers have their own purpose and supports a set of APIs as well as offer services like security, database access, transaction handling, naming directory, resource injection to components (Goncalves 2013, 3). The software does not need to utilize all these containers only those which it really needs. For example pure back-end that does not provide any graphical user interface needs only EJB container.

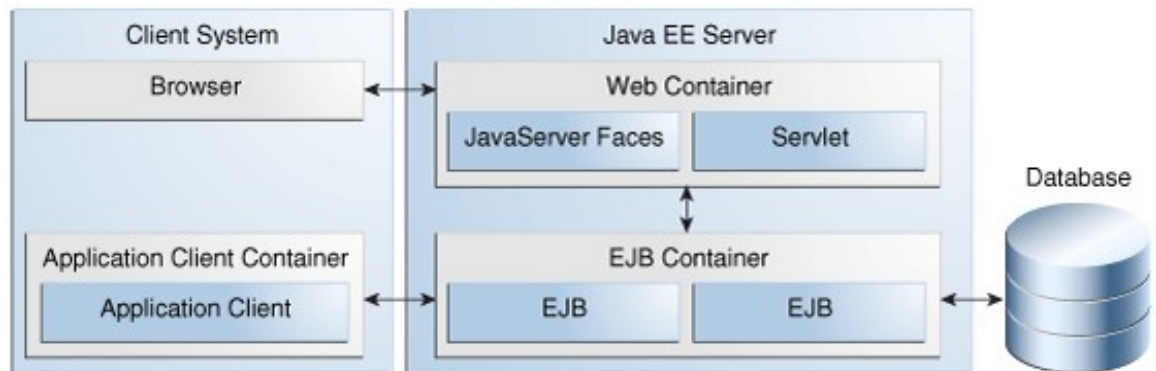


Figure 1: Java EE Containers (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D. 1-12)

Application client container can be used to bring dependency injection, security management and naming service to Java SE software. The application client container uses RMI-IIOP to communicate with EJB container and HTTP to communicate with Web container. (Goncalves 2013, 3). Figure 2 shows what APIs the application client container contains.

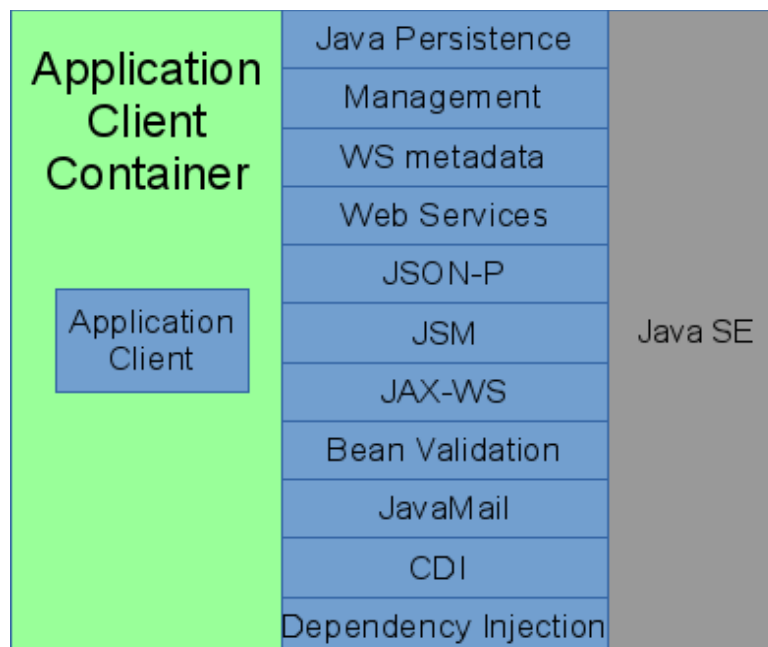


Figure 2: Java EE APIs in Application Client Container (Jendrock, Cervera-navarro, Evans, Haase & Markito N.D,1-15)

Web container is used to produce web pages that are based on technologies like servlets, JSPs, filters, listeners, JSF and web services. The web container instantiates, initializes and invokes servlet and filters. It also supports HTTP and HTTPS protocols that are used to communicate with web browsers.

(Goncalves 2013, 3). Figure 3 shows what APIs the Web Container contains.

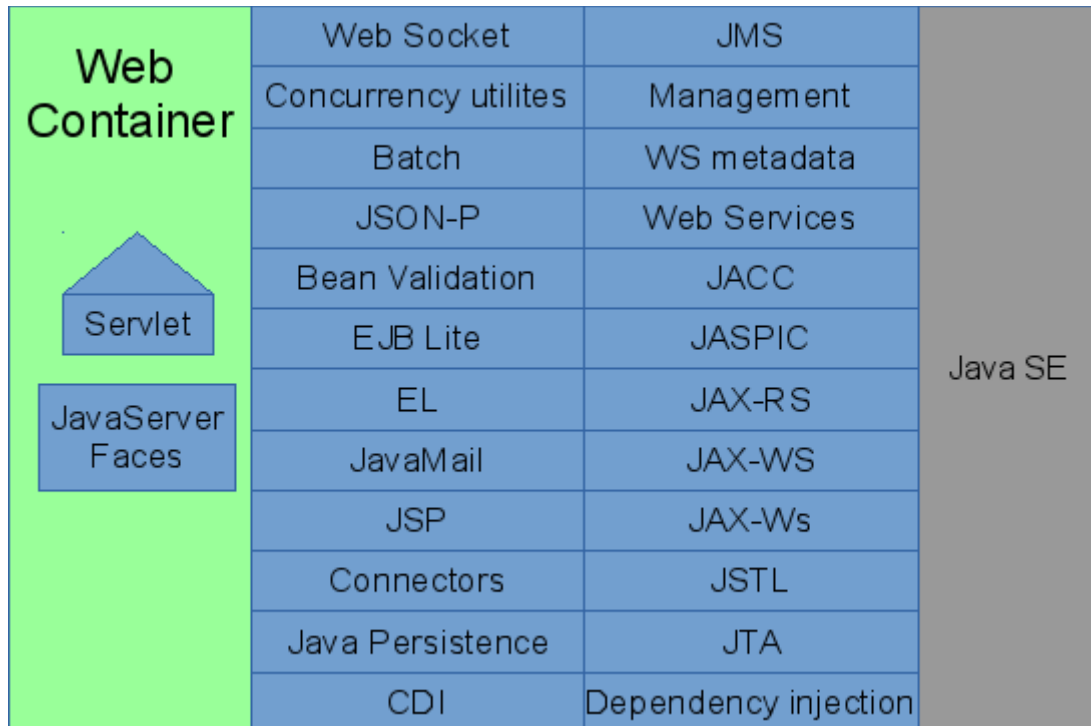


Figure 3: Java EE APIs in Web Container (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D., 1-13)

EJB container is used in back-end components that contains Java EE application's business logic. EJB container is responsible for managing the execution of the Enterprise Java Bean (EJB). This container provide services like transactions, security, concurrency, distribution, naming services, or possibility to be invoked asynchronously. (Goncalves 2013, 3). Figure 4 shows what APIs the EJB Container contains.

EJB Container <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 10px auto;">EJB</div>	Concurrency utilities	JMS	Java SE
	Batch	Management	
	JSON-P	WS metadata	
	CDI	Web Services	
	Dependency Injection	JACC	
	JavaMail	JASPIC	
	Java persistence	Bean Validation	
	JTA	JAX-RS	
	Connectors	JAX-Ws	

Figure 4: Java EE APIs in EJB Container (Jendrock; Cervera-navarro; Evans; Haase; Markito N.D, 1-14)

2.1.2 Technologies and their security perspectives

This chapter introduces different Java EE technologies, their common use cases and security mechanisms. The focus in introductions is on those technologies that are designed for developers to use and those which involve security heavily.

2.1.2.1 JSR-224 Java API for XML-Based Web Services (JAX-WS) 2.2

JAX-WS specification is a follow-up to JAX-RCP by extending it using JAXB XML mapping rules instead of defining their own mapping rules, adding support for SOAP 1.2, WSDL 2.0 and WS-I Basic Profile 1.1, adding better metadata annotation support and aligning with, complementing the security API defined by JSR-183 and describing techniques and mechanisms for versioning services. Other updates that JAX-WS brings are improvements for document/message centric usage, which is listed as follow. (Kotamraju 2011, 1-2.)

- Supports client side asynchronous operations
- Improve separating XML message format and transport mechanism
- Simplifies clients and services access to the message
- Supports message based session management

JAX-WS client is implemented by using *javax.xml.ws.Service* class and *javax.xml.ws.Dispatch* and *javax.xml.ws.BindingProvider* interfaces. Service class represents WSDL service. The actual service instance can be acquired dynamically through Service.create method or statically by implementing its own class that extends Service class. Both ways need service an endpoint address and a Java type that represents the service. BindingProvider interface provides protocol bindings to client and methods to manipulate binding provider's context. Mandatory binding provider context properties that can be manipulated are presented in Table 1. The Dispatch interface gives developer access to XML message level. The XML message can be accessed in message payload or message mode, where message payload gives access to the data sent and in message mode to the protocol specific message structure.(Katomraju 2011, 55 - 68)

Table 1: Mandatory binding provider context properties

Property	Description
javax.xml.ws.endpoint.address	Endpoints address
javax.xml.ws.security.auth.username	User name for HTTP basic authentication
javax.xml.ws.security.auth.password	Password for HTTP basic authentication
javax.xml.ws.session.maintain	Indicates whether client is prepared to participate in services session

To implement a service, the endpoint specification offers an API that contains total of four interface and class in package *javax.xml.ws*. The endpoint service low level implementation can be accomplished by implementing the class that implements Provider interface. The provider interface is the counterpart for clients Dispatch interface and can also operate in two modes Payload and

Message. When using Payload mode the provider must be typed to implement `Provider<Source>` and in Message mode `Provider<Message>`. The mode is defined by using type level annotation called `@ServiceMode`. Higher level services are implemented as normal Java classes and interfaces where implementing class is annotated with `@WebService` annotation that defines port name, service name, target namespace and endpoint interface. The interface can be annotated with `@WebService` annotation that defines the service's namespace. The implemented service is published by `Endpoint` class. `Endpoint` instance is first acquired with `create` method which takes service implementation class's instance as parameter. After that service is published with the endpoint's `publish` method. (Katomraju 2011, 71-81.)

The other two parts are `WebServiceContext` interface and `W3EndpointReferenceBuilder` class. `WebServiceContext` interface is a shared context for all objects that involves handling invocation of the web service. If `WebServiceContext` methods are invoked out side of web service methods, the invocation implementation should throw *`java.lang.IllegalStateException`*. The `WebServiceContext` is thread safe and uses thread-locals to identify correct information between different requests. `W3EndpointReferenceBuilder` can be used to create `Endpoint` reference to another web service endpoint. (Katomraju 2011, 81 - 84.)

JSR-224 simplifies developing web services to a developer. There is still two possible points where mistakes can be made. The first one is in client side, where it is possible to use `BindingProvider` to hard code basic authentication username and password. The other one relates to XML namespaces which should be defined to web services but `@WebService` annotation does not require namespace. The missing namespace can cause conflict in service calls if two or more services have the same name and same endpoint address.

2.1.2.2 JSR-236 Concurrency Utilities for Java EE 1.0

JSR-236 offers concurrency API to developer to use in his or her application. It

extends Java SE's concurrency API so that Java EE containers can manage threads that are created in application. If Java SE's concurrency API is used in Java EE environment it can cause weird race conditions because the container is not aware of these threads and that they are accessing shared resources like data source.(Concurrency Utilities for Java EE 2013, 2-1 - 2-2).

As Vidergar states in his white paper incorrectly coded concurrency handling can cause race condition, deadlock or denial of service through poor performance or scalability. He continues that concurrency mistakes are hard to notice in testing phase and they might rise only in certain situations like under heavy load. (Vidergar, Stender 2008, 2).

2.1.2.3 JSR-250 Common Annotations for the Java Platform 1.2

The JSR-250 specification defines set of annotation that are used in other specifications and how they are handled in case of inheritance. The specification defines fourteen different annotations that are explained in Table 2.

Table 2: Common annotations

Annotations name	Annotations description
javax.annotation.Generated	Indicates that code is generated by defined generator. Can also imply date of generation.
javax.annotation.Resource	Declares resource reference. Resources name, type, authentication type, jndi lookup name, shareable and mapped name can be defined by this annotation.
javax.annotation.Resources	Permit to define multiple javax.annotation.Resource annotations to class, method or field.
javax.annotation.PostConstruct	Defines method that can be used to initialize the object after injections
javax.annotation.PreDestroy	Defines method that will be invoked before container removes the bean. Can be used, for example, to clean resources properly before removing the bean.
javax.annotation.Priority	Indicates order of the classes been used.
javax.annotation.security.RunAs	Defines role that is used to run application. Role must be mapped to user or group of security realm.
javax.annotation.security.RolesAllowed	Defines roles that are permitted to invoke methods in class. Can be used in class or method level.
javax.annotation.PermitAll	Allows all security roles to invoke methods of class. Can be used in class or method level.
javax.annotation.DenyAll	Denies all security roles to invoke methods of class. Can be used in class or method level.
javax.annotation.security.DeclareRoles	Declares security roles that are used in the application. Can be used only in class level.
javax.annotation.sql.DataSourceDefinition	Defines containers datasource and to registering it by JNDI. This annotation permits to define datasource type (driver class), URL, username, password, database name, port number, server name, isolation level, connection transaction capabilities, pool size properties, idle time, maximum statement count, login timeout and vendor specific properties.
javax.annotation.sql.DataSourceDefinitions	Permit to define multiple javax.annotation.sql.DataSourceDefinition annotations to class.
javax.annotation.ManagedBean	Defines object to be container managed. Can be used only in class level.

This master's thesis is only interested in RunAs, RolesAllowed, PermitAll, DenyAll, DeclaredRoles and DataSourceDefinition annotations. The most interesting common annotation is DataSourceDefinition because it allows

definition of username and password. As the JSR-250 specification states defining password is not recommended at least in production code (Mordani 2013, 2-26).

2.1.2.4 JSR-318 Interceptors 1.2

The JSR-318 defines interceptor mechanism that can be used to interpose on business method invocation or specific event. There for the interceptors can be divided into two different categories: business method interceptors and interceptors for life-cycle event callbacks. All business method interceptors implements method with `@AroundInvoke` annotation that is able to execute code before and after the actual method invocation. Life cycle event callback interceptor implements a method or methods annotated with `@AroundConstructor`, `@PostConstruct`, `@PreDestroy` or `@AroundTimeout` annotations. The following list presents what can be done with each of these annotations. (Vatkina 2013a, 11.)

- `@AroundConstructor` annotated methods can execute code before and after invocation of constructor
- `@PostConstruct` annotated methods can execute code after bean's injection is done
- `@PreDestroy` annotated methods can execute code on an event when container is going to remove the bean
- `@AroundTimeout` annotated methods will be executed by Timer service. Annotation can define calendar-based schedule, specific time, specific amount of time elapsed or specific interval that fires interceptors event.

JSR-381 does not specify anything security related, although interceptor mechanisms can be used to improve application's security. For example there could be an interceptor in a public web service interface that handles all technical exceptions and throws user friendly exception to caller or one that logs all incoming requests; however these are application specific custom

implementations, and therefore they are not within of this master's thesis.

2.1.2.5 JSR- 338 Java Persistence API 2.1

JSR-338 defines API for managing persistence and mapping relation database to Java objects (DeMichiel 2013, 21). As the specification defines both mappings database to java classes and query language to manage database, these should be handled separately.

JSR-338 defines a large set of annotations which can be used to as metadata that represents database definitions. These annotations are applied to classes that represent database structure and are annotated with `@Entity` annotation. Each field or property that represent a column in a database is annotated with corresponding metadata annotation in entity class, for example, a primary key field is annotated with `@Id` annotation or a one-to-one relationship is marked with `@OneToOne` annotation. These annotations are highly tight to database design and application's needs from the database and the way they are used varies from case to case.

The query language part is more interesting in the perspective of this master's thesis because SQL injections are conducted by querying or updating a database. By using Java Persistence API queries can be executed by using two different techniques `NamedQueries` and `CriteriaQueries`. `NamedQueries` are static expression and they can be defined by using Java Persistence Query Language (JPQL) or using native SQL (DeMichiel 2013, 151-152). `Criteria` API queries are defined by using object-based query definition objects (DeMichiel 2013, 235). Following code snippets shows the usage of `NamedQueries` and `Criteria` API queries.

```

public Person getPersonNamedQuery(String name) {
    Query personQuery = this.entityManager.
        createNamedQuery("SELECT P FROM Person P WHERE
p.name = :name", Person.class);
    personQuery.setParameter("name", name);
    return (Person)personQuery.getSingleResult();
}

public Person getPersonCriteriaAPI(String name) {
    CriteriaBuilder builder = this.entityManager
        .getCriteriaBuilder();
    CriteriaQuery<Person> query =
        builder.createQuery(Person.class);
    ParameterExpression<Integer> parameter =
        builder.parameter(Integer.class);
    Root<Person> person = query.from(Person.class);
    query.select(person).where(builder.equal(person.get("name"),
        name));
    return entityManager.createQuery(query)
        .getSingleResult();
}

```

Gnanasundar expresses in his blog post that JPQL and native queries have an injection weakness if not used correctly (Gnanasundar N.D). This is because neither way cannot detect if query itself is parsed or not and the parsed parameter can have harmful characters that are not escaped. The setParameter method will escape harmful characters, and injections are not possible. For the sake of clarity the following code snippets shows an example of this.

```

public Person getPersonNamedQueryUsingParsing(String name) {
    String queryString =
        "SELECT P FROM Person P WHERE p.name=" + name;
    Query personQuery = this.entityManager
        .createQuery(queryString, Person.class);
    return (Person)personQuery.getSingleResult();
}

```

2.1.2.6 JSR-339 Java API for ReSTful Web Services (JAX-RS) 2.0

JSR 339 defines how to implement Representational State Transfer (REST) Web services and their clients with Java. Components for implementing a REST are resources, providers, filters, interceptors and validation.

Resources are the main part of REST services because they are entry points

to the service. Resources are defined by using `@Path` annotation. Resource class can have properties or fields that are annotated with `@MatrixParam`, `@QueryParam`, `@PathParam`, `@CookieParam`, `@HeaderParam` or `@Context`. Values for these properties or fields are extracted from the corresponding part of the request. These annotations are supported only for resources that use per-request life cycle.

Resource methods present methods in resource class with `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD` or `@OPTIONS` annotation on them. These annotations represent the HTTP method used to access the resource.

Resource methods can return `Void`, `Response` or `GenericEntity` which each are mapped to 200 or 204 HTTP return code to indicate that all went fine.

Resource methods also can have `@Path` annotation to specify an additional URL or a parameter that has to be present to invoke the resource method.

`@Path` annotation takes `String` as its value that presents URL's part or placeholder for the parameter's name. As following code snippet shows, `deletePerson` method is invoked from URL `http://localhost/persons/9` when `DELETE` HTTP method is used.

```
@Path("persons")
public class PersonService {

    @DELETE
    @Path("{id}")
    public Response deletePerson(@PathParam("id")
        String id) {
        ...
        return response;
    }
}
```

Method could even define in the `@Path` annotation that id-parameter must match the regular expression which would be defined as `@Path("{path: ([ABC])"}")` and which would match only those requests that have only A, B or C character in the id part. (Pericas-Geertsen & Potociar 2013, 13-16.)

JAX-RS implementation can be extended in run-time by using providers. Providers provide `MessageBodyReader` and `MessageBodyWriter` implementations that are responsible for converting messages to Java objects and Java objects to messages. Providers themselves are not prone to security weaknesses as a technology is, but if `MessageBodyWriter` or `MessageBodyReader` are badly implemented they can cause side effects that cannot be predicted. (Pericas-Geertsen & Potociar 2013, 27- 30.)

Providers, on the other hand, offer a way to extend JAX-RS to support different type of messages interceptors and filters, and enables developer to add different capabilities to JAX-RS service, like logging, authentication, confidentiality, entity compression etc. Interceptors wrap method invocation and can execute code around invocation as filters execute code at the extension point; however they do not wrap method invocation. Filters offer four extension points for the response and the request at client and at server end. These are invoked when a request leaves from client or when it is received by server and when a response is sent from server and received by client. (Pericas-Geertsen & Potociar 2013, 37- 40.)

JAX-RS relies on the JSR 349 bean validation specification which is introduced later in this chapter.

In perspective of this master's thesis, the interesting parts of the specification are resources with JAX-RS annotated properties, non-public methods with `@Path` annotation and validation. JAX-RS annotated resource properties should not be written in any other life cycle phase than creation because that can cause errors in concurrency. Non-public methods with `@Path` annotation cannot be accessed outside of application and therefore are unnecessary. The regular expression capabilities of `@Path` annotations are also interesting because by using them it is possible to white list valid paths.

2.1.2.7 JSR-340 Java Servlet 3.1

JSR-340 is used to produce dynamic content in web applications. Servlets use by default HTTP and optionally HTTPS protocols to communicate with clients,

for example web browsers, and all JSR-340 containers must support at least HTTP protocol. `HttpServlet` subclass adds dedicated methods for all HTTP methods that call automatically `GenericServlets` service method. The servlets are initialized through `init`-method of the `Servlet` interface, therefore developers should not do any container related operation in class construction methods because the servlet might not be yet active in the container. The servlet container can handle concurrent requests; however the developer can alternate this behaviour by implementing `SingleThreadModel` interface which forces the container to serialize requests or to maintaining pool of servlet instances. Another way to achieve this is to mark the service method as synchronized; however this could have a huge performance impact. As the servlets support concurrency it is important that the developer is aware that `Request` and `Response` classes methods are not thread safe, except `startAsync` and `complete` methods. If other methods are called by multiple thread the container can not ensure that results are correct from the caller's point of view. (Wai Chang & Mordani 2013, 2-5 - 2-21.)

When using HTTP or HTTPS protocol the servlets support cookies by `HttpServletRequest` class `getCookies` method. Also, `HttpOnly` cookies are supported that indicated that cookies cannot be read on client side by scripts. When HTTPS protocol is used with the servlet, container expose cipher suite bit size of the algorithm and SSL session id to developer to use. And if the request includes SSL certificate it is also exposed to the developer. The Servlet API also allows controlling timeout time of the sessions. If the timeout is set to zero it will be handled as infinite timeout. (Wai Chang & Mordani 2013. 3-29 - 3-30.)

Static resources can be accessed from the servlet by using `getResource` or `getResourceAsStream` methods. These methods load resource relative to root of context or relative to `META-INF/resources` from jars that are in `WEB-INF/lib` folder. These methods should not be used to obtain dynamic resources because they will not be processed. (Wai Chang & Mordani 2013. 4-41.)

The Response class has sendRedirect method which can be used to redirect client to a different URL. The method's parameter should be the absolute path of the new address. Response class has also sendError method which should be used to send an error message to a client with appropriate headers and body content.(Wai Chang & Mordani 2013, 5-48.)

2.1.2.8 JSR-341 Expression Language 3.0

JSR 341 specifies simple language that is syntax restricted to the evaluation expressions which can be used to access underlying Java object's values and methods for example from the presentation layer. Expression language uses `$ {}` and `#{}` expression to imply expressions which are evaluated in run-time. (Chung 2013, 2-3.)

As expression language only allows developers to access the underlying Java objects it will itself not present any possible ways for the developer to misuse it, and because of that, it will not expose any security weaknesses to developer and will not be discussed within the scope of this master's thesis.

2.1.2.9 JSR-343 Java Message Service API 2.0

JSR-343 specifies standard Java API and architectural solution for enterprise messaging products which are used in a company's internal network. Enterprise messaging systems can include non-java products, which may be communicated with this API. JSR-343 defines two types of communication, point-to-point and publish and subscribe. In point to point communication the client will send a message straight to another client by using an abstract queue, and in the publish and subscribe solution the client sends messages to topic, and clients that want receive messages will subscribe to the same topic. (Deakin 2013, 12 -13.)

The specification states that it will not include any security API for controlling the privacy and integrity of the messages. An as using the new 2.0 API is not error prone because it only includes some interfaces and their methods. For these reasons, JSR-343 will be left out of this master's thesis.

2.1.2.10 JSR-344 JavaServer Faces 2.2

JSR-344 specifies user interface framework for Java web applications. The JSF framework is based on JavaServer Pages, Expression language 3.0, Servlets, JavaBean and JavaServer Pages Standard Taglibrary. JSF provides easy-of-use reusable components for building a user interface. Framework also allows developer to develop their own reusable components. It also simplifies data migration to and from the user interface and offers a simple model for wiring user interface events to server side code (Burns 2013, 44 & 47.)

When building user interface with JavaServer Faces the view layer is developed by using components representing different UI elements that create tree of components. All components have common ancestor *javax.faces.component.UIComponent* and have unique identifier in context of *Naming Container*. All components in the view can be accessed through component tree by their unique identifier. Components are transformed to HTML output stream by *javax.faces.render.Renderer* implementations that are assigned to the component. The value of the component is bound to it by using expression languages value expressions, which will wire up the value in the page to the corresponding Java Bean variable. The bound value is converted from *java.lang.String* to appropriate by *javax.faces.convert.Converter* assigned to the component. The component can have *javax.faces.validator.Validator* implementation that is responsible for validating the given data. The expression language can also be used to method expression which presents the method calls to the corresponding object's public methods. (Burns 2013, 85 - 96, 165-166.)

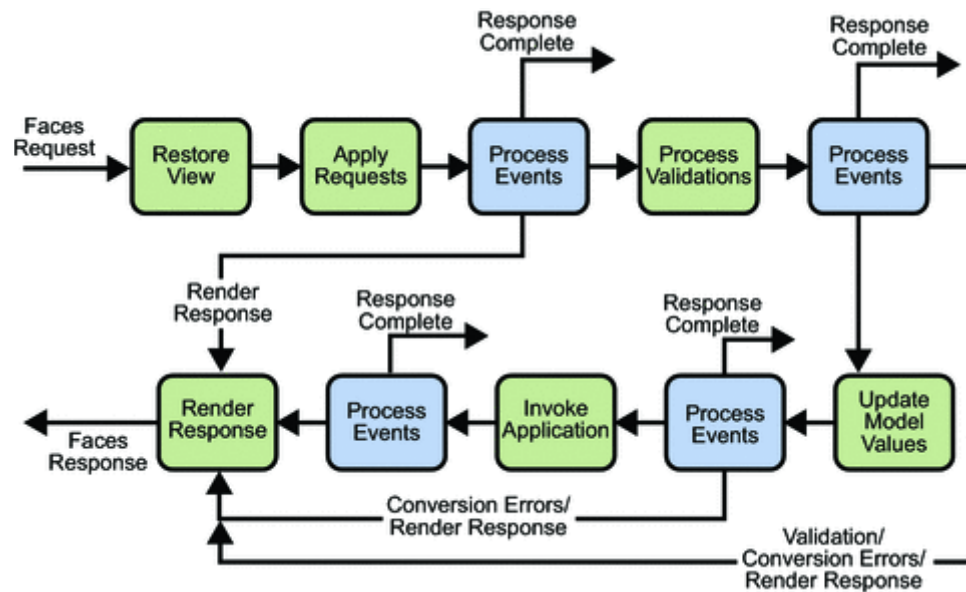


Figure 5: JSF execute and render lifecycle (see org. Burns 2013, 55)

JSR-344 defines execute and render life cycle for handling incoming requests. The life cycle consist of six different phases which all have their own responsibilities for handling requests. The life cycle handles one view and all its components at the time. The life cycle takes care of component's states in any given moment. As Figure 5 shows the view is first restored, then values are processed by converters and validator, and updated to model and last before sending the response the application itself is invoked.(Burns 2013, 56-60).

As JavaServer Faces technology is used to implement web applications all common web applications are present in it. Open Web Application Security Projects has a top ten list of most common web application security risks that have to be acknowledged when developing user interfaces with JavaServer Faces. It is also possible to developers to misuse the framework and access the component's raw value through component tree and use an invalidated value.

2.1.2.11 JSR-345 Enterprise JavaBean 3.2

Enterprise JavaBeans aim to be standard component architecture for building object-oriented Java EE applications. They simplify application's development by hiding low-level transaction and state management details, multi-threading, connection pooling and other complex low-level APIs. (Vatkina 2013b, 26.)

JSR-345 defines three types of enterprise beans; session objects, message-driven objects and entity objects which are optional. Session beans are executed on behalf of the client, they can be transaction-aware and update shared data but does not represent the data itself. Message-driven objects have the same characteristics as session beans ; however they are always asynchronously invoked and are stateless. Entity objects represents the data and are long lived. (Vatkina 2013b, 32-33.)

Session objects have three subtypes; stateful, stateless and singleton session beans. The main difference with these subtypes is how beans are presented for clients. Stateful session bean instances are always client specific, once the client acquires references to the bean. The client can invoke bean's business methods multiple times and it will always get the same instance of the bean. The instance is destroyed after the client invokes `@Remove` annotated method or if the instance is passivated specified amount of time. The EJB container passivate an instance when the container implementation specific caching algorithm decides so, generally it should be done at the end of each method although the instance cannot be passivated within transaction. Stateful session bean can store the state of the client but the state is lost if the instance is destroyed. Invoking destroyed stateful bean will throw *javax.ejb.NoSuchEJBException*. Stateless session beans otherwise are not client specific and the client can be sure that it does get the same instance reference when invoking a bean multiple times. Container will create and destroy stateless session bean instances on demand of active clients; therefore a client's state cannot be stored in stateless session beans. Singleton session bean instances are shared among all clients. There can be

only one instance of a singleton session bean per JVM. As singleton session bean instance is shared, it should not store client specific state. (Vatkina 2013b, 83 - 85, 92 - 94, 98-99.)

JSR-345 have some points where a developer can make a mistake and cause security weaknesses. If the developer stores client's state to a stateless session bean it could be exposed to a different client. This could be hard to find in run-time because the client that sets the state could get the same instance back in the next invocation or the bean's instance could be destroyed before any other clients acquire it. Also, the developer could store a client's state to singleton session bean, and the state would be shared among all clients. Depending on business case this could be a wanted behaviour; however developers should still pay attention to this. This is much easier to find out than a case with stateless session beans because this happens during every invocation.

2.1.2.12 JSR-346 Context and Dependency Injection for Java 1.1

JSR-346 specification aims to provide a set of services that can help to improve the application's structure. The specified services are following:

- Life cycle for stateful objects that are bound to life cycle contexts.
- Type-safe dependency injection mechanism that can select dependencies either on development or deployment time.
- Integration to JSR-341
- Way to decorate injected objects
- Way to associate interceptors with injected objects
- Event notification model
- Addition to servlet specifications contexts, conversational context

- Portable extensions to integrate with the container

JSR-345 is not meant to be used alone but along with other specifications such as JSR-318, JSR-330, JSR-344, JSR-345 or JSR-349. (Muir, 2013, 1-3.)

As JSR-345 shows through examples, the only thing that is left to the application developer, in perspective of JSR-345, is adding annotations to code and if needed implementing marker annotations. All the other things are done under the hood by containers and other specifications, and as configuration is evaluated in development or deployment time, it is highly unlikely that any error could be slipped to production. Specification even states that all definition errors are developer errors and are cached in container's initialization time. The last point for the discovery of definition errors is application's startup, as JSR-345 specifies that containers must perform bean discovery and raise definition an exception if any definition errors exists. (Muir 2013, 4-10, 111-112.)

Therefore, this specification will itself not cause possible security weaknesses that a developer can implement, thus it is not interesting from the point of view of this master's thesis.

2.1.2.13 JSR-349 Bean Validation 1.1

JSR-349 defines validation mechanism and object level constraint declarations for Java. Constraints are defined by using annotations that have been marked with `@Constraint` annotation. Constrains can be applied to types, fields, methods, constructors, parameters or other constraints if composition is needed. Constrains define a valid value of the target or multiple Java types if used to cross-parameter validation. `ConstraintValidators` are used to implement the constraint's validation logic. The validation framework automatically invokes `ConstraintValidators` for the correct constraints and validates the given value. If constraint annotation is used for unsupported type `UnexpectedTypeException` will be thrown. If the constrains definition is not valid `ConstraintDefinitionException` will be thrown in run-time. (Bernard 2013,

5-9.)

The life cycle of constraint validation object is not defined and validation providers can cache these instances for future use. Although initialize -method is invoked before using the implementation, the value should not be stored into instances state. (Bernard 2013, 25.)

JSR-349 have some points of failure that can cause unwanted behaviour in application's run-time. Developers should be able to get a warning if constraint's definition is not valid or constraint is applied to unsupported type, or the validated object's value is stored to constraint validator's state.

2.1.2.14 JSR-352 Batch Application for the Java Platform

JSR-352 specification defines Java API for applications that are intended for bulk processing and usually are long running and computing or date intensive. Batch application can be divided into seven components; JobRepository, JobOperator, Job, Step, ItemReader, ItemProcessor and ItemWriter as shows. JSR-352 can execute a batch sequentially or parallel. (vignola 2013, 5)

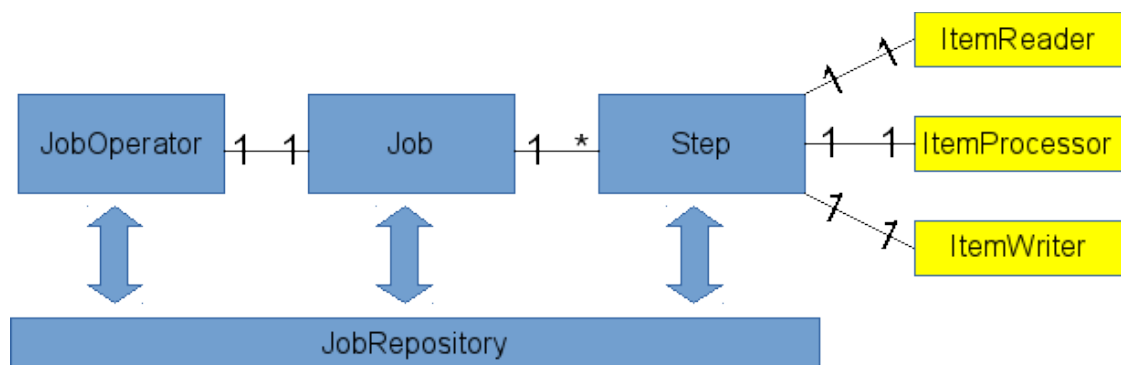


Figure 6: Batch Applications components (See org. Vignola 2013, 5)

The job is specified by using Job Specification Language (JSL) which JSR-352 defines. JSL is implemented by using XML and has its own XML element for Job and step components and their attributes. (Vignola 2013, 19.)

The batch API or the JSL does not expose any possible security themselves as APIs are simple and JST is used to control these APIs. Security

weaknesses in batch applications are caused by misuse of other technologies used to implement batch application's functionality. Therefore, this specification is not within scope of this master's thesis.

2.1.2.15 JSR-907 Java Transaction API (JTA) 1.2

JSR-907 specifies interfaces between transaction manager and the application, resource managers and application servers. The interface for application's is a high level interface that can be used to define the application's transactions. The interface for application server allows application server to control transaction boundaries for the application being managed. The specification also offers Java mapping for the industry standard X/Open XA protocol, so transactional resource manager can participate in a global transaction that is controlled by external transaction manager. (Parkinson 2013, 7.)

In transactional Java EE applications there are two of kinds of transactions, container managed and user managed. If the developer uses container managed transactions he/she need only to define transaction type for methods. Available types are REQUIRED, REQUIRES_NEW, MANDATORY, NOT_SUPPORTED and NEVER which can be assigned by using Transactional annotation. When using user managed transactions the developer will handle start and end of the transaction progmatically. For this specification offers UserTransaction interface which has following methods to interact with the transaction. (Parkinson 2013, 11 - 25.)

- begin: Create a new transaction and associate it with current thread
- commit: Complete the transaction associated with current thread
- getStatus: obtain the status of the transaction associated with current thread
- rollback: Rollback the transaction associated with current thread
- setRollbackOnly: Modify the transaction associated so that its only

outcome can be roll back

- `setTransactionTimeout`: Modify current threads transactions timeout

The container managed transactions do not leave much place for misuses and therefore are not within scope of this master's thesis. Misused user transactions, on the other hand can cause dramatic errors in run-time. If a transaction is not completed after an operation that needs it, the resource will be reserved longer and will cause performance issues.

2.1.3 Security layers

Java EE applications have three security layers application, transport and message. Developers usually handle application-layer's security and transport-layer and message-layer security are handled by infrastructure or application server maintainers.

The containers which are introduced in chapter 2.1 provide application-layer security. As Jendrock and partners report containers can be secured using declarative or programmatic security; declarative means using either annotations in code or deployment descriptors to define secured resources and their authentication and authorization information programmatic security is embedded in the application itself. The advantage of application-layer security is that the security is uniquely tailored for the application and it is fine-grained with application-specific settings. On the other hand, it is dependent on security attributes that cannot be transferred between application types, and support for different protocols makes it vulnerable and data is lost or contained with the point of vulnerability. (Jendrock and co N.D, 47-8 - 47-9.)

The transport-layer security is used to secure data transport between server and client. It is fully implemented outside of the application and so it is out of the focus of this master's thesis.

Message-layer security is used to secure SOAP messages or SOAP message attachments. As Jendrock and partners lights up in their Java EE tutorial, WSS is used to implement message-layer security and it is not part of Java EE platform; therefore, this level static security analysis is also out of the focus for this master's thesis (Jendrock and co N.D, 47-8 - 47-9).

2.2 Static Application Testing

Static application testing is usually done by developers with tools that are specially developed to analyse source code. Applications source code itself has not been executed while performing static application testing so it does not need run-time environment for the application. Instead, static application testing aims to find violence of best practices rather than trying to prove that an application works as planned (Ayewah, Pugh, Hovermeyer, Morgenthaler & Penix 2010, 22). These best practices include practices from code styling, line length or use of parentheses to application design that can cause cyclomatic complexity. These best practise violations can cause serious vulnerabilities like SQL-injections where user can execute unwanted SQL-statement to database (Livshits & Lam N.D, 3). The following code example would be marked as issue by static application analysis because it does concatenate input parameters to SQL-query.

```
public boolean authenticate(String username, String password){
    Connection conn = getConnection();
    Statement statement =
        connection.createStatement(
            "SELECT * FROM accounts WHERE username =\'" +
            username + "\' and password=\'" +
            password + "\'")
    );
    ResultSet rs = statement.execeuteQuery();
    return rs.next();
}
```

Another example is infinite recursive loops that cause application to crash eventually to stack overflow (Ayewah and co, 23).

```
public boolean isAuthenticated() {  
    return this.isAuthenticated();  
}
```

Beside finding the best practice violation static application analysis tools can show code metrics by counting depth of nesting, cyclomatic complexity or distinct paths from one line of code to the another (Graham, Veenendall, Evans & Black, 73).

2.2.1 SonarQube

SonarQube is an open source platform for source code quality management that is developed by SonarSource. SonarQube has four components server, database, plugins and scanner. Server has two processes, a web server which offers user interface to explore analysed projects and Elasticsearch based search server which is used by user interface for queries. The database is used to store installation's configuration and project analyses. Scanner is the component that do the hard part of work by analysing projects and sending the results to the SonarQube server. The fourth component contains plugins that can be used to extend SonarQube platform's functionalities. SonarQube architecture is presented in Figure 7. (Gigleux 2015.)

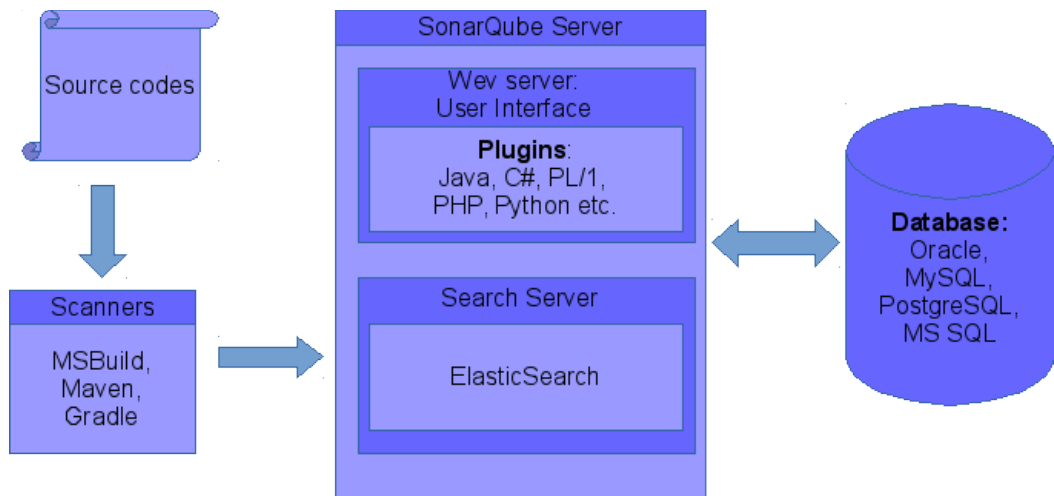


Figure 7: SonarSource architecture (See org. Gignoux 2015)

SonarQube can analyse multiple programming languages through language plugins (Mallet 2016). Language plugins contains default analysing rules for the language they support, however, more rules can be added through rule plugins like checkstyle plugin which enforces coding convention standards for Java.

There are two types of rules: standard rules and security related rules. Standard rules should not produce any false positive issues where as security related rules can produce some false positive issues. Every rule represents a single issue type in the code like Exception should be caught instead of Throwable. Rules can have tags defined, which makes it easier to categorize rules, tags can be something like security, CWE or convention. (Campbell 2015a.)

The Java Plugin itself contains more than 300 rules for analysing Java source code. There are rules for coding conventions, bug detection and security problems. Security related rules contains checks for some CERT and CWE weaknesses as for some SANS top 25 most dangerous software errors and some OWASP top 10 weaknesses. All rules are CWE compatible so it is possible to search rules by CWE identifier. (Racodon 2016.)

For all the issues that SonarQube recognizes a severity classification is

applied:, blocker, critical, major, minor and info (Campbell 2015b). Default severity of the rule can be changed to express better company policies for example when creating a quality profile. Quality profiles are sets of rules that are assigned to projects under analysis (Campbell 2015c). Ideally all projects that are implemented with same language would use the same quality profile so they can be compared to each other.

2.3 Application vulnerabilities and how to measure them

Application vulnerabilities are weaknesses in application that users can exploit. By exploiting an application's weakness malicious user can affect applications functionalities and gain some benefit from this or influence the service's confidentiality, integrity or availability which are the three tenets of information security as shown in Figure 8.

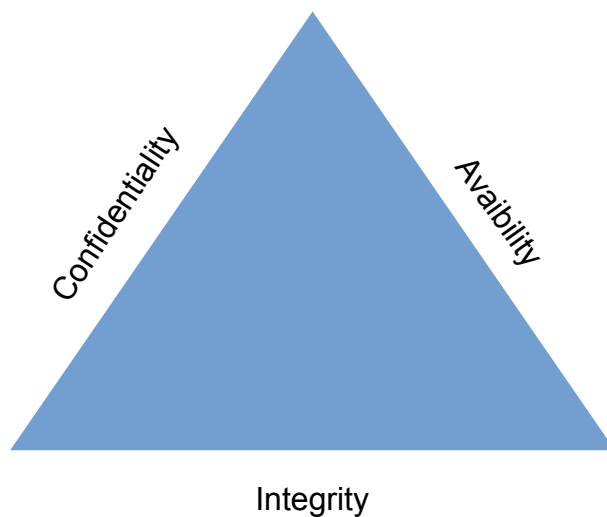


Figure 8: The three tenets of information system security (See org. Kim; Solomon. 10)

Weaknesses can be ranked by using different scoring systems like CWSS or CVSS. These different scoring systems are presented next.

2.3.1 Common Weaknesses Scoring System

CWSS (Common Weaknesses Scoring System) offers a mechanism for ranking weaknesses in consistent, flexible and open manner. It uses three main metric groups to rank weaknesses: Base Finding, Attack Surface and Environmental metric group. Each of these metric groups contains multiple other metrics that are used to calculate the weakness ranking value (Coley & Martin 2014). A full function listing of the sub metrics can be found in Appendix A: CWSS submetrics.

Base finding metric group expresses the risk of weakness, how accurate the finding is and the strength of controls. Attack Surface metric group handles how easily attacker can exploit the weakness and Environmental metric group specifies the environment and operational context of the weakness. CWSS ranking value is calculated by placing the value to each factor in the Base Finding metric group and then calculating them to Base Finding sub score which will be between 0 to 100. This same method is used to calculate Attack Surface and Environmental metric groups which produces value between zero to one. Finally, these three values are multiplied together to gain final CWSS score. The formulas for each sub score are presented in Figure 9. (Coley & Martin 2014.)

Base Finding subscore**formula**

$$\text{Base} = [(10 * \text{TechnicalImpact} + 5 * (\text{AcquiredPrivilege} + \text{acquiredPrivilegeLayer}) + 5 * \text{FindingConfidence}) * f(\text{TechnicalImpact}) * \text{InternalControlEffectiveness}] * 4.0$$

$$f(\text{TechnicalImpact}) = 0 \text{ if } \text{TechnicalImpact} = 0; \text{ otherwise } f(\text{TechnicalImpact}) = 1$$

Attack surface subscore**formula**

$$[20 * (\text{RequiredPrivilege} + \text{RequiredPrivilegeLayer} + \text{AccessVector}) + 20 * \text{DeploymentScope} + 15 * \text{levelOfInteractions} + 5 * \text{AuthenticationStrength}] / 100.0$$

Environmental subscore**formula**

$$[(10 * \text{BusinessImpact} + 3 * \text{LikelihoodOfDiscovery} + 4 * \text{LikelihoodOfExploit}) + 3 * \text{Prevalence}] * f(\text{BusinessImpact}) * \text{ExternalControlEffectiveness}] / 20.0$$

$$f(\text{BusinessImpact}) = 0 \text{ if } \text{BusinessImpact} == 0; \text{ otherwise } f(\text{BusinessImpact}) = 1$$

Figure 9: CWSS Score Formulas (Coley; Martin. 2014)

2.3.2 Common Vulnerability Scoring System

CVSS (Common Vulnerability Scoring System) is a similar scoring system than CWSS; however, instead of weaknesses it focuses straight on the vulnerabilities. So it is kind of one layer higher scoring system as earlier mentioned vulnerabilities are weaknesses that have been exploited. Many vulnerability that is listed in www.cvedetails.com site uses CVSS contains link to the actual weaknesses in <http://cwe.mitre.org> site. As CWSS CVSS also uses three sub metrics, however, they are called metric groups in its ranking system: Base, Temporal and Environmental.

Base metric group is used to characterise vulnerabilities of those variables that will not change over the time or the environment. Temporal metric group is used to those variables that might changes over time but not across run-

time environment. As in CWSS, the environmental metric group represents those variables that are relevant and unique to specific run-time environment. This helps organizations to mitigate vulnerability by making changes to run-time environment. (Hanford, 5-6). All these three metric groups contain sets of metrics variables that help define vulnerability score. Only those metric variables that belong to Base metric group are mandatory to calculate CVSS score as Appendix B: CVSS metric vectors explains.

CVSS produces a ranking value between 0.0 and 10.0. It also produces a vector string that represents values that are used to form CVSS value. Its format is (AV:N/AC:M/Au:N/C:N/I:P/A:N) and it is usually displayed with vulnerability details as Figure 10 shows. The ranking value itself is calculated using formulas in Appendix C: CVSS score formulas. (Hanford 2015, 18 - 19.)

Vulnerability Summary for CVE-2015-3439

Original release date: 08/05/2015
Last revised: 09/03/2015
Source: US-CERT/NIST

Overview

Cross-site scripting (XSS) vulnerability in the Ephox (formerly Moxiecode) plupload.flash.swf shim 2.1.2 in Plupload, as used in WordPress 3.9.x, 4.0.x, and 4.1.x before 4.1.2 and other products, allows remote attackers to execute same-origin JavaScript functions via the target parameter, as demonstrated by executing a certain click function, related to `_init.as` and `_fireEvent.as`.

Impact

CVSS Severity (version 2.0):

CVSS v2 Base Score: 4.3 MEDIUM

Vector: (AV:N/AC:M/Au:N/C:N/I:P/A:N) ([legend](#))

Impact Subscore: 2.9

Exploitability Subscore: 8.6

Figure 10: CVSS information example

2.4 Theoretical framework

It is widely studied that with static application testing it is possible to find different kinds of coding mistakes that can cause bugs to application. But does static application testing find security related mistakes that in the Java EE are

more or less related to metadata annotations for underlying containers that take care of implementation of security mechanisms? Although it is possible for developers to do programmatic security to applications, does static application analysis know which is the correct way to use the APIs that make it possible? Or does static application analysis only find coding mistakes that are based Java SE's technologies?

I expected that static application analysis does not raise issues about inadequate security definitions in Java EE technologies and that it can understand the misuse of APIs used to produce programmatic security. However, I also expect that by using static application analysis it is possible to implement better security to the Java EE applications through the findings it does from misuses of the Java SE technologies and misuses of the Java language itself.

3 Methodology

This chapter presents the methodology used to determinate the usefulness of SonarQube from security perspective. First subchapter presents how data is produced. After that the testing environment is presented and in the end target of the analysis.

3.1 Data collection

To study what weaknesses the SonarQube can find, a special application was developed that contains security weaknesses that are identified in this master's thesis, CWE and OWASP top ten. Not all weaknesses from those sources are implemented to the application because it would expand the master's thesis too much. The selected weaknesses and their implementation reference points are listed in Appendix D: Identified weaknesses in Java EE technologies and their implementation references

The application is analysed with SonarQube against two quality profiles. The first one contains only security related rules that are provided by Java Plugin, listed in Appendix E: Java plugins security rules profile, and the second one is expanded with rules from third party plugins offering security related rules, listed in Appendix F: Security rules profile from multiple plugins. After analysing SonarQube the results are mirrored against the lists of known weaknesses.

3.2 Static Application Testing environment

This chapter and its sub chapters defines the testing environment and all components in it and presents the application under the analysis.

3.2.1 SonarQube setup

SonarQube is installed to a virtual machine running Arch Linux which has Oracle JRE 7 installed for the SonarQube. To SonarQube is added Java plugin and PMD, Findbugs, Web and XML plugins. From rules that are provided by these plugin, two quality profiles are created. The first one contain security related rules from Java plugin and the second one contain security related rules from all of these plugins. Another virtual machine is used for standalone PostgreSQL database where the SonarQube stores configurations and results. The analysis is done by using SonarQube scanner for Maven from developer desktop machine. Specific version of each component are presented in Table 3.

Table 3: Test environment component versions

Component	Version	Comment
SonarQube Platform	5.4	
SonarQube scanner for Maven	3.0.1	org.sonarsource.scanner.maven:sonar-maven-plugin:3.0.1
Java Plugin	3.12	For the Java Language support and default rules
PMD plugin	2.5	For enabling more security related rules
Findbugs	3.3	For enabling more security related rules
Web plugin	2.4	For enabling more security related rules
XML plugin	1.4.1	For enabling more security related rules
PostgreSQL	9.5.1	
Java Runtime Environment	1.7.0_79	64bit build 15

3.2.2 Target of analysis

The analysed application is a simple application for registering responses to invitations with authentication. The application also has management interface for creating events and their invitations. The application's ready state represents software under development. There is only 3041 lines of code so application can be concerned as very small application.

The application was developed for the master's thesis and contains known security weaknesses. The application's source code can be found in GitHub.com repository called *summons* owned by *Timizki* and its tag called *thesis_frozen*. The application is implemented by using the technologies used by the assigner. The technologies were also limited further to contain only those technologies that were recognized to have possible misuse weaknesses. All weaknesses are marked with comment "SECURITY_WEAKNESS" in the code. The Application's structure is presented in Figure 11.

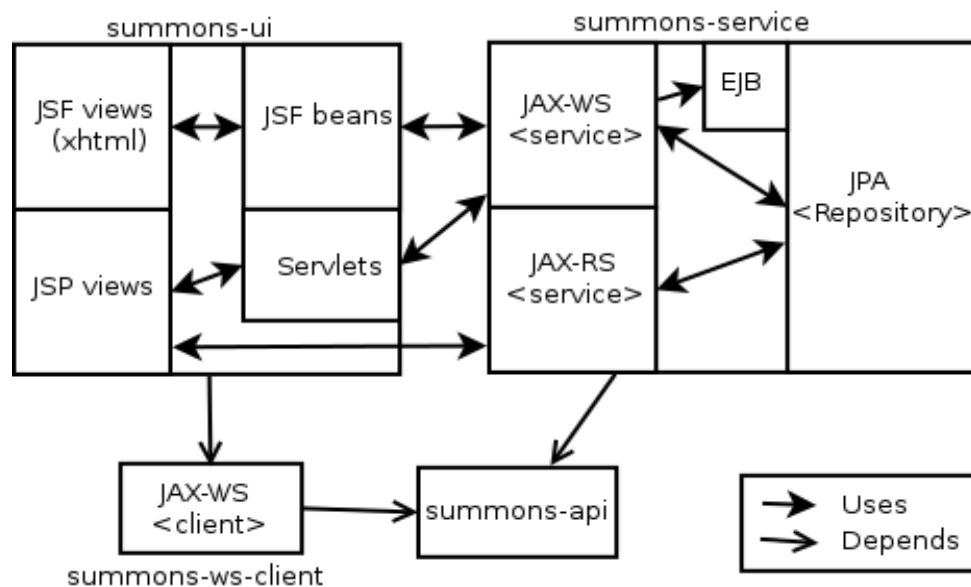


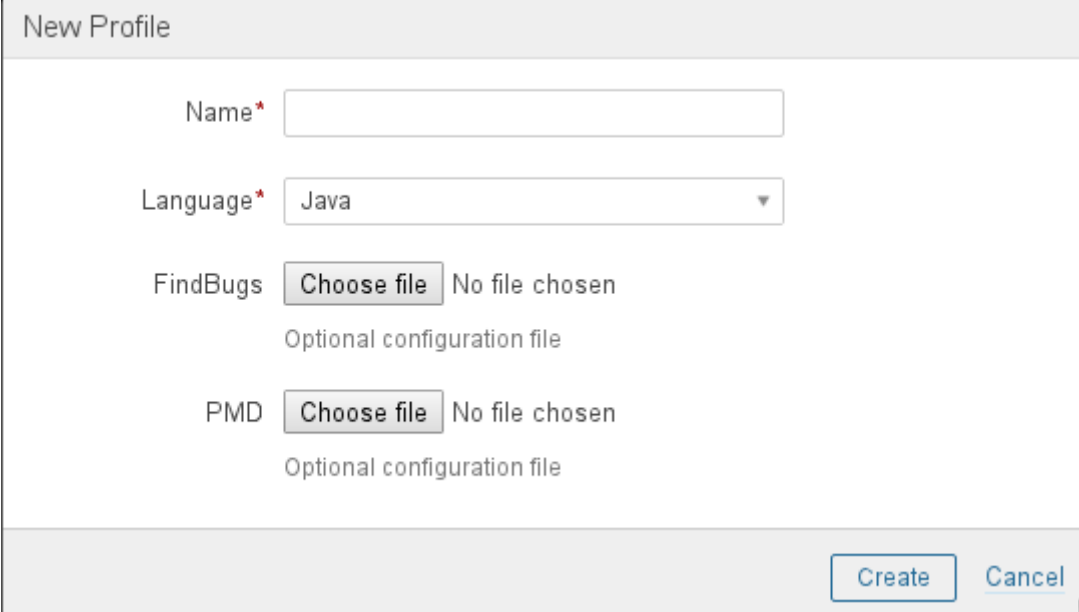
Figure 11: Test applications structure

4 Results

This chapter presents all results gained from the analysis of the test application. First is presented how quality profiles were created. After that, the results from the analysis of the Java plugins security rules profile are presented. After that, the results of the analysis with security rules profile from multiple plugins are discussed. The last subchapter analyses the results.

4.1 Creating quality profiles

Empty quality profiles were created from the Quality Profiles page in SonarQube. SonarQube asks only the quality profile's name and language, however, each additional plugin can add optional fields to the form as Figure 12 shows.



New Profile

Name*

Language*

FindBugs No file chosen
Optional configuration file

PMD No file chosen
Optional configuration file

Figure 12: Create new quality profile dialog

After empty quality profiles were created, they had to be populated with rules which was carried out through rule query page. The rule queries were limited by tags and rule repositories. For both quality profiles the same tags was used: security, cwe, owasp-a1, owasp-a2, owasp-a3, owasp-a4, owasp-a6 and owasp-a7. For the first quality profile, vanilla installation's profile, rules were limited to SonarQube Java repository and for the second one all repositories were included. At the end of each query all results were added to the quality profile through bulk change button.

4.2 Results of the vanilla installations

The quality profile created from rules offered by the Java plugin contained 70 security related rules. Ten of those were classified as blocker, 43 as critical, 15 as major and 2 minor. Every rule had at least two tags where one was security related and another might have been non-security related like *bug*. The quality profile contained only 13 rules related to any Java EE technologies, all the other rules were targeted for Java language in generally. The Java EE related rules are listed in Table 4. Twelve of these rules were classified as critical and the last one was classified as major.

Table 4: Java EE security rules in Java Plugin

Severity	Rule name
Critical	"HttpServletRequest.getRequestId()" should not be used
Critical	Cookies should be "secure"
Critical	Credentials should not be hard-coded
Critical	Exceptions should not be thrown from servlet methods
Major	Exit methods should not be called
Critical	Fields in a "Serializable" class should either be transient or serializable
Critical	HTTP referers should not be relied on
Critical	Non-serializable objects should not be stored in "HttpSessions"
Critical	Security constraints should be defined
Critical	Struts validation forms should have unique names
Critical	Values passed to SQL commands should be sanitized
Critical	Web applications should use validation filters
Critical	Web applications should not have a "main" method

With this quality profile SonarQube was able to detect 24 issues. From these issues 14 were Java EE related and they are presented in Table 5. Two rules raised four issues each, in different places. Thus, only eight unique issues were found. All issues that was found are listed in Appendix G: Security issues with Java plugin appendix contains also more information about the issues.

Table 5: Java EE related issues found with Java Plugin's security rules

Keys	Severity	Tags	message
JDS-1	CRITICAL	cwe, jee, owasp-a7, security, websphere	Add "security-constraint" elements to this descriptor.
JDS-12, JDS-13, JDS-14, JDS-15	CRITICAL	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JDS-16	CRITICAL	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "sendRedirect".
JDS-2	CRITICAL	injection, owasp-a1, security	Add a validation filter to this "web.xml".
JDS-5	CRITICAL	cwe, owasp-a2, owasp-a6, security	Add the "secure" attribute to this cookie
JDS-6	CRITICAL	bug, cwe	Make "Invitation" serializable or don't store it in the session.
JDS-3, JDS-4, JSD-8, JDS-10	CRITICAL	bug, cwe, serialization	Make "<variable name>" transient or serializable.
JDS-18	CRITICAL	cwe, owasp-a2, sans-top25-porous, security	Remove this hard-coded password.

The analysis was started with Maven by command *mvn clean verify sonar:sonar* which started sonar-maven-plugin. The whole build took approximate only 42 seconds as Figure 13 shows.

```

INFO] -----
INFO] Reactor Summary:
INFO]
INFO] summons ..... SUCCESS [ 33.804 s]
INFO] summons-api ..... SUCCESS [ 4.012 s]
INFO] summons-ws-client ..... SUCCESS [ 0.415 s]
INFO] summons-service ..... SUCCESS [ 1.590 s]
INFO] summons-ui ..... SUCCESS [ 0.460 s]
INFO] summons-ear ..... SUCCESS [ 0.598 s]
INFO] -----
INFO] BUILD SUCCESS
INFO] -----
INFO] Total time: 41.616 s
INFO] Finished at: 2016-03-28T16:48:51+03:00
INFO] Final Memory: 26M/269M
INFO] -----

```

Figure 13: Analyze time with Java plugin security rules profile

4.3 Security tuned installations results

The second analysis was done by using quality profile that contained rules from all plugins; thus, it contained all rules from the first quality profile and 74 more security related rules from other plugins. New rules were gathered from FindBugs, SonarQube Web and PMD plugin repositories. In total the second quality profile contained 144 active security related rules. From these rules 40 were Java EE related, and they are listed in Table 6.

Table 6: Java EE related security rules in second quality profile

Severity	Rule Name
Critical	"HttpServletRequest.getRequestedSessionId()" should not be used
Critical	Cookies should be "secure"
Critical	Credentials should not be hard-coded
Critical	Exceptions should not be thrown from servlet methods
Major	Exit methods should not be called
Critical	Fields in a "Serializable" class should either be transient or serializable
Critical	HTTP referers should not be relied on
Critical	Non-serializable objects should not be stored in "HttpSessions"
Critical	Security constraints should be defined
Critical	Struts validation forms should have unique names
Critical	Values passed to SQL commands should be sanitized
Critical	Web applications should use validation filters
Critical	Web applications should not have a "main" method
Major	Absolute path traversal in servlet
Major	Relative path traversal in servlet
Minor	Security - A prepared statement is generated from a nonconstant String
Minor	Security - Found JAX-RS REST Endpoint
Major	Security - Hard Coded Password
Blocker	Security - Hardcoded constant database password
Minor	Security - HTTP Headers Untrusted
Major	Security - HTTP Response splitting vulnerability
Major	Security - JSP reflected cross site scripting vulnerability
Critical	Security - Nonconstant string passed to execute method on an SQL statement
Critical	Security - Potential SQL/JPQL Injection (JPA)
Critical	Security - JSP reflected cross site scripting vulnerability
Critical	Security - Potential XSS in JSP
Minor	Security - Potentially Sensitive Data in Cookie
Critical	Security - Potential XSS in Servlet
Critical	Security - Servlet reflected cross site scripting vulnerability
Critical	Security - Servlet reflected cross site scripting vulnerability
Minor	Security - Untrusted Content-Type Header
Minor	Security - Untrusted Hostname Header
Minor	Security - Untrusted Query String
Minor	Security - Untrusted Referer Header
Minor	Security - Untrusted Servlet Parameter
Minor	Security - Untrusted Session Cookie Value
Minor	Security - Untrusted User-Agent Header
Major	Security - Unvalidated Redirect
Major	Security - XSSRequestWrapper is Weak XSS Protection

Before rerunning a static analysis with Maven the old analysis was deleted from SonarQube platform and the second quality profile was set as default profile. The analysis produced 27 issues at this time. From these issues seventeen were Java EE related which are listed in Table 7. Also this time two rules found four issues different places and because of that only eleven unique issues were identified. As with the first analysis all the rest issues were related to Java in generally. All issues that were found are listed in Appendix H: Security issues with multiple plugins.

Table 7: Java EE related issues found with security rules from multiple plugins

Key	Severity	Tags	Message
JES-25	CRITICAL	injection, owasp-a1, security	Add a validation filter to this "web.xml".
JES-24	CRITICAL	cwe, jee, owasp-a7, security, websphere	Add "security-constraint" elements to this descriptor.
JES-17, JES-18, JES-19, JES-20	CRITICAL	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JES-21	CRITICAL	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "sendRedirect".
JES-10	CRITICAL	cwe, owasp-a2, owasp-a6, security	Add the "secure" attribute to this cookie
JES-22	MAJOR	cwe, owasp-a3	HTTP parameter directly written to HTTP header output in io.vksn.summons.ui.servlet.RedirectServlet.doGet(HttpServletRequest, HttpServletResponse)
JES-11	CRITICAL	bug, cwe	Make "Invitation" serializable or don't store it in the session.
JES-8, JES-9, JES-13, JES-15	CRITICAL	bug, cwe, serialization	Make "<variable name>" transient or serializable.
JES-2	CRITICAL	cwe, owasp-a2, sans-top25-porous, security	Remove this hard-coded password.

Key	Severity	Tags	Message
JES-5	CRITICAL	cwe, injection, owasp-a1, security, wasc	The query is potentially vulnerable SQL/JPQL injection
JES-23	MAJOR	cwe, security, wasc	Unvalidated Redirect

The second analysis took 12 seconds longer to finish meaning a total approximate build time of 54 seconds as Figure 14 shows.

```
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] summons ..... SUCCESS [ 47.102 s]
[INFO] summons-api ..... SUCCESS [ 3.113 s]
[INFO] summons-ws-client ..... SUCCESS [ 0.456 s]
[INFO] summons-service ..... SUCCESS [ 1.373 s]
[INFO] summons-ui ..... SUCCESS [ 0.358 s]
[INFO] summons-ear ..... SUCCESS [ 0.795 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 53.928 s
[INFO] Finished at: 2016-03-30T20:36:02+03:00
[INFO] Final Memory: 30M/265M
[INFO] -----
```

Figure 14: Analyze time with extended security rules profile

4.4 Analysis of results

SonarQube was able to find issues from Java SE technologies as Java EE technologies, although there were more Java SE related issues than Java EE related issue. This was expected as the application contained more Java SE related code than Java EE. The SonarQube was able to find eleven unique weaknesses out of the 35 Java EE weaknesses implemented in the test application. In total there were 44 different recognized weaknesses, one of those, MTW-17, was detected by a compiler giving compile error.

The vanilla installation found seven of these eleven issues whereas security tuned installation found all the issues. However, when rule count is taken into account there is not such a big difference since the security tuned installation had more than 50% more rules. The issues which were found are mapped to identified weaknesses in Table 8. SonarQube did found one issue that was not

identified in the master's thesis. The issue was about storing non-serializable object to session. Even though SonarQube was able to find some weaknesses from owasp top ten categories, it does not mean that all of them were to be found because weaknesses in the categories can be implemented in many ways and almost in all Java EE technologies.

Table 8: Issues mapped to identified weaknesses

Weakness identifier	Issue identifier
MTW-2	JDS-18 JES-2
MTW-8	JES-5
OWASP-A1	JDS-2, JDS-25
OWASP-A3	JES-22
OWASP-A7	JDS-1, JES-24
OWASP-A10	JES-23
CWE-536	JDS-12, JDS-13, JDS-14, JDS-15, JDS-16, JES-17, JES-18, JES-19, JES-20, JES-21
CWE-594	JDS-4, JDS-10, JDS-3, JDS-8, JES-9, JES-13, JES-15, JES-8
CWE-600	JDS-12, JDS-13, JDS-14, JDS-15, JDS-16, JES-17, JES-18, JES-19, JES-20, JES-21
CWE-601	JES-23
CWE-614	JDS-5, JES-10
Weakness not identified	JDS-6, JES-11

Even though SonarQube was not able to find all weaknesses it is not catastrophe because SonarQube did not report any false positive issues either, which makes SonarQube more reliable and decreases the time to be used to ensure the correctness of bugs. The time used to resolve bug was reduced even more because most of the rules in SonarQube offers clear

information on what is a noncompliant solution and what is a compliant solution as Figure 15 shows. This offers also a good way to all developers to learn how some specific thing in code should be done even if they do not fix the bug. All issues were also clearly listed and categorized by file.

Noncompliant Code Example

```
try {  
    // do something that might throw an UnsupportedOperationException or UnsupportedEncodingException  
} catch (Exception e) { // Noncompliant  
    // log exception ...  
}
```

Compliant Solution

```
try {  
    // do something  
} catch (UnsupportedEncodingException|UnsupportedDataTypeException|RuntimeException e) {  
    // log exception ...  
}
```

Figure 15: Issue explanation

Even though there were more than 50% more rules in the second quality profile the analysis time did not increase proportionately as much. This encourages to create quality profiles that contain a large amount of rules and still developers could run the analysis quickly and often and get feedback from SonarQube.

SonarQube's Java plugin itself offers good set of rules that are able find reliable security issues from Java EE and Java SE code. And when quality profiles are enriched rules from the other plugins SonarQube's capabilities are even more reliable; however, there is still long way to go before SonarQube can find even all the major security issues. There is also some parties that works with the SonarSource to bring more security rules to SonarQube's Java plugin. This will ensure that SonarQube's capability to find security issues will get better in the future.

5 Conclusions

The objectives that were set to this master's thesis were reasonable and achievable. The research questions and restrictions set to the master's thesis guided me through the work. Without the restrictions research background would have increased too much to be carefully covered in the implementation phase.

The results shows that SonarQube can be used to improve Java EE application's security because SonarQube is Java EE and Java SE technology aware more or less. The test application developed for this master's thesis should be peer reviewed or evaluated to make sure that the security weaknesses are implemented correctly to be found. Also, the security weaknesses recognized in this master's thesis marked with identifier MTW-* should be evaluated to ensure that they are real weaknesses.

In the implementation phase when analysing the results it was hard to draw a line which of the rules was related to Java EE and which to Java SE, however in the end it does not matter so much because they all involved the security aspect. To get more reliable results the test applications should contain more security weaknesses. Weaknesses should be implemented in many different ways so that SonarQube's ability to detect different weakness variants could be studied. However, it is time consuming to implement weaknesses so that they mirror even somehow the real world use cases.

SonarQube performed better in detecting weaknesses from the Java EE application than was expected, which is a good thing. However, it still can detect only the tip of the iceberg from the all possible weaknesses. Luckily, SonarSource and other parties are working to improve the SonarQube's security weakness detection capabilities in every release.

To me this master's thesis was very interesting to do because I had to study many Java EE specifications to get an understanding if they have possible security weaknesses, and on the side of that I could gather much valuable

knowledge about those technologies. When studying those technologies there was a worry about what the security state of reference implementations of those technologies is, and it would be interesting to study security of Oracle's JDK and Open JDK. Another issue that came to my mind while writing this master's thesis was how static application testing could effectively be a part of the software development process and how other parties, like project managers, could use information it produces.

List of references

Ayewah, N. Pugh, W. Hovemeyer, D. Morgenthaler, D. Penix J. 2010. Using Static Analysis to Find Bugs. IEEE Computer Society.

Chinnici, R. 2003. Java(TM) API for XML-based Remote Procedure Call (JAX-RPC) Specification ("Specification") Version 1.1

Concurrency Utilities for Java EE. Version 1.0. Final Release. 2013. Oracle America, inc.

Goncalves, A. 2013. Beginning Java EE 7 (Expert voice in Java).Apress.

Graham, D. Van Veenendaal, E. Evans, I. Black, R. 2008. Foundations of software testing ISTQB Certification. Thomson.

Hanford, S. 2015. Common Vulnerability Scoring System V3.0 specification Document.

Jendrock, E., Cervera-navarro, R., Evans, I., Haase K., Markito, W. 2014. Java Platform, Enterprise Edition (The Java EE Tutorial). Oracle.

JSR-196 java Authentication service provider Interface for Container("Specification"). Version 1.1. Maintenance Release. 2013. Oracle America, Inc.

Kim, D. Solomon, M. 2012. Fundamentals of Information System Security. Jones & Bartlett learning.

Kin-man, C. 2013. JavaServer Pages(TM) Specification. Version 2.3. maintenance Release 3.

Livshits, B. Lam, M. N.D. Finding Security Vulnerabilities in Java applications with Static Analysis. Computer Science Department Stanford University.

Martin, R. Coley, S. Kenderdine, J. Piper, L. 2015. CWE Version 2.9. The MITRE Corporation.

OWASP Top 10. 2013. The Open Web Application security Project.

Vidergar, A. Stender, S. 2008. Concurrency Attacks in Web Applications. iSEC Partners, Inc.

Völter, M., Schmid, A., Wolff, E. 2002. Server Component Patterns. John Wiley & Sons ltd.

Bernard, E. 2013. JSR 349 Bean Validation specification. Version 1.1. Final

- Release. Accessed on 22.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/bean_validation-1_1-fr-eval-spec/bean-validation-specification.pdf
- Burns, E. 2013. JSR 344 JavaServer™ Faces Specification. Version 2.2. Final Draft. Accessed on 22.2.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/jsf-2_2-fr-eval-spec/javax.faces-api-2.2-FINAL.zip
- Butek, R. Gallardo, N. 2006. Web services hints and tips: JAX-RPC versus JAX-WS, Part 1. Accessed on 28.7.2015. Retrieved from <http://www.ibm.com/developerworks/library/ws-tip-jaxwsrpc/index.html>
- Campbell, A. 2015a. Rules. Accessed on 4.3.2016. Retrieved from <http://docs.sonarqube.org/display/SONAR/Rules>
- Campbell, A. 2015b. Issues. Accessed on 4.3.2016. Retrieved from <http://docs.sonarqube.org/display/SONAR/Issues>
- Campbell, A. 2015c. Quality profiles. Accessed on 4.3.2016. Retrieved from <http://docs.sonarqube.org/display/SONAR/Quality+Profiles>
- Chung, K. 2013. JSR 341 Expression Language Specification. Version 3.0. Final Release. Accessed on 15.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/el-3_0-fr-eval-spec/EL3.0.FR.pdf
- Coley, S. Martin, B. 2014. Common Weakness Scoring system (CWSS(TM)). Accessed on 4.8.2015. Retrieved from http://cwe.mitre.org/cwss/cwss_v1.0.1.html
- Common Vulnerability Scoring System, V3 Development Update. 2015. Accessed on 5.8.2015. Retrieved from <https://www.first.org/cvss>
- Data Validation. 2013. Accessed on 3.8.2015. https://www.owasp.org/index.php?title=Data_Validation
- Deakin, N. 2013. JSR 343 Java Message Service. Version 2.0. Final Release. Accessed on 15.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/jms-2_0-fr-eval-spec/JMS20.pdf
- DeMichiel, L. 2013. JSR 338: Java Persistence API, version 2.1. Final Release. Accessed on 18.12.2015. Retrieved from http://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf
- Gigleux, A. 2015. Arhitecture and Integration. Accessed on 3.3.2015. Retrieved from <http://docs.sonarqube.org/display/SONAR/Arhitecture+and+Integration>
- Gnanasundar, P. How To Fix SQL Injection: JPA. N.D. Accessed on 18.12.2015. Retrieved from

[to/fix-sql-injection-in-java-persistence-api-jpa](#)

Mordani, R. 2013. JSR 250: Common annotations for the java platform. Maintenance release. version 1.2. Accessed on 11.12.2015. Retrieved from http://download.oracle.com/otn-pub/jcp/common_annotations-1_2-mrel2-eval-spec/jsr-250-1.2-final.pdf

JSR 67: Java(TM) APIs for XML Messaging 1.0. N.D. Referenced 29.7.2015. Retrieved from <https://jcp.org/en/jsr/details?id=67>

JSR-93: Java(TM) API for XML Registries 1.0 (JAXR). N.D. Accessed on 29.7.2015. Retrieved from <https://www.jcp.org/en/jsr/details?id=93>

Kotanraju, J. 2011. The Java API for XML-Based Web Services (JAX-WS) 2.2 Rev a. Maintenance Release. Accessed on 25.02.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/jaxws-2_2a-mrel4-eval-spec/jaxws-2_2a-mrel4-eval-spec.pdf

Idemichiel. 2014. Java EE Platform Specification. Accessed on 28.7.2015. Retrieved from <https://java.net/projects/javaee-spec/pages/Home>

Mallet, F. 2016. Plugin Library. Accessed on 4.3.2016. Retrieved from <http://docs.sonarqube.org/display/PLUG/Plugin+Library>

Muir, P. 2013. JSR 346 Contexts and Dependency Injection for the Java EE platform. Version 1.1. Final Release. Accessed on 22.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/cdi-1_1-fr-eval-spec/cdi-spec.pdf

Operations. 2014. Article in Kelas website. Accessed on 25.5.2015. Retrieved from <http://www.kela.fi/web/en/operations>

Parkinson, P. 2013. Java Transaction API (JTA). Version 1.2. Accessed on 22.2.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/jta-1_2-mrel2-eval-spec/JTA1.2Specification.pdf

Pericas-Geertsen, S. Potociar, M. 2013. JAX-RS: Java™ API for RESTful Web Services. Version 2.0. Final Release. Accessed on 14.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/jaxrs-2_0_rev_A-mrel-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf

Racodon, D. 2016. Java Plugin. Accessed on 4.3.2016. Retrieved from <http://docs.sonarqube.org/display/PLUG/Java+Plugin>

Shanon, B., Hapner, M. 2003. JSR 151: Java 2 Platform, Enterprise Edition 1.4 (j2ee 1.4) Specification. Accessed on 28.7.2015. Retrieved from <https://jcp.org/en/jsr/detail?id=151>

Vatkina, M. 2013a. JSR 318: Interceptors 1.2. Maintenance Release. Version 1.2. Accessed on 11.12.2015. Retrieved from http://download.oracle.com/otn-pub/jcp/interceptors-1_2-mrel2-eval-spec/interceptor-1-2-mrel-spec.pdf

Vatkina, M. 2013b. JSR 345 Enterprise JavaBeans™, Version 3.2 EJB Core Contracts and Requirements. Accessed on 24.2.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/ejb-3_2-fr-spec/ejb-3_2-core-fr-spec.pdf

Vignola, C. 2013. JSR 352: Batch Application for the Java Platform. Version 1.0. Final Release. Accessed on 23.2.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/batch-1_0-fr-eval-spec/JSR-352-1.0-Final-Release.pdf

Wai Chang, S. Mordani, R. 2013 JSR 340 Java™ Servlet Specification. Version 3.1. Accessed on 15.1.2016. Retrieved from http://download.oracle.com/otn-pub/jcp/servlet-3_1-fr-eval-spec/servlet-3_1-final.pdf

Appendices

Appendix A: CWSS submetrics

Group	Name	Summary
Base Finding	Technical Impact (TI)	The potential result that can be produced by the weakness, assuming that the weakness can be successfully reached and exploited.
Base Finding	Acquired Privilege (AP)	The type of privileges that are obtained by an attacker who can successfully exploit the weakness.
Base Finding	Acquired Privilege Layer (AL)	The operational layer to which the attacker gains privileges by successfully exploiting the weakness.
Base Finding	Internal Control Effectiveness (IC)	The ability of the control to render the weakness that can be exploited by an attacker.
Base Finding	Finding Confidence (FC)	The confidence that the reported issue is a weakness that can be utilized by an attacker.
Attack Surface	Required Privilege (RP)	The type of privileges that an attacker must already have in order to reach the code/functionality that contains the weakness.
Attack Surface	Required Privilege Layer (RL)	The operational layer to which the attacker must have privileges in order to attempt to attack the weakness.
Attack Surface	Access Vector (AV)	The channel through which an attacker must communicate to reach the code or functionality that contains the weakness.
Attack Surface	Authentication Strength (AS)	The strength of the authentication routine that protects the code/functionality that contains the weakness.
Attack Surface	Level of Interaction (IN)	The actions that are required by the human victim(s) to enable a successful attack to take place.
Attack Surface	Deployment Scope (SC)	Whether the weakness is present in all deployable instances of the software, or if it is limited to a subset of platforms and/or configurations.
Environmental	Business Impact (BI)	The potential impact to the business or mission if the weakness can be successfully exploited.
Environmental	Likelihood of Discovery (DI)	The likelihood that an attacker can discover the weakness.

Enviromental	Likelihood of Exploit (EX)	The likelihood that, if the weakness is discovered, an attacker with the required privileges/authentication/access would be able to successfully exploit it.
Enviromental	External Control Effectiveness (EC)	The capability of controls or mitigations outside of the software that may render the weakness more difficult for an attacker to reach and/or trigger.
Enviromental	Prevalence (P)	How frequently this type of weakness appears in software.

Appendix B: CVSS metric vectors

Group name	Metric name	Possible value	Mandatory
Base	Attact Vector, AV	[N,A,L,P]	True
Base	Attack Complexity, AC	[L,H]	True
Base	Priveleges required, PR	[N,L,H]	True
Base	User Interaction, UI	[N,R]	True
Base	Scope, S	[U,C]	True
Base	Confidentiality, C	[H,L,N]	True
Base	Integrity, I	[H,L,N]	True
Base	Availability, A	[H,L,N]	True
Temporal	Exploit code maturity, E	[X,H,F,P,U]	False
Temporal	Remediation level, RL	[X,U,W,T,O]	False
Temporal	Report confidence, RC	[X,C,R,U]	False
Environmental	Confidentiality req., CR	[X,H,M,L]	False
Environmental	Integrity req., IR	[X,H,M,L]	False
Environmental	Availibility req., AR	[X,H,M,L]	False
Environmental	Modified attack vector, MAV	[X,N,A,L,P]	False
Environmental	Modified attack complexity, MAC	[X,L,H]	False
Environmental	Modified privileges required, MPR	[X,N,L,H]	False
Environmental	Modified user interaction, MUJ	[X,N,R]	False
Environmental	Modified scope, MS	[X,U,C]	False
Environmental	Modified confidentiality, MC	[X,N,L,H]	False
Environmental	Modified integrity, MI	[X,N,L,H]	False
Environmental	Modified availability, MA	[X,N,L,H]	False

Appendix C: CVSS score formulas

Base metric group

Base score

If (Impact sub score \leq 0) 0 else,
 Scope Unchanged $\lceil 4 \times \text{Round up}(\text{Minimum}[(\text{Impact} + \text{Exploitability}), 10]) \rceil$
 Scope Changed $\lceil 1.08 \times (\text{Impact} + \text{Exploitability}) \rceil$

Impact sub score (ISC)

Scope Unchanged $6.42 \times \text{ISCBASE}$
 Scope Changed $7.52 \times [\text{ISCBASE} - 0.029] - 3.25 \times [\text{ISCBASE} - 0.02]^{15}$
 $\text{ISCBASE} = 1 - [(1 - \text{ImpactConf}) \times (1 - \text{ImpactInteg}) \times (1 - \text{ImpactAvail})]$

Exploitability sub score

$8.22 \times \text{AttackVector} \times \text{AttackComplexity} \times \text{PrivilegeRequired} \times \text{UserInteraction}$

Temporal metric group

Temporal score

$\text{Round up}(\text{BaseScore} \times \text{ExploitCodeMaturity} \times \text{RemediationLevel} \times \text{ReportConfidence})$

Environmental metric group

Environmental score

If (Modified Impact Sub score \leq 0) 0 else,
 If Modified Scope Unchanged $\text{Round up}(\text{Round up}(\text{Minimum}[(\text{M.Impact} + \text{M.Exploitability}), 10]) \times \text{Exploit Code Maturity} \times \text{Remediation Level} \times \text{Report Confidence})$
 If Modified Scope Changed $\text{Round up}(\text{Round up}(\text{Minimum}[(1.08 \times (\text{M.Impact} + \text{M.Exploitability}), 10]) \times \text{Exploit Code Maturity} \times \text{Remediation Level} \times \text{Report Confidence}))$

Modified impact sub score

If Modified Scope Unchanged $6.42 \times [\text{ISCMODIFIED}]$
 If Modified Scope Changed $7.52 \times [\text{ISCMODIFIED} - 0.029] - 3.25 \times [\text{ISCMODIFIED} - 0.02]^{15}$
 $\text{ISCMODIFIED} = \text{Minimum}[[1 - (1 - \text{M.IConf} \times \text{CR}) \times (1 - \text{M.IInteg} \times \text{IR}) \times (1 - \text{M.IAvail} \times \text{AR})], 0.915]$

Modified exploitability sub score

$8.22 \times \text{M.AttackVector} \times \text{M.AttackComplexity} \times \text{M.PrivilegeRequired} \times \text{M.UserInteraction}$

Appendix D: Identified weaknesses in Java EE technologies and their implementation references

Identifier	Name	Description	Implementation reference
MTW-1	Do not use Java SE concurrency API in Java EE application	Container is not aware threads started through Java SE concurrency API and they can cause race conditions	SECURITY_WEAKNESS: weakness_4:
MTW-2	DataSourceDefinition have password defined	The specification encourages not to add password to DataSourceDefinition annotation	SECURITY_WEAKNESS: weakness_8
MTW-3	Services access have not been limited	Service does not declare RunAs, RolesAllowed, PermitAll, DenyAll or DeclaredRoles annotations to restrict access to service	Application does not restrict any public interfaces
MTW-4	JAX-RS annotated properties should not be written other lifecycle phase than creation	JAX-RS annotated resource properties should not be written other life cycle phase than creation because it can cause errors in concurrency	Not implemented
MTW-5	Non-public methods should not be annotated with @Path annotation	Non-public methods with @Path annotation can not be accessed outside of application and therefore are unnecessary annotations	SECURITY_WEAKNESS: weakness_12
MTW-6	@PATH annotation could use regular expression to white list accepted paths	By using regular expression services possible access paths could be limited and therefore attack surface would be smaller	SECURITY_WEAKNESS: weakness_2
MTW-7	JAX-RS service should use bean validation to validate methods parameters	All input to service should be handled as insecure to increase security.	Application does not validate any input
MTW-8	JSQL injection by misuse of createQuery method	If JPQL-query is concatenated from input parameters the query is not parsed by JPA	SECURITY_WEAKNESS: weakness_1
MTW-9	Using container in construction of servlet	Servlets are initialized through init-method and are not active in container before this	Not implemented
MTW-10	Servlets service method is marked as synchronized	Performance can be lost if servlets service methods is marked as synchronized	SECURITY_WEAKNESS: weakness_9

MTW-11	Request objects other methods than startAsync and completed should not be accessed multiple threads	Request objects startAsync and complete methods are only threads safe methods in class. Accessing other methods from multiple threads can cause concurrency problems	Not implemented
MTW-12	Response objects other methods than startAsync and completed should not be accessed multiple threads	Response objects startAsync and complete methods are only threads safe methods in class. Accessing other methods from multiple threads can cause concurrency problems	Not implemented
MTW-13	Servlets getResource and getResourceAsStream methods should use only for static resources	If dynamic resources, like jsp pages, are acquired through getResource or getResourceAsStream methods they will not be processed by servlet resources	Not implemented
MTW-14	Servlet should use sendError method instead of throwing exception to indicate error	Servlets sendError method adds appropriate header information to response	SECURITY_WEAKNESS: weakness_11
MTW-15	Servlets session timeout should not be infinite	If servlets session timeout is set to zero , session will not be timeouted and it can reserve unnecessary server resources.	SECURITY_WEAKNESS: weakness_10
MTW-16	ConstraintValidator should not store validated value to its state	ConstrainValidator instances can be reused and therefore it should not store validated value to its state	SECURITY_WEAKNESS: weakness_21
MTW-17	Constraint is defined to unsupported type	Constraints that are bound to unsupported type will cause UnexpectedTypeException to raise	Not implemented because IDE gives compiler error when applying wrong type constraint
MTW-18	Components value should not be accessed through component tree	JavaServer Faces components value should not be obtained through component tree because it is not necessary been validated yet.	SECURITY_WEAKNESS: weakness_25
MTW-19	User managed transaction associated with current thread should be mark as completed	Developer starts transaction with commit method invocation but commit, rollback or setRollback method is never called.	SECURITY_WEAKNESS: weakness_9
MTW-20	Stateless session bean should not store client state to beans member variables	Container can reuse stateless session bean instances between clients so clients confidential data can be exposed to other clients	SECURITY_WEAKNESS: weakness_5

MTW-21	Storing client state to singleton session bean can cause lose of confidentiality	Same singleton session bean instance is shared between all clients, therefore it can expose clients confidential data to other clients	SECURITY_WEAKNESS: weakness_3
MTW-22	BindingProvider should not be used to hard code username and password	Hard coded passwords and usernames are harder and slower to change when needed.	SECURITY_WEAKNESS: weakness_7
MTW-23	@WebService annotations should define namespace	Missing namespace can cause conflicts in services published in same endpoint address.	Not implemented because application server failed obtain reference to web service missing namespace
OWASP-A1	A1 - Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query (OWASP Top 10 2013, 6).	SECURITY_WEAKNESS: weakness_1
OWASP-A2	A2 - Broken Authentication and Session Management	Application to compromise passwords, keys, or session tokens, or have other implementation flaws so attacker can assume other user's identities (OWASP Top 10 2013, 6).	SECURITY_WEAKNESS: weakness-20
OWASP-A3	A3 - Cross-site scripting	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping (OWASP Top 10 2013, 6).	All applications public interfaces are missing validation
OWASP-A4	A4 - Insecure direct object references	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key (OWASP Top 10 2013, 6).	Every class in io.vksn.summons.entity package exposes atleast database key with direct reference
OWASP-A6	A6 - Sensitive data exposure	Web application do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data. (OWASP Top 10 2013, 6)	Application does not protect any data

OWASP-A7	A7 - Missing function level access controll	Web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. (OWASP Top 10 2013, 6)	None of public interface functions limits access rights
OWASP-A8	A8 - Cross-site request forgery	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application (OWASP Top 10 2013, 6).	Applications servlet does not implement any CSRF protection mechanisms.
OWASP-A10	A10 - Unvalidated redirects and forwards	Web applications redirect or forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages. (OWASP Top 10 2013, 6)	SECURITY_WEAKNESS: weakness_22
CWE-536	Information Exposure Through Servlet Runtime Error Message	A servlet error message indicates that application does not handle all errors correctly (Martin, Coley, Kenderdine and Piper 2015, 879).	SECURITY_WEAKNESS: weakness_11
CWE-574	EJB Bad Practices: Use of Synchronization Primitives.	Application violates EJB specification by using thread synchronization primitives. (Martin, Coley, Kenderdine and Piper 2015, 916 - 917).	SECURITY_WEAKNESS: weakness_26
CWE-575	EJB Bad Practices: Use of AWT Swing	EJB session bean uses AWT or Swing classes in implementation (Martin, Coley, Kenderdine and Piper 2015, 917 - 919)	Not implemented because Swing and AWT are outdated technologies
CWE-576	EJB Bad Practices: Use of Java I/O	Application uses classes from java.io package and will not behave consistently between EJB containers (Martin, Coley, Kenderdine and Piper 2015, 919 - 920).	SECURITY_WEAKNESS: weakness-23
CWE-577	EJB Bad Practices: Use of Sockets	Application implements Socket server as EJB which conflicts basic function of the EJB (Martin, Coley, Kenderdine and Piper 2015, 920 - 922).	Not implemented

CWE-578	EJB Bad Practices: Use of Class Loader	Application creates or obtains current class loader from EJB which can compromise containers security (Martin, Coley, Kenderdine and Piper 2015, 922 - 923).	SECURITY_WEAKNESS: weakness-24
CWE-579	J2EE Bad Practices: Non-serializable Object Stored in Session	The application stores a non-serializable object as an HttpSession attribute which cause that session cannot be replicated (Martin, Coley, Kenderdine and Piper 2015, 924).	SECURITY_WEAKNESS: weakness-16
CWE-594	J2EE Framework: Saving Unserializable Objects to Disk	Application uses in the J2EE container non-serializable classes which can cause application to crash if they are tried to serialize to disk (Martin, Coley, Kenderdine and Piper 2015,)	SECURITY_WEAKNESS: weakness-15
CWE-598	Information Exposure Through Query Strings in GET Request	Application uses the GET method to process request that contains sensitive information like social security number. These URLs can be exposed later through the browser's history. (Martin, Coley, Kenderdine and Piper 2015, 944).	SECURITY_WEAKNESS: weakness-14
CWE-600	Uncaught Exception in Servlet	Application leaks technical exceptions from public interfaces like servlets that can reveal sensitive debugging information (Martin, Coley, Kenderdine and Piper 2015, 946).	SECURITY_WEAKNESS: weakness-13
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')	Application accepts user-controlled input as target of external link (Martin, Coley, Kenderdine and Piper 2015, 946 - 950).	SECURITY_WEAKNESS: weakness-22
CWE-614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute	Application sets cookie with sensitive data without calling setSecure(true) method (Martin, Coley, Kenderdine and Piper 2015, 966 - 967).	SECURITY_WEAKNESS: weakness-17
CWE-615	Information Exposure Through Comments	Comments that are in view layer expose applications structure or known bugs (Martin, Coley, Kenderdine and Piper 2015, 967 - 968).	SECURITY_WEAKNESS: weakness-18, SECURITY_WEAKNESS: weakness-19

Appendix E: Java plugins security rules profile

```

<?xml version='1.0' encoding='UTF-8'?>
<profile>
  <name>Java-security</name>
  <language>java</language>
  <rules>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>AssignmentInSubExpressionCheck</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>CallToDeprecatedMethod</key>
      <priority>MINOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>ClassVariableVisibilityCheck</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>ObjectFinalizeCheck</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>ObjectFinalizeOverridenCallsSuperFinalizeCheck</key>
      <priority>BLOCKER</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S00112</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S1143</key>
      <priority>BLOCKER</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S1145</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S1147</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S1148</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>squid</repositoryKey>
      <key>S1174</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
  </rules>
</profile>

```

```

<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1181</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1182</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1193</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1194</key>
  <priority>MINOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1206</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1217</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S128</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1313</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1444</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1696</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1698</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1724</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1850</key>
  <priority>MAJOR</priority>

```



```

        <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1854</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1872</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1873</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1948</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1989</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2039</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2068</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2070</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2076</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2077</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2078</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2089</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>

```

```

        <key>S2092</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2095</key>
    <priority>BLOCKER</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2096</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2142</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2151</key>
    <priority>BLOCKER</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2184</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2221</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2222</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2225</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2245</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2250</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2254</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2257</key>
    <priority>BLOCKER</priority>
    <parameters />
</rule>

```

```

<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2258</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2259</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2276</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2277</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2278</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2384</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2386</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2441</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2583</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2653</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2658</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2976</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S3066</key>
  <priority>CRITICAL</priority>

```

```

        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3318</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3355</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3369</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3374</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S864</key>
        <priority>MAJOR</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S888</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>StringEqualityComparisonCheck</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
</rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>SwitchLastCaseIsDefaultCheck</key>
        <priority>MAJOR</priority>
        <parameters />
    </rule>
</rules>
</profile>

```

Appendix F: Security rules profile from multiple plugins

```

<?xml version='1.0' encoding='UTF-8'?>
<profile>
  <name>extended-security</name>
  <language>java</language>
  <rules>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>DMI_CONSTANT_DB_PASSWORD</key>
      <priority>BLOCKER</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>HRS_REQUEST_PARAMETER_TO_HTTP_HEADER</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>PT_ABSOLUTE_PATH_TRAVERSAL</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>PT_RELATIVE_PATH_TRAVERSAL</key>
      <priority>MAJOR</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>SQL_PREPARED_STATEMENT_GENERATED_FROM_NONCONSTANT_STRING</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>XSS_REQUEST_PARAMETER_TO_JSP_WRITER</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>XSS_REQUEST_PARAMETER_TO_SEND_ERROR</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findbugs</repositoryKey>
      <key>XSS_REQUEST_PARAMETER_TO_SERVLET_WRITER</key>
      <priority>CRITICAL</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findseccbugs</repositoryKey>
      <key>ANDROID_BROADCAST</key>
      <priority>INFO</priority>
      <parameters />
    </rule>
    <rule>
      <repositoryKey>findseccbugs</repositoryKey>
      <key>ANDROID_EXTERNAL_FILE_ACCESS</key>

```

```

        <priority>MAJOR</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>ANDROID_GEOLOCATION</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>ANDROID_WEB_VIEW_JAVASCRIPT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>ANDROID_WEB_VIEW_JAVASCRIPT_INTERFACE</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>ANDROID_WORLD_WRITABLE</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>BAD_HEXA_CONVERSION</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>BLOWFISH_KEY_SIZE</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>CIPHER_INTEGRITY</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>COMMAND_INJECTION</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>COOKIE_USAGE</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>CUSTOM_MESSAGE_DIGEST</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>DES_USAGE</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>ECB_MODE</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
</rule>

```

```
        <repositoryKey>findsecbugs</repositoryKey>
        <key>ESAPI_ENCRYPTOR</key>
        <priority>INFO</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>FILE_UPLOAD_FILENAME</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>HARD_CODE_KEY</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>HARD_CODE_PASSWORD</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>HAZELCAST_SYMMETRIC_ENCRYPTION</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>JAXRS_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>JAXWS_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>LDAP_INJECTION</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>NULL_CIPHER</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>PADDING_ORACLE</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>PATH_TRAVERSAL_IN</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>PATH_TRAVERSAL_OUT</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>PREDICTABLE_RANDOM</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
```

```
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>REDOS</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>RSA_KEY_SIZE</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>RSA_NO_PADDING</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SCRIPT_ENGINE_INJECTION</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_CONTENT_TYPE</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_HEADER</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_HEADER_REFERER</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_HEADER_USER_AGENT</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_PARAMETER</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_QUERY_STRING</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_SERVER_NAME</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SERVLET_SESSION_ID</key>
  <priority>INFO</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>findsecbugs</repositoryKey>
  <key>SPEL_INJECTION</key>
```



```

        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>SPRING_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>SQL_INJECTION_HIBERNATE</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>SQL_INJECTION_JDO</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>SQL_INJECTION_JPA</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>STATIC_IV</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>STRUTS1_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>STRUTS2_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>STRUTS_FORM_VALIDATION</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>TAPESTRY_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>UNENCRYPTED_SOCKET</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>UNVALIDATED_REDIRECT</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsecbugs</repositoryKey>
    <key>WEAK_FILENAMEUTILS</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>

```

```

        <repositoryKey>findsechugs</repositoryKey>
        <key>WEAK_MESSAGE_DIGEST</key>
        <priority>MAJOR</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>WEAK_TRUST_MANAGER</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>WICKET_ENDPOINT</key>
    <priority>INFO</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XML_DECODER</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XPATH_INJECTION</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XSS_JSP_PRINT</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XSS_REQUEST_WRAPPER</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XSS_SERVLET</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XXE_DOCUMENT</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XXE_SAXPARSER</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>findsechugs</repositoryKey>
    <key>XXE_XMLREADER</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>pmd</repositoryKey>
    <key>ArrayIsStoredDirectly</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>pmd</repositoryKey>
    <key>MethodReturnsInternalArray</key>
    <priority>CRITICAL</priority>
    <parameters />

```

```

</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>AssignmentInSubExpressionCheck</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>CallToDeprecatedMethod</key>
  <priority>MINOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>ClassVariableVisibilityCheck</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>ObjectFinalizeCheck</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>ObjectFinalizeOverriddenCallsSuperFinalizeCheck</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S00112</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1143</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1145</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1147</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1148</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1174</key>
  <priority>MAJOR</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1181</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S1182</key>

```

```

        <priority>MAJOR</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1193</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1194</key>
    <priority>MINOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1206</key>
    <priority>BLOCKER</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1217</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S128</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1313</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1444</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1696</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1698</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1724</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1850</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S1854</key>
    <priority>MAJOR</priority>
    <parameters />
</rule>
<rule>

```

```

        <repositoryKey>squid</repositoryKey>
        <key>S1872</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S1873</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S1948</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S1989</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2039</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2068</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2070</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2076</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2077</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2078</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2089</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2092</key>
        <priority>CRITICAL</priority>
        <parameters />
</rule>
<rule>
        <repositoryKey>squid</repositoryKey>
        <key>S2095</key>
        <priority>BLOCKER</priority>
        <parameters />

```

```

</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2096</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2142</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2151</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2184</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2221</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2222</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2225</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2245</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2250</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2254</key>
  <priority>CRITICAL</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2257</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2258</key>
  <priority>BLOCKER</priority>
  <parameters />
</rule>
<rule>
  <repositoryKey>squid</repositoryKey>
  <key>S2259</key>

```

```

        <priority>BLOCKER</priority>
        <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2276</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2277</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2278</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2384</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2386</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2441</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2583</key>
    <priority>BLOCKER</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2653</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2658</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S2976</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S3066</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>
    <repositoryKey>squid</repositoryKey>
    <key>S3318</key>
    <priority>CRITICAL</priority>
    <parameters />
</rule>
<rule>

```

```

        <repositoryKey>squid</repositoryKey>
        <key>S3355</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3369</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S3374</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S864</key>
        <priority>MAJOR</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>S888</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>StringEqualityComparisonCheck</key>
        <priority>CRITICAL</priority>
        <parameters />
    </rule>
    <rule>
        <repositoryKey>squid</repositoryKey>
        <key>SwitchLastCaseIsDefaultCheck</key>
        <priority>MAJOR</priority>
        <parameters />
    </rule>
</rules>
</profile>

```

Appendix G: Security issues with Java plugin

Key	Rule	Severity	Module	File	Line	Tags	message
JDS-1	squid:S3369	CRITICAL	summons-ui	WEB-INF/web.xml		cwe, jee, owasp-a7, security, webspHERE	Add "security-constraint" elements to this descriptor.
JDS-2	squid:S3355	CRITICAL	summons-ui	WEB-INF/web.xml		injection, owasp-a1, security	Add a validation filter to this "web.xml".
JDS-3	squid:S1948	CRITICAL	summons-ui	io/vksn/summons/ui/beans/CreateEventBean.java	38	bug, cwe, serialization	Make "service" transient or serializable.
JDS-4	squid:S1948	CRITICAL	summons-ui	io/vksn/summons/ui/beans/CreateEventBean.java	41	bug, cwe, serialization	Make "event" transient or serializable.
JDS-5	squid:S2092	CRITICAL	summons-ui	io/vksn/summons/ui/beans/ParticipateBean.java	100	cwe, owasp-a2, owasp-a6, security	Add the "secure" attribute to this cookie

JDS-6	squid:S2441	CRITICAL	summons-ui	io/vksn/summons/ui/beans/ParticipateBean.java	93	bug, cwe	Make "Invitation" serializable or don't store it in the session.
JDS-7	squid:S2221	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	37	cwe, error-handling, security	Catch a list of specific exception subtypes instead.
JDS-8	squid:S1948	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	25	bug, cwe, serialization	Make "service" transient or serializable.
JDS-9	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	39	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block.
JDS-10	squid:S1948	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	22	bug, cwe, serialization	Make "service" transient or serializable.
JDS-11	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	31	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "parseLong".
JDS-12	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	34	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JDS-13	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	34	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JDS-14	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	36	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JDS-15	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	36	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JDS-16	squid:S1989	CRITICAL	summons-ui	io/vksn/summons/ui/servlet/RedirectServlet.java	23	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "sendRedirect".
JDS-17	squid:S1148	CRITICAL	summons-service	io/vksn/summons/ejb/AuditLogger.java	38	error-handling, security	Use a logger to log this exception.
JDS-18	squid:S2068	CRITICAL	summons-service	io/vksn/summons/repository/ListEventsRepository.java	22	cwe, owasp-a2, sanst-25-toporous, security	Remove this hard-coded password.

JDS-19	squid:S2221	CRITICAL	summons-service	io/vksn/summons/repository/ListEventsRepository.java	44	cwe, error-handling, security	Catch a list of specific exception subtypes instead.
JDS-20	squid:S00112	CRITICAL	summons-service	io/vksn/summons/repository/ListEventsRepository.java	46	cwe, error-handling, security	Define and throw a dedicated exception instead of using a generic one.
JDS-21	squid:S2384	CRITICAL	summons-service	io/vksn/summons/rest/model/SeatingPlan.java	21	cert, cwe, security, unpredictable	Return a copy of "tables".
JDS-22	squid:S2384	CRITICAL	summons-service	java/io/vksn/summons/rest/model/SeatingPlan.java	24	cert, cwe, security, unpredictable	Store a copy of "tables".
JDS-23	squid:S2384	CRITICAL	summons-api	io/vksn/summons/entity/Table.java	83	cert, cwe, security, unpredictable	Return a copy of "chairs".
JDS-24	squid:S2384	CRITICAL	summons-api	io/vksn/summons/entity/Table.java	86	cert, cwe, security, unpredictable	Store a copy of "chairs".

Appendix H: Security issues with multiple plugins

Key	Rule	Severity	Module	File	Line	Tags	Message
JES-1	squid:S0012	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/repository/ListEventsRepository.java	46	cwe, error-handling, security	Define and throw a dedicated exception instead of using a generic one.
JES-2	squid:S2068	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/repository/ListEventsRepository.java	22	cwe, owasp-a2, sants-top25-porous, security	Remove this hard-coded password.
JES-3	squid:S2221	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/repository/ListEventsRepository.java	44	cwe, error-handling, security	Catch a list of specific exception subtypes instead.
JES-4	squid:S1148	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/ejb/AuditLogger.java	38	error-handling, security	Use a logger to log this exception.
JES-5	findsecbugs:SQL_INJECTION_JPA	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/repository/SummonsRepository.java	81	cwe, injection, owasp-a1, security, wasc	The query is potentially vulnerable SQL/JPQL injection
JES-6	squid:S2384	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/rest/model/SeatingPlan.java	21	cert, cwe, security, unpredictable	Return a copy of "tables".
JES-7	squid:S2384	CRITICAL	io.vksn.summons:summons-service	io/vksn/summons/rest/model/SeatingPlan.java	24	cert, cwe, security, unpredictable	Store a copy of "tables".
JES-8	squid:S1948	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/beans/CreateEventBean.java	38	bug, cwe, serialization	Make "service" transient or serializable.
JES-9	squid:S1948	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/beans/CreateEventBean.java	41	bug, cwe, serialization	Make "event" transient or serializable.
JES-10	squid:S2092	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/beans/ParticipateBean.java	100	cwe, owasp-a2, owasp-a6, security	Add the "secure" attribute to this cookie
JES-11	squid:S2441	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/beans/ParticipateBean.java	93	bug, cwe	Make "Invitation" serializable or don't store it in the session.

JES-12	squid:S2221	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	37	cwe, error-handling, security	Catch a list of specific exception subtypes instead.
JES-13	squid:S1948	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	25	bug, cwe, serialization	Make "service" transient or serializable.
JES-14	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListEventsServlet.java	39	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block.
JES-15	squid:S1948	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	22	bug, cwe, serialization	Make "service" transient or serializable.
JES-16	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	31	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "parseLong".
JES-17	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	34	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JES-18	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	34	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JES-19	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	36	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JES-20	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/ListInvitationsServlet.java	36	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "forward".
JES-21	squid:S1989	CRITICAL	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/RedirectServlet.java	23	cert, cwe, error-handling, owasp-a6, security	Add a "try/catch" block for "sendRedirect".
JES-22	findbugs:HS_REQUEST_PARAMETER_TO_HTTP_HEADER	MAJOR	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/RedirectServlet.java	23	cwe, owasp-a3	HTTP parameter directly written to HTTP header output in io.vksn.summons.ui.servlet.RedirectServlet.doGet(HttpServletRequest Request, HttpServletResponse)

JES-23	finds ecbugs:UNVALIDATED_REDIRECT	MAJOR	io.vksn.summons:summons-ui	io/vksn/summons/ui/servlet/RedirectServlet.java	23	cwe, security, wasc	Unvalidated Redirect
JES-24	squid:S3369	CRITICAL	io.vksn.summons:summons-ui	WEB-INF/web.xml		cwe, jee, owasp-a7, security, webspere	Add "security-constraint" elements to this descriptor.
JES-25	squid:S3355	CRITICAL	io.vksn.summons:summons-ui	WEB-INF/web.xml		injection, owasp-a1, security	Add a validation filter to this "web.xml".
JES-26	squid:S2384	CRITICAL	io.vksn.summons:summons-api	io/vksn/summons/entity/Table.java	83	cert, cwe, security, unpredictable	Return a copy of "chairs".
JES-27	squid:S2384	CRITICAL	io.vksn.summons:summons-api	io/vksn/summons/entity/Table.java	86	cert, cwe, security, unpredictable	Store a copy of "chairs".