National College of Ireland

Security Audit for web application

MSc Research Project
Cloud Computing

Ramyabharathi Duraichamy
x15044424

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet – 2015/2016
School of Computing

Student Name:	Ramyabharathi Duraichamy
Student ID:	x15044424
Programme:	Cloud Computing
Year:	2016
Module:	MSc Research Project
Lecturer:	Manuel Tova-Izquierdo
Submission Due Date:	15/09/2017
Project Title:	Security Audit for web application
Word Count:	5971

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	Ramyabharathi Duraichamy
Date:	12th September 2017

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Security Audit for web application

Ramyabharathi Duraichamy
x15044424
MSc Research Project in Cloud Computing

12th September 2017

Abstract

Though security has become a major concern of a user using a cloud application, it is the responsibility of the cloud provider to secure the collected users data. The most targeted layer of a cloud service is the application layer containing too many loop holes through which an attacker can steal the users data. The service provided by each SaaS provider may differ from each one, but the threat level of all those cloud services are same. So the pressure of tightening the security of the application is with the cloud service provider. The provider has to analyse the possible ways his cloud service can be attacked, this analysis will help them to frame the preventive measures. The problem in this is not framing those preventive measures or implementing them, but identifying all the methods or syntax present in the source code which has the possibility to expose taint content and verifying whether the needed measures are implemented before updating them in live. The main aim of this research is to identify those codes and verify whether necessary measures are implemented to prevent the application from any attacks.

1 Introduction

Cloud providers these days gives equal priority to security like how they give for their application development and infrastructure. Security has become a part for all the three layers(IaaS, PaaS and SaaS) of cloud computing, the measures for each of them differs from the other. Among the three the probability of getting attacked at the application layer is high compared with the others. Kaufman (2009) says that the loopholes in getting into an application layer is unpredictable, because a single simple thing can lead to serious security breach. Even the data provided by the user may also have vulnerable content, so the providers must validate all the parameter values provided by the user. Securing user's data is the most important thing of every cloud service provider, once the users feel that their data is no more safe with a cloud then it is the end of that cloud service. Zissis and Lekkas (2012) states that protecting an user's account by not letting to any data loss is the biggest task of a service provider.

Subashini and Kavitha (2011) has listed the classification of security vulnerabilities which are needed to be considered while developing the cloud application. The vulnerabilities pointed in their list are of different categories through which an attacker can easily get an access to the user's account. Subashini and Kavitha (2011) states the differences in handling security in both on-premise and SaaS, in SaaS the control over the sensitive

data is only with the service provider and it is completely opposite in the on-premise, in on-premise everything is controlled based on the enterprise policy. Both Zissis and Lekkas (2012) and Subashini and Kavitha (2011) points out how important is securing user's data, every attacker's aim is to steal the data of users. This research is also concerned about the data security and mainly aim is to make the work easy to providers in fixing them.

The cloud providers are identifying all possible ways through which their Cloud application can be attacked and they are also framing all the preventive measures. But Zissis and Lekkas (2012) says that identifying and framing the measures to prevent those threats is not the end, it is the responsibility of the provider to bring them up to the development team who needs to be aware of those threats and the ways to avoid them. Informing the developers about the security measures framed is not the end, it has to be followed properly by the developers. The research problem - **'Can all the un-trusted methods (using taint data which causes security attacks) used in the source files be identified and listed for a specific project?'** - it is really hard to check whether each and every file before updating the latest changes to the live. Security Audit helps in doing that job, it scans all the source files and identifies those un-trusted methods or variables or objects in them. And this Audit is developed only for "**Java** based applications, the scan is done only for Java byte code, JSP and js files.

This research is organized into 5 sections, firstly it starts with the section 2 **Related Work** which shows the need of this research, it gives the rough idea as how important is the cloud application security. It comprises with two major clusters preventing the application from **XSS**(Cross-site scripting) and the **Session and cookie manipulation**. All the papers mentioned in this section were really useful in developing the whole idea of this research.

The section 3 **Methodology** which beefily discusses how this research project is constructed, and the research is explained based on some classification of methods which will give an idea like how the rest of the process are done. The important thing this section has is the details about the data which are used during the evaluation of this research project. This section is equally important like the previous section 2, because this section actually defines the research project.

The next section 4 **Design** will give the overall structure of Security audit, this comprises of the the data model and how the entire process of this research project will be done. Some class diagram is also used in this for better understanding about how exactly it works form the beginning till the generation of the expected result of the process.

The next section 5 is **Implementation** the main part which explains how this whole research is developed, how the methodology is molded to produce the expected result. It contains the detailed information of the design part and how each part mentioned in the design in developed and how the result is generated. It also contains some images of the code pieces to show how certain objects are created to process the security audit.

The last and final important part of the research is this section 6 **Evaluation** showing how the developed project is been tested to check whether the generated results are correct. It not only to check whether all obtained results are correct or not, but also to check whether the project is capable in processing and generating the results in various circumstances. This section points out the source code which is being used to test the security audit.

2 Related Work

In-order to continue a research program it is necessary to identify the related works of the project, the below sections of related work really helped a lot in understanding the concept and also strongly supports the cause of this research project. All related works identified for this research are categorised into different sections based on what type of problem they solved and also for the better understanding of this research work. As mentioned in the previous section this research focuses on the two among the top ten vulnerabilities listed by OWASP, so the major two sections of the related work section will be XSS(Cross-site scripting) 2.1 and Cookie & Session manipulation 2.2.

2.1 Cross-Site scripting (XSS)

XSS is being considered as one of the major vulnerability in SaaS, most of the xss attacks are executed using simple JavaScript code which helps in invoking the browser's JavaScript interpreter. Kirda et al. (2006) and Kieyzun et al. (2009) says that not all xss attacks are executed in the same way, some are caused because of the stored information and some are reflected. This section is further classified based on the type of problem solved in the peppers like 2.1.1 and 2.1.2 contains papers who tried solving the client and server side issues of Reflect XSS attacks, 2.1.3 and 2.1.4 contains papers who tried to solve the client and server side issues of Stored XSS attacks, the 2.1.5 contains the papers who tried to provide a solution for both Stored and Reflected XSS attacks and the last section 2.1.6 has the paper which talks about the importance of having a security framework which would help a lot in prevent the web application form the security attacks.

2.1.1 Client side solution for Reflected XSS attacks

The reflected attacks are caused because of using third party links in the web application leading to serious XSS vulnerability, the below figure 1 shows how reflected attacks are performed. Kirda et al. (2006) in their paper clearly explained about the reflected attacks and focussed on preventing the application from reflected attack. The created a personal firewall called Noxes acting as a web proxy which automatically identify the vulnerability based on the rules defined and helps in providing an increased protecting to the web application. The Noxes tries to identify most of the reflected XSS present the current page.

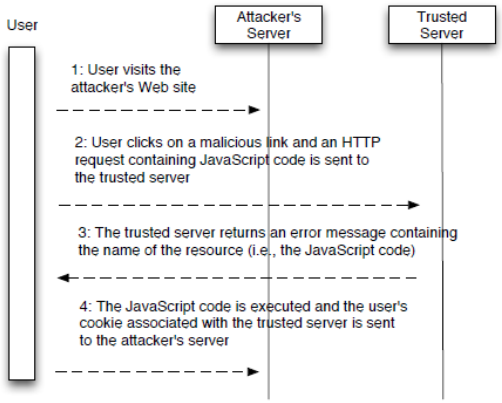

Figure 1: Example of Reflected XSS attack

Bates et al. (2010) believed rather than providing a defence filters in the client side to protect the web application from XSS, they suggest all the developers to include the XSS filter in the web browser. In-order to make the above statement true Bates et al. (2010) created a client side XSS filter called XSSAuditor being placed between the HTML parser and JavaScript interpreter. With the help of URL decoder, char set encoder and HTML entity decode they formed an algorithm using which they transformed all the url request to achieve high level of performance and it is enabled in Google chrome 4. With the help of the decoders the filter decodes all the HTML entities present in the requested page and then with the help of the char set encoder all the data are encoded to avoid the XSS attacks.

Vogt et al. (2007) says that the control of the injected content stays completely with the attacker and also points that if an object contains one taint property, then the object itself is completely taint. Vogt et al. (2007) created a FireFox based web crawler working based on the dynamic data tainting, this crawler tries to identify the sensitive data and with the help of some scripts it dynamically tracks whenever an identified sensitive data is been accessed. The Javascript engine and the DOM tree of the browser has been modified to make the web crawler to track the data which are dependent and also to have a direct control over them.

Meyerovich and Livshits (2010) also talks about the usage of external/third-party links or api or libraries in the web application and says that even though those externals are useful for the development but the possibility of containing vulnerability is equally high. Meyerovich and Livshits (2010) built a solution over Internet Explorer of version 8 called CONSCRIPT which tries to ensure whether the third party included in the hosting page is free from any threat. Though CONSCRIPT is browser based client solution Meyerovich and Livshits (2010) modified the specific browser's javascript interpreter present in the script engine in-order to support the 17 polices(both security and reliability) framed in CONSCRIPT.

2.1.2 Server side solution for Reflected xss attacks

When most of them focused on the preventing the web application from client side Jovanovic et al. (2006) focused in preventing the application from server side stating that the benefit of identifying the vulnerabilities is high in server side. Jovanovic et al. (2006)

created the tool Pixy developed in Java which identifies the vulnerabilities in taint-style based on the flow of the data present in PHP code. Even though it covers most of the PHP's objects it failed to identify the files which uses ïncludeänd all the keywords related to it, not only that but Jovanovic et al. (2006) was not able include array's aliasing relationships which was a negative of Pixy.

2.1.3 Client side solution for Stored xss attacks

The stored xss attacks are invoked by passing input data containing string which is executed when they are rendered in the web page, in this attack the possibility of getting the access over the user's sensitive data is possible, leading to serious problems. Wassermann and Su (2008) says that rendering those stored un trustable data in browser is only possible with the help of the request attributes, they created a list made with the help of W3C recommendation, some source code of FireFox browser and some related tutorials available in online. Wassermann and Su (2008) made a policy using the above list to validate the data whether it is capable of invoking the javascript interpreter. It first decodes all the characters present in data to be rendered in the browser, then it extracts all the tags present in it and then finally it checks whether that data has any characters which can invoke some scripts.

Jim et al. (2007) states that the only way to prevent the web application from stored xss attack is either filter or transform all the content to avoid invoking the scripts, this has been achieved by them by creating the mechanism BEEP using which the web site can be prevented from script injection. The BEEP mechanism works with the help of the 2 framed policies DOM and whitelist file, the DOM here acts as the blacklist file. To make this work Jim et al. (2007) has made some modifications in both browser and application, it reads all the application files rendered in the browser and identifies the un-trustable contents in them.

Ter Louw and Venkatakrishnan (2009) says that a best way to stop the execution of the scripts in web application is by either filtering the content or using some browser collaboration approach. BLUEPRINT a solution created by Ter Louw and Venkatakrishnan (2009) to prevent the web applications from XSS attacks, with the help of BLUEPRINT the web application can understand the contents present in it. The BLUEPRINT is designed to generate a parse tree in the server side based on various vulnerable HTMLs, the vulnerable contents are removed and are structured in the parse tree and that parse tree is passed to the document generator. Ter Louw and Venkatakrishnan (2009) wanted to remove the vulnerabilities present in the major parsers of the browser like HTML, CSS, UI and JavaScript using their solution BLUEPRINT, in the CSS parser part it wanted to stop the influence in it and by disabling the dynamically generated properties to prevent from the attacks caused in CSS. The BLUEPRINT has a client side interpreter using which it decoded all the JavaScript code models to remove the vulnerabilities from the browser parsers.

2.1.4 Sever side solution for Stored xss attacks

According to Wurzinger et al. (2009) to protect a web application from XSS, it necessary to validate all the inputs and outputs as an initial step which prevents more than 60% of the web application. SWAP the web application proxy created by Wurzinger et al. (2009) stands in the middle, before sending the constructed response to the browser it sends them to the javascript detection component. The component is the place where all

the vulnerable scripts are identified, it checks all the values in the response whether there is any malicious content in it and when it ensures that there is no such content exists in the response then it sends it to the browser. The work of SWAP is not to stop or to do any extra work with the script rather it notifies the user regarding the presence of malicious scripts in the response. The only drawback in SWAP is it has some performance issues, because of this it cannot be used for any Web Service doing high performance and this is accepted by Wurzinger et al. (2009) themselves.

2.1.5 Both Stored and Reflected xss attacks

Kieyzun et al. (2009) created two techniques one is to manually create XSS and SQL injections in the web application and another to to identify the malicious code present in the developed source files. They named the second one as Ardilla which identifies SQL Injection, Reflected and Stored XSS vulnerable codes in the developed PHP programming. Kieyzun et al. (2009) created the tool Ardilla using 6 different SQL Injection patterns with the help of various list and 113 patterns for XSS including encode. Ardilla is created to do 4 major jobs like creating input for test cases, an executor to run all the created inputs, an attack generator taking all the taint list for creating attacks and finally a Concrete cum Symbolic Database for executing all SQL related statements present in the PHP programs.

Van Gundy and Chen (2012) the second one to concentrate on both the reflected and stored attacks, they created a tool called Noncespaces. Van Gundy and Chen (2012) created the Noncespaces to achieve their aim of rendering both trusted and non trustable data safely, Noncespaces identifies the differences in both trusted and non trustable data by enabling a web client to prevent the application from XSS threats. Noncespaces dynamically determines the un trustable data using program analysis or using the information flow tracker, this is done with the help of a server which can decide of what kind of approach is needed in identifying the taint data. The Noncespaces has a policy validator to connect both the client and server, before rendering the data to the browser the client validator validates all the document based on its policy.

2.1.6 Importance of frameworks in web application.

While all the above papers concentrated in identifying the XSS present in written code or the input/output data, but Weinberger et al. (2011) explains about the importance of understanding the logics of the web framework used for the development rather than creating a tool like above papers. They made an analysis to prove that why it is needed to understand the logic of how a framework works and what are the thinks needed to be noted regarding security. Weinberger et al. (2011) states that a framework has to provide some basic XSS security to prevent them or it should automatically add the security protection needed for the application and used 14 web application development and 8 other web application for the analysis to know the requirements and the sanitization of the framework used in them. As an outcome of the analysis Weinberger et al. (2011) concluded that most of the frameworks fails in providing the XSS security for the web application, also it lacks in providing what the specific application actually need.

2.2 Cookie & Session Manipulation

Both cookie and session details are the most important part of an application, because these two obviously contains the sensitive data like user name, email address, user id, session id or even password. Once an attacker gets any of them in the right way then he can get access to the user's account, so all the sensitive data which are needed to be stored in the browser cookie or the server's session then it has to be treated special like encoding and encrypting them. Cookie and Session manipulation is equally a great threat to all web application, the below papers are analysis explaining what are to be considered as serious issues and the things to do avoid the.

According to Tappenden and Miller (2009) cookies are useful to store the textual information in the browser which helps in identifying the users of the web application, even though they are useful there are some complexities in it which is being ignored while application testing. The cookies and session details are to be given equal importance as given for the other storage's, Tappenden and Miller (2009) in this paper they focused on 4 types cookies i.e. First-part, Third-party, Sessional cookies and Persistent cookies. Their paper was completely an analysis rather providing any tools to identify they suggested the ways using which the cookie manipulation can be avoided.

Miyazaki (2008) says that the cookies are generally designed to stay in the browser which even lasts for months, so the chance of tracking a customer in online is possible with those stored cookie attributes. Though the cookies becomes on of the key point for an attacker to gain the access of the user's account Miyazaki (2008) researched as how much the customers are really worrying about the privacy policies and the problems in storing the sensitive data in the browser cookie.

Rabinovich (2013) mentions about session and persistent cookies the sub divisions of cookies which are used to save the preferences and some of the sensitive details which are needed to maintain the session in a specific browser. Rabinovich (2013) also mentioned about the same domain cookie policy using which the cookies stored in one domain cannot be used in other domain a suggested option as the sensitive data are stored in the user's trusted browser. An authorization model is being introduced by Rabinovich (2013) to help in sharing the cookies between various DNS domains, which is used to write or read cookies from cross-domain channels(XDC). Rabinovich (2013) says both XDC and HTTP cookies are same in structure wise but XDC cookies can only be accessed through a secure SSL-protected connection and also in transferring the cookies via XDC gives lots of benefits that too specifically in avoiding DNS spoofing.

Even though the Ajax technique plays a major role showing the evolution of web application development, but the need for cookies has not decreased. Tappenden and Miller (2014) after their previous analysis made for cookies they proposed a strategy called Ä̈utomated Cookie Collectionserving as an automation testing tool modifying the collection of cookies of an user which are stored in a browser. The Cookie collection testing is capable of working in web applications and specially between the client and server of a web application

3 Methodology

This section also has equal importance like the 2, this contains the information like how this product will be evaluated, what kind of data are used for it and all other details which are needed to considered to prove that this project is capable of doing the research

idea. As mentioned earlier this section defines the Security Audit based on 4 categories starting from Type(3.1) to Reasoning(3.4).

3.1 Type

It is most important to decide what kind of data is going to be used for the evaluating the research project, sometimes it can be collecting new data (primary type) and sometimes constructing with the available data in all means(secondary type). Security Audit belongs to Secondary type, though the testable source files are constructed with the program codes which are available as open source in the various repositories like GitHup,Bitbucket,etc,. The program codes are selected based on the declared blacklisted and whitelisted configuration objects which are explained in sections 5.3.1 and 5.3.2.

3.2 Objective

All researches aims to identify certain things either the quality like human behaviour or quantity which talks about numbers of the the information or data of the output and sometimes a research can be conducted based on the mixture of both. Security audit is of Quantitative though its main objective is to identify the number of threats in the provided source code and inform the developers about it. The evaluation section 6 shows whether the main objective of security audit is achieved or not.

3.3 Form

A research can be formed based on whether it is going to identify a new problem and provide solution(Exploratory) or developing a solution for an existing problem(Constructive) or testing the feasibility of a provided solution(Empirical) . Security audit provides a solution for an existing problem, so it is a constructive research.

3.4 Reasoning

A research has to state which kind of problem it is addressing whether it is of deductive reasoning(i.e. general problem to specific) or inductive reasoning(i.e. specific problem to general). Security audit does inductive type of reasoning, i.e. it fucus on identifying the general security threats from the java specific source codes.

4 Design

The processed Security Audit is designed to identify the set of predefined invoker and syntax which are formed based on certain criteria and currently it identifies them for the provided java byte code files and jsp files. The security audit notes down all identified threats in a file and stores them in a list, once the scanning is completed the list is processed to generate a pdf formate report. The figure 2 has the overview of the Security Audit, showing how the entire process work which comprises 3 major process as explained from 4.1 Upload Files to 4.2 Scanning extracted files.

4.1 File upload & extraction

The user can upload the files after logging into their project portal, the users are only allowed to upload .zip or .war format files. The security audit scans all the files included in the uploaded file. Once the server gets the request then it extracts the files to the respective folder a The user scan request the scanning for the uploaded file will be sent to server and the server action will start the scanning process and in mean while the user can continue his work.

4.2 Scan Extracted files

This part has the major functionality of Security Audit, this object reads all the files one by one and the process works based on the file specific configuration objects. The objects are inter-linked, these objects checks whether the passed content is taint or not and these objects will be called only for the specific file format. For example currently security audit supports for java byte code and jsp, so if the uploaded compressed file contains any other format files like css or js or asp then the scan object skips the identification process. Two configuration files(blacklisted file and whitelisted object) is created for each file type. Each code line is sent to the blacklist identification and if the line contains some blacklisted invoker or syntax then it again sent to the whitelist object to check whether it needs any be skipped or not. Once the blacklist object confirms then the line is recorded in the list, once the scan is completed then the list is passed to the report generation object where the pdf file is created and stored. The figure 3 shows how the scan object and the configuration object process a single file.

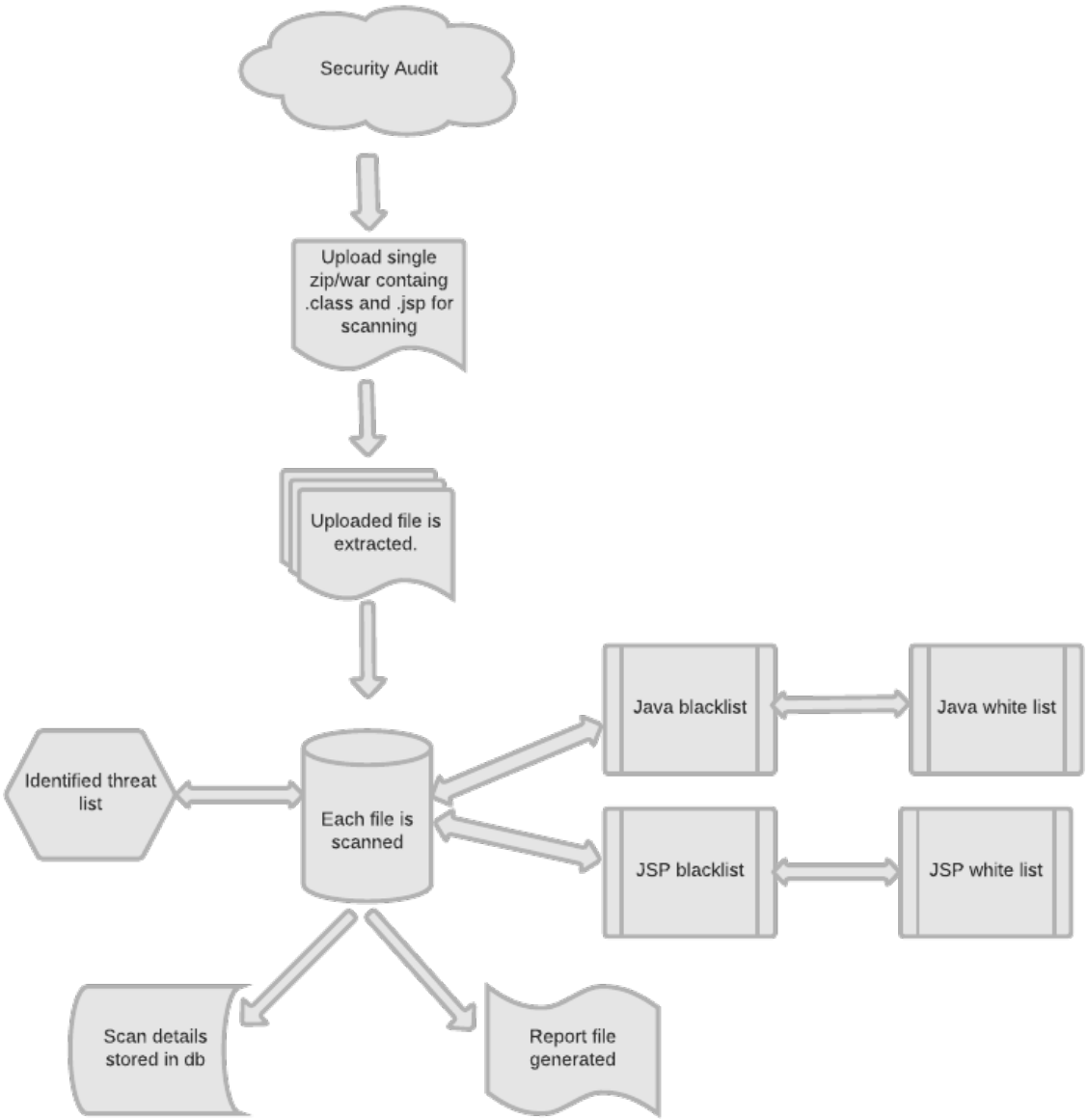

Figure 2: Overview design of security audit

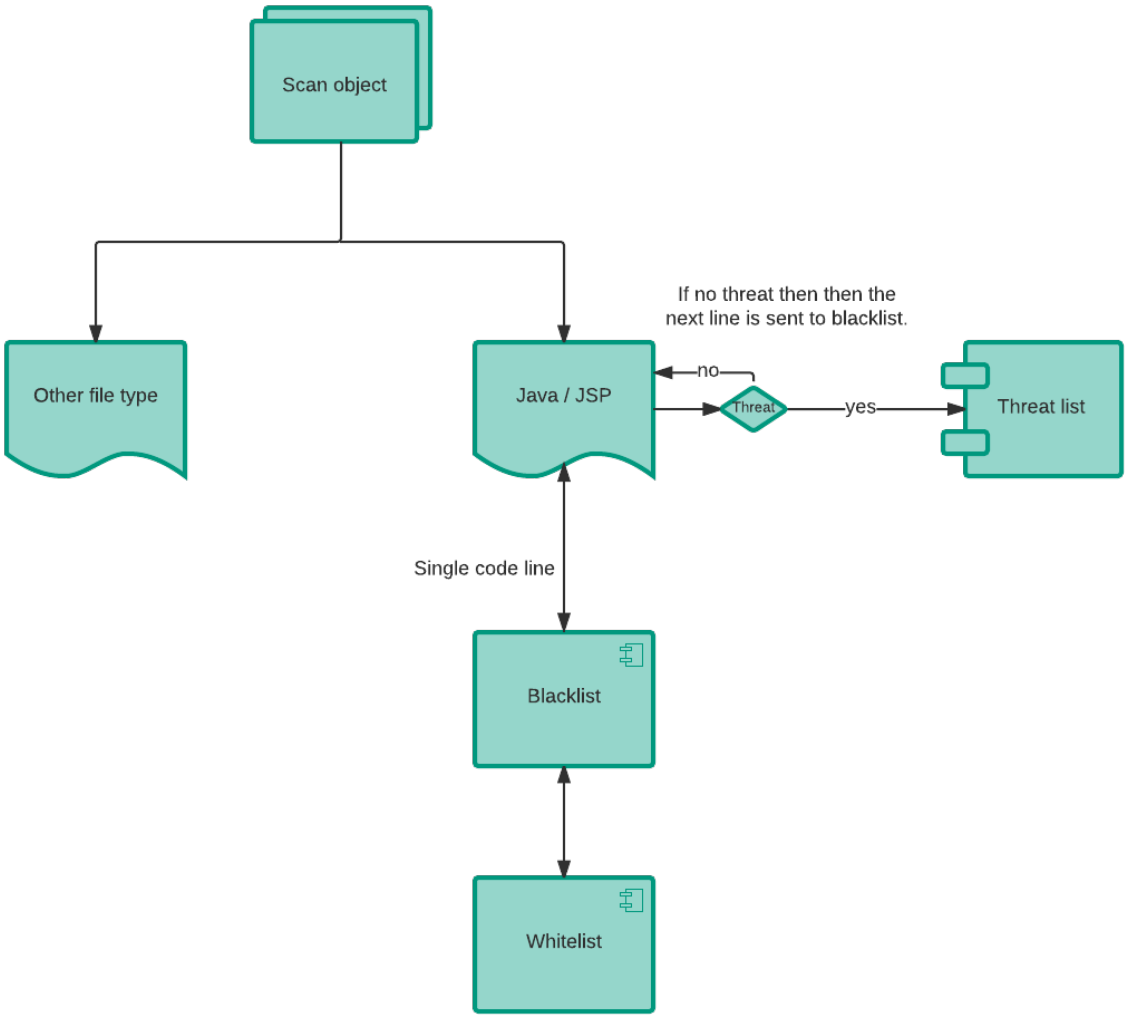

Figure 3: Structure of scan object

5 Implementation

The figure2 show how security audit will process and the major features of it, but there are lot more things implemented in-order to frame it. As mentioned Security audit focuses on the 2 major security threats of a SaaS application XSS and Cookie & Session manipulation. All the implementation of Security audit are mentioned in the order it was developed starting from authentication & authorization(5.1) till how the scanned information are rendered in report and stored in data base (5.4).

5.1 Authentication & Authorization

Though Security audit is a web application it is necessary to implement authentication and authorization to make the application secure by allowing only the specific users to use the specific project. The developers are requested to signup to Security audit in-order to perform the audit process, the user credentials are validated, safely encoded using the AES algorithm and then stored in the data base. Once the user is successfully signed

up then he will be redirected to a project setup page, through which he has to create a project by providing the basic information and the users who are in that project. The user will be redirected to this page until he is a part of a project.

In the authorization part only the added users are has the access to the portal and only permitted users can do all the operations of the project. In-order to add an user in the project the user does not needed to be signed up in security audit. When an user who already belongs to some portal is signing up then he will be automatically mapped to the previously created entry and he does not need to create any portal. A user can be a part of many no of portals, the section 5.2 gives how this is implemented and the authorization is handled.

5.2 Project portals & Project Dashboard

5.2.1 Project portals

As mentioned a user can be a part of multiple projects, the user can view all his projects in the user's home page and he can click on the specific project to do the operations. The figure 4 shows how the explained behaviour looks in the browser, the figure the project "Testing2" is owned by the user and he is just an user to the other project "Testing". As a part of authorization the owner of the project is only allowed to edit the project and the users are not allowed to perform such actions. The user can create a new project if he wants, this option is also provided in the user's home page.

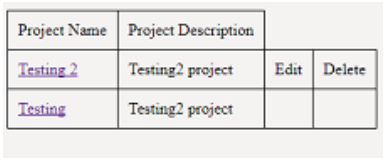

Figure 4: Project portals of user.

5.2.2 Project dashboard

The user can click on the project in which he needs to perform any operation or view some information, once he clicks he will be redirected to the project dashboard which gives the complete information of the project. This the place where the user can find the operations which are permitted for him to perform like adding or removing users from project(currently this is performed only by the admin), start a security scan, view the scan details and download the generated report. The currently all the users in the project can start a new security scan and download the report file. The scan details section contains the following below details and the figure5 show how it is shown in the user interface.

- Scanned by(User name of the developer who started the scan).
- Scanned Date (When it was started).
- Status (Completed or In progress or Failure)
- Total vulnerable count(Threat count identified in the uploaded source package)
- Top vulnerable file(The file with top must vulnerable count)

- Generated File(Scan Report)

Recent scanned reports +

Scanned By	Scanned Date	Scan Status	Total Vulnerable Count	Top Vulnerable File	Generated File
ramyabharathi.d	8/9/17	Completed	18	action.SecurityScan with 6 threats.	Testing 2.pdf
ramyabharathi.d	8/9/17	Completed	25	action.SecurityScan with 6 threats.	Testing 2.pdf
ramyabharathi.d	8/9/17	Completed	24	action.SecurityScan with 6 threats.	Testing 2.pdf
ramyabharathi.d	8/9/17	Completed	25	action.SecurityScan with 6 threats.	Testing 2.pdf
ramyabharathi.d	8/9/17	Completed	25	action.SecurityScan with 6 threats.	Testing 2.pdf
ramyabharathi.d	8/9/17	In progress	0		

Figure 5: Recent project scans.

5.3 Security Scan

The user uploads the source code package from the project dashboard, only .zip and .war file is accepted for scanning. When the scan request is received by the action file it extracts all the files into the respective directories as it is in the compressed file. Once all the files are extracted a schedule is immediately started to scan all the files and the schedule calls the scan object by passing the scan id.

Once the scan object gets the scan id then it fetches the project scan details from the data base, the project scan details contains the extracted file path. It reads the file one by one, the scan object sends the file to the respective method for checking all the lines in the file. Each and every line is passed to the blacklisted object which further checks with the whitelisted object whether it is safe or not. If the line contains any threat then it is added to the threat list of the file and reads the next line and this process continues till the last file.

Reading the file and both black and whitelist object differs for the java byte code file and jsp file which clearly explained in the sections 5.3.1 and 5.3.2. As said in the 1 most of the xss and cookie & session manipulation occurs because of exposing the user data in the web application without any proper treatment. So the black and whitelisted object of the both the file type concentrates on certain object classes which exposes the sensitive data.

5.3.1 Java byte code file scanning

Reading a java byte code file is complectly different from that of the jsp file, with help of the Apache BCEL all the instructions are read. The BCEL instructionhandel identifies what kind of instruction in executed in the specific line. The byte code instruction is of many types, but here the blacklist check is done for the instruction which comes under invokevirtual, invokespecial and invokeinterface. The instruction of those 3 kind is sent to the blacklist object, if the invoked instruction is listed in the blacklist object then the blacklist object sends the params used in the instruction to the whitelist and checks

whether it is using any of the whitelisted class objects. If so then that specific line is considered as threat free and it starts reading the next instruction. The identified instruction details like invoker specific type, line no and the reason for considering it as threat is added to the file list. The below table **??** and 2 contains the complete list of blacklisted invoker class and the whitelisted objects of them.

Blacklist Class	**Method**	**Need to check Whitelist**
PrintStream	print & println	No
PrintWriter	print & println	Yes
OutPutStream	flush	No but ignores if responseheader is set
FileOutputStream	flush	No but ignores if responseheader is set
BufferedOutputStream	flush	No but ignores if responseheader is set
DataOutputStream	flush	No but ignores if responseheader is set
ObjectOutputStream	flush	No but ignores if responseheader is set
OutputStreamWriter	flush	No but ignores if responseheader is set
HttpSession	setAttribute & putValue	yes
HttpServletRequest	setAttribute	yes
Cookie	setValue and new creation	yes

Table 1: Blacklisted java class object

Whitelisted class object
Integer
Long
Double
Boolean
URLEncoder
com.itextpdf.text.html.HtmlEncoder
com.security.Encypter

Table 2: Whilisted class object

5.3.2 JSP file scanning

The jsp file is read like a .txt file, the process is same like the 5.3.1 execpt the blacklist and whitelist configurations and certain logics in identifying the threats. The one major difference in reading both files is java byte code contains only one invocable instruction in one line but in jsp it may contain more than one in a single line. So the line is checked again and again after identifying the threat only because no threat should not be missed, so the logic for this is the additional one added in the process compared to 5.3.1. The blacklisted and their whitelist methods are listed in the tables 3 and 4.

Blacklist Class	**Method**	**Need to check Whitelist**
HttpServletRequest	setAttribute & getAttribute	yes
Struts2:property	escapeHtml,escapeJavaScript & escapeXml	yes

Table 3: Blacklisted java class object

Whitelisted class object
Integer
Long
Double
Boolean
escapeAttrbute=false
JSP comment

Table 4: Whilisted class object

5.4 Report generation and storing scan details

Once the scanning for all the files are completed, the identified threats are added to a list and that list is passed to the report generation object. The report generation with the help of "itextpdf" it writes all the threats added in the list. The pdf starts with the basic information of the scan i.e. the details shown in the recent scan details section. It iterates all file threats and list them accordingly, it creates a section for each file and all the threats are written in acceding order based on the line no. It mentions the threat content, line no and the reason for marking it as threat and it may also suggest some remedies. After writing all the threats in the list the file is saved in the system director.

Once the report generation object completes writing the file, then the scan object stores all the needed scan information in the data base, which is then rendered in the project information page. The figures 6, 7 and 8 are sample of a generated report.

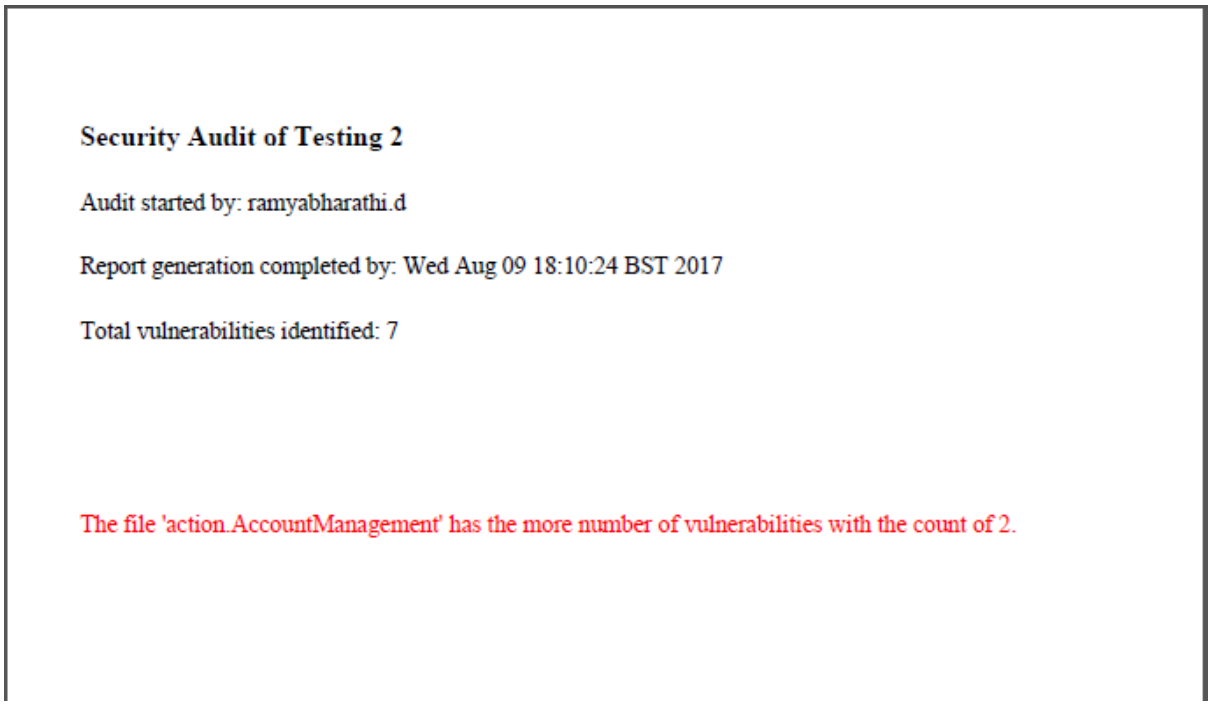

Figure 6: Basic scan info

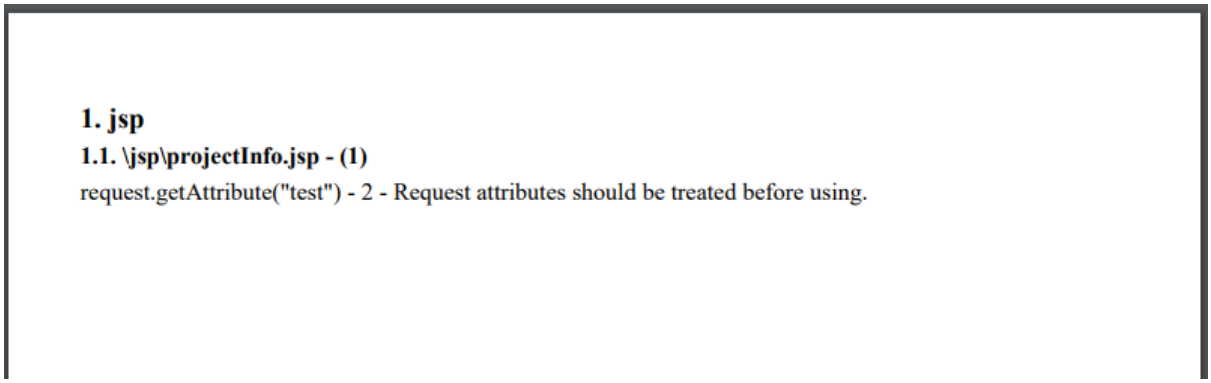

Figure 7: JSP identified threats

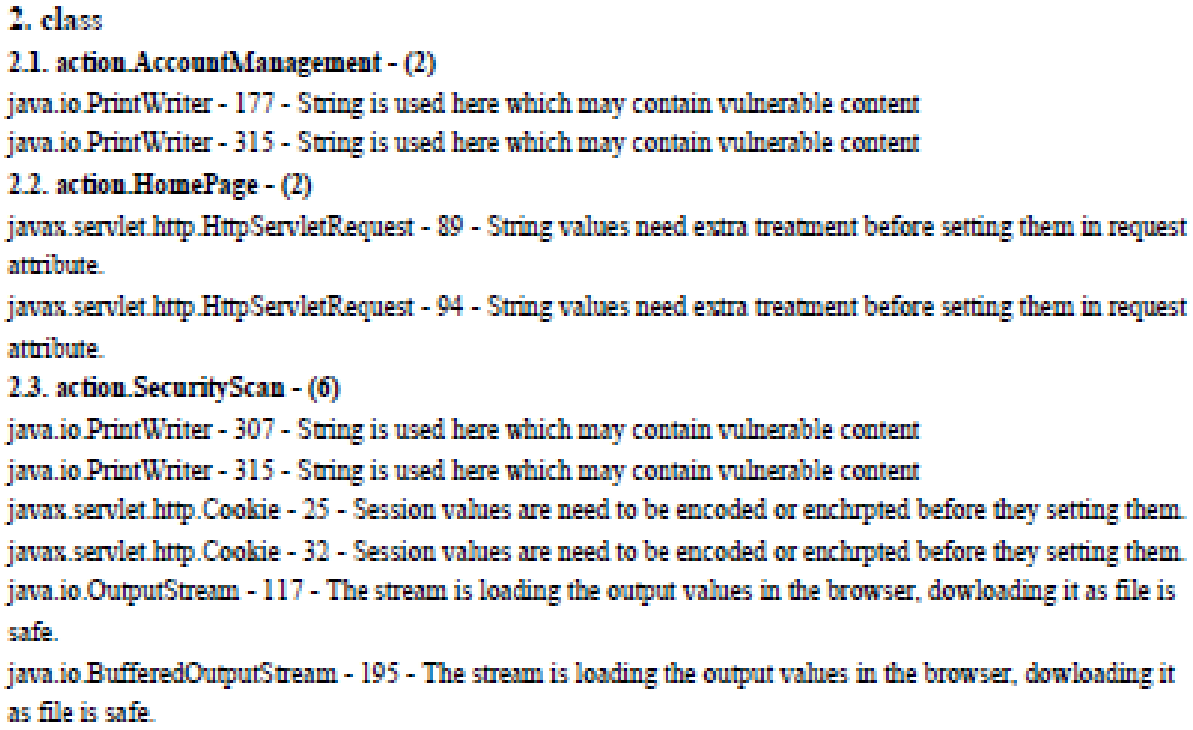

Figure 8: Java Bytecode threats

6 Evaluation

This section performs few tests in-order to check whether the developed research project is working fine and generates proper results. Security audit is evaluated using the source package which is formed using the online source code files, the source code package is altered based on the use case.

6.1 Case 1: Identifying the threats in Java byte code

Using the created source file the security scan is started, it identified all marked blacklisted invoker and syntaxes. The invokers are added one by one in the blacklist object and used

the same source code. When the invoker count is increased then gradually the threat count increased, also based on the usage of the invoker the top file also changes.

At first the whitelist object was not declared to identify the difference before and after adding the whitelist. The Object class which are considered to be safe are then added one by one and the count decreased if the specific passed arguments are instance of the whitelisted objects. The count difference is as shown in the table 5.

Blacklist invoker count	Whitelist count	Threat count
3	0	8
6	0	14
11	0	19
11	3	15
11	7	10

Table 5: Evaluation of Java byte code

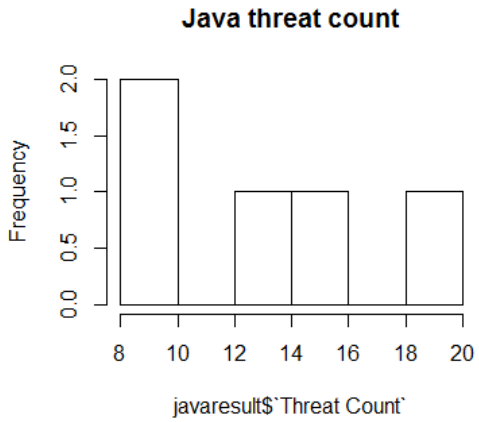

Figure 9: Graphical representation of identified Java threat count

6.2 Case 2: Identifying the threats in JSP

The source code for this evaluation contains only JSP files, the same process is continued here. The blacklist objects are added one by one and first the evaluation is made without whitelist and then again the source code is evaluated after adding the whitelist. There no much threats identified in this file type as compared to java identification, which is shown in table 6.

Blacklist invoker count	Whitelist count	Threat count
1	0	7
2	0	12
2	3	8
2	6	7

Table 6: Evaluation of Java byte code

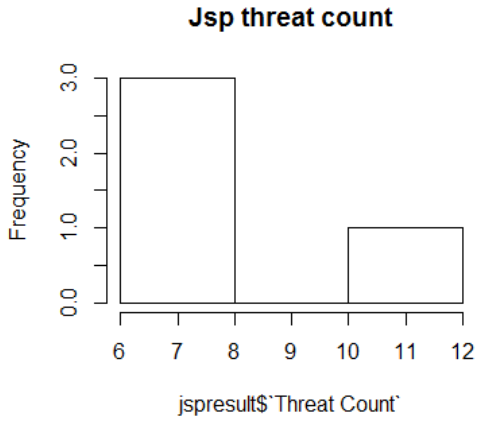

Figure 10: Graphical representation of identified JSP threat count

6.3 Final Evaluation

In this both the file types(java & jsp) are included and the scan is run for the source package. This is helped to know whether Security audit is capable of identifying both the types. So for this test the source files which were used in last test of both the above is combined, the obvious thing is Security audit should identify and list total of 17 threats and it did. And here some of the threats where removed and new codes where inserted in different files, the result fluctuated which is shown in the table 7.

Scan count	Threat count
1	17
2	14

Table 7: Evaluation of Java byte code

6.4 Discussion

With the help of the above evaluation it is clear that scanning the source code for security concern is very important and helpful while developing. The generated result helps the developers to realise how poor their security management is and also helps them identify the files which really needs more attention for security rather than other. This also helps the manager to know which developer is continuously making such mistakes and correct it.

7 Conclusion and Future Work

The proposed research helps in understanding the importance of running a security audit frequently to avoid the minor code mistakes which causes major threats. It is hard to identify certain things in the code, because checking line by line is not possible for any managers or team leaders. Auditing the code is a best practise and this helps not only identify the threats but also the developers who are not practicing the security preventive measures. Those developers has to be informed about the practice and its importance.

Security audit helps in reducing the threats at basic level, not only this but using security audit they can frame their own security measures and implement them to increase the quality.

Security audit aims in supporting lot more things than what it is now made. Currently is scans only jsp and java, but in future it will include js files too. Now Security audit has included only struts2 framework tags, but in future it will include struts 1, user's customized taglibrary and spring framework. The company can also include their own policies in security audit and security audit will identify those issues too. The whitelisted class object "com.security.Encypter" is created for encrypting and decrypting purpose of Security audit and it is included as in whitelist, likewise the company can also include their class objects.

Acknowledgements

I am taking this opportunity to thank **Mr. Manuel Tova-Izquierdo** for giving me this opportunity and helping me in completing this project, without him it would have not been possible for me to do the research project.

References

Bates, D., Barth, A. and Jackson, C. (2010). Regular expressions considered harmful in client-side xss filters, *Proceedings of the 19th international conference on World wide web*, ACM, North Carolina, USA, pp. 91–100.

Jim, T., Swamy, N. and Hicks, M. (2007). Defeating script injection attacks with browser-enforced embedded policies, *Proceedings of the 16th international conference on World Wide Web*, ACM, Alberta, Canada, pp. 601–610.

Jovanovic, N., Kruegel, C. and Kirda, E. (2006). Pixy: A static analysis tool for detecting web application vulnerabilities, *Security and Privacy, 2006 IEEE Symposium on Security and Privacy*, IEEE, pp. 6pp – 263.

Kaufman, L. M. (2009). Data security in the world of cloud computing, *IEEE Security and Privacy* **7**(4).

Kieyzun, A., Guo, P. J., Jayaraman, K. and Ernst, M. D. (2009). Automatic creation of sql injection and cross-site scripting attacks, *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on Software Engineering*, IEEE, Vancouver, Canada, pp. 199–209.

Kirda, E., Kruegel, C., Vigna, G. and Jovanovic, N. (2006). Noxes: a client-side solution for mitigating cross-site scripting attacks, *Proceedings of the 2006 ACM symposium on Applied computing*, ACM, Dijon, France, pp. 330–337.

Meyerovich, L. A. and Livshits, B. (2010). Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser, *Security and Privacy (SP), 2010 IEEE Symposium on Security and Privacy*, IEEE, pp. 481–496.

Miyazaki, A. D. (2008). Online privacy and the disclosure of cookie use: Effects on consumer trust and anticipated patronage, *Journal of Public Policy & Marketing* **27**(1): 19–33.

Rabinovich, P. (2013). Secure cross-domain cookies for http, *Journal of Internet Services and Applications* **4**(1): 13.

Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing, *Journal of network and computer applications* **34**(1): 1–11.

Tappenden, A. F. and Miller, J. (2009). Cookies: A deployment study and the testing implications, *ACM Transactions on the Web (TWEB)* **3**(3): 9.

Tappenden, A. F. and Miller, J. (2014). Automated cookie collection testing, *ACM Transactions on Software Engineering and Methodology (TOSEM)* **23**(1): 3.

Ter Louw, M. and Venkatakrishnan, V. (2009). Blueprint: Robust prevention of cross-site scripting attacks for existing browsers, *Security and Privacy, 2009 30th IEEE Symposium on Security and Privacy*, IEEE, pp. 331–346.

Van Gundy, M. and Chen, H. (2012). Noncespaces: Using randomization to defeat cross-site scripting attacks, *computers & security* **31**(4): 612–628.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C. and Vigna, G. (2007). Cross site scripting prevention with dynamic data tainting and static analysis., *NDSS*, Vol. 2007, p. 12.

Wassermann, G. and Su, Z. (2008). Static detection of cross-site scripting vulnerabilities, *Proceedings of the 30th international conference on Software engineering*, ACM, Leipzig, Germany, pp. 171–180.

Weinberger, J., Saxena, P., Akhawe, D., Finifter, M., Shin, R. and Song, D. (2011). A systematic analysis of xss sanitization in web application frameworks, *European Symposium on Research in Computer Security*, Springer, Verlag, Berlin, pp. 150–171.

Wurzinger, P., Platzer, C., Ludl, C., Kirda, E. and Kruegel, C. (2009). Swap: Mitigating xss attacks using a reverse proxy, *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*, IEEE Computer Society, Vancouver, Canada, pp. 33–39.

Zissis, D. and Lekkas, D. (2012). Addressing cloud computing security issues, *Future Generation computer systems* **28**(3): 583–592.

A Appendix

```
public enum BLACK_LISTED_INVOKERS {

    PRINTSTREAM(PrintStream.class.getName(),PrintStream.class) {
    PRINTWRITER(PrintWriter.class.getName(),PrintWriter.class) {
    OUTPUTSTREAM(OutputStream.class.getName(),OutputStream.class) {
    FILEOUTPUTSTREAM(FileOutputStream.class.getName(),FileOutputStream.class) {
    BUFFEREDOUTPUTSTREAM(BufferedOutputStream.class.getName(),BufferedOutputStream.class) {
    DATADOUTPUTSTREAM(DataOutputStream.class.getName(),DataOutputStream.class) {
    OBJECTDOUTPUTSTREAM(ObjectOutputStream.class.getName(),ObjectOutputStream.class) {
    OUTPUTSTREAMWRITER(OutputStreamWriter.class.getName(),OutputStreamWriter.class) {
    HTTPSESSION(HttpSession.class.getName(),HttpSession.class) {
    HTTPSERVLETREQUST(HttpServletRequest.class.getName(),HttpServletRequest.class) {
    COOKIE(Cookie.class.getName(),Cookie.class) {

    private Class invokerClass;
    private String invokerString;
    private boolean checkResoponseHeader;
    private static boolean isInvokeSpecial = false;

    public boolean isCheckResoponseHeader() {
        return checkResoponseHeader;
    }

    public void setCheckResoponseHeader(boolean checkResoponseHeader) {
        this.checkResoponseHeader = checkResoponseHeader;
    }

    private BLACK_LISTED_INVOKERS(String invokerString,Class invokerClass)
    {
        this.invokerString = invokerString;
        this.invokerClass = invokerClass;
    }
```

Figure 1: Java blacklist object

```
public enum WhitelistedParams {

    LONG(Long.class.getName(),Long.class),
    DOUBLE(Double.class.getName(),Double.class),
    INTEGER(Integer.class.getName(),Integer.class),
    URLENCODER(URLEncoder.class.getName(),URLEncoder.class),
    HTMLENCODED(HtmlEncoder.class.getName(),HtmlEncoder.class),
    ENCRYPTER(Encrypter.class.getName(),Encrypter.class);

    private Class paramClass;
    private String classString;

    private WhitelistedParams(String classString,Class paramClass)
    {
        this.classString = classString;
        this.paramClass = paramClass;
    }

    public Class getParamClass() {
        return paramClass;
    }
    public String getClassString() {
        return classString;
    }
```

Figure 2: Java whitelist object

```
public enum JSP_BLACK_LIST {

    REQUEST_GET("request",".","get",")") {..
    REQUEST_PUT("request",".","put",")") {..
    STUTS2_PROPERTY("<s",":","property",">") {..

    private String jspKeyword;
    private String method;
    private String joiner;
    private String endString;
    private int startIndex;
    private String vulnerableContent;

    public  int getStartIndex() {
        return startIndex;
    }

    public void setStartIndex(int startIndex)
    {
        this.startIndex = startIndex;
    }
    public String getJspKeyword() {
        return jspKeyword;
    }

    public String getMethod() {
        return method;
    }

    public String getJoiner()
    {
        return joiner;
    }
```

Figure 3: JSP blacklist object