# Prevention of cross site request forgery in CRM

Submitted By

**Taslim Agwan**

**17MCEI01**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**INSTITUTE OF TECHNOLOGY**

**NIRMA UNIVERSITY**

**AHMEDABAD-382481**

**May 2019**

# Prevention of cross site request forgery in CRM

**Major Project**

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering (INS)

Submitted By

**Taslim Agwan**

**(17MCEI01)**

Guided By

**Dr. Zunnun Narmawala**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**INSTITUTE OF TECHNOLOGY**

**NIRMA UNIVERSITY**

**AHMEDABAD-382481**

**May 2019**

# Certificate

This is to certify that the major project entitled **"PREVENTION OF CROSS SITE REQUEST FORGERY IN CRM"** submitted by **TASLIM AGWAN (17MCEI01)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination.

Dr. Zunnun Narmawala
Associate Professor,
CE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Sharda Valiveti
Associate Professor,
Coordinator M.Tech - CSE (INS),
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Madhuri Bhavsar
Professor and Head,
CE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad.

# Statement of Originality

I, **Taslim Agwan**, Roll. No. **17MCEI01**, give undertaking that the Major Project entitled "**Prevention of cross site request forgery in CRM**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering (Information & Network Security)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:

Place:

Endorsed by

Dr. Zunnun Narmawala

(Signature of Guide)

# Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Dr. Zunnun Narmawala**, Associate Professor, Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Sharada Valiveti** , Hon'ble Head of Computer Engineering/ Information Technology Department (INS) , Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

It gives me an immense pleasure to thank **Dr. Madhuri Bhavsar** , Hon'ble Head of Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

**Taslim Agwan**
**17MCEI01**

# Abstract

The project aims to provide security against cross-site request forgery attack in CRM (customer relationship management). CRM system manages huge customer data. Customer data needs to be protected from the OWASP security risks. Which includes cross-site request forgery. This is a type of attack where unapproved actions are performed using a legitimate login session that the web application trusts. The current approach is using XSRF tokens to prevent CSRF attack in GWT application. PMD tool is used to analyze the use of vulnerable GWT APIs. CSRF Guard is used to defend against CSRF attack on JAVA platform. This module implements the CSRF guard configurations for Oracle JET framework in CRM. CSRF token injection approach is introduced.

# Abbreviations

| | |
|---|---|
| **CRM** | Customer Relationship Management. |
| **CSRF / XSRF** | Cross Site Request Forgery. |
| **POS** | Point Of Sale. |
| **HTTP** | Hyper Text Transfer Proeocol |
| **URL** | Uniform Resource Locator. |
| **XSS** | Cross Site Scripting. |
| **GWT** | Google Web Toolkit. |
| **HTTP** | Hyper Text Transfer Proeocol |
| **HTML** | Hyper Text Markup Language |
| **JSON** | JavaScript Object Notation |
| **XML** | Extensible Markup Language |
| **OJET** | Oracle JavaScript Extension Toolkit |

–

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview

POS system is widely used in the retail industry in order to enhance sales and productivity. Figure 1.1 shows the integration between POS and CRM. There is a number of POS systems integrated with a central customer data maintenance system like CRM systems. CRM is a suite of integrated services designed to increment revenue and profitability for retail enterprises. It collects customer details (transaction details, contact details) from the POS system, in order to analyze customers purchase pattern and target high potential customers. Hence prevention of data loss is needed.

This prevention module specifically targets CSRF attack prevention in CRM. CSRF is an attack that ploys the victim's browser to execute unauthorized commands in CRM to which an associate is logged in. CSRF attack is performed using social engineering, which tricks the authorized user into sending a fake request to a server. As a victim is authorized while sending a forged request. It is hard to differentiate an authorized request from a fake request.

Previously, in CRM, XSRF tokens were used to prevent such vulnerabilities. Existing prevention techniques are time-consuming and weak. These techniques need manual handling to apply protection techniques into an existing CRM system. Minimal manual efforts are needed in the new technique (CSRF guard).

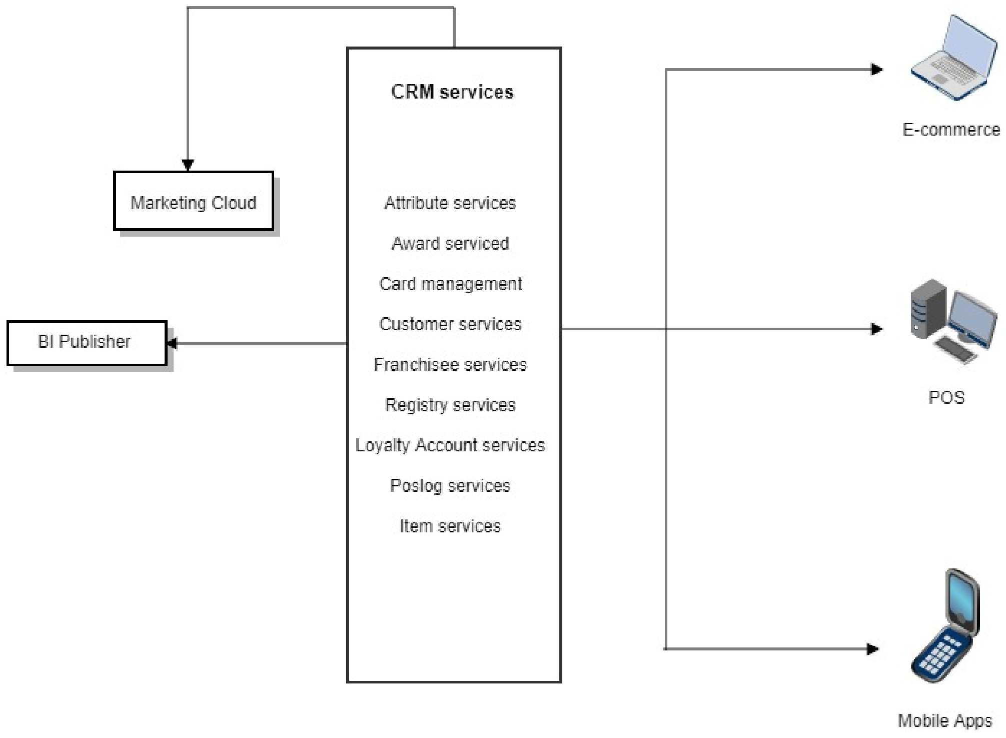


Figure 1.1: Point of sale and CRM

## 1.2 Objective

- To reduce Retail Industry loss.
- To reduce the number of application layer attacks (CSRF attack).
- To integrate the defense technique for new Oracle JET framework.
- To overcome the inadequacies of XSRF tokens and other existing prevention techniques.
- To ease the risk of bypassing a new approach, that is introduced.
- To reduce data loss and theft using a secure exchange of CSRF prevention.

# Chapter 2

# Literature Survey

## 2.1 Survey on the mechanism behind CSRF attack

Stateless protocol (HTTP) can not recognize whether all the requests belong to a particular user or not. Protocol use client-side cookies to maintain user-specific state. The browser will automatically add this cookie information in HTTP header[1]. This is not helpful when there is a huge data exchanging between client and server. To address this issue sessions are used. The session ID is generated, in order to distinguish the legitimate user. Manual or automatic URL rewriting is used to append session ID into URL. Here session ID is used as absolute authentication token[2].

Session mechanism can be exploited by the attackers. The session ID is equivalent to the users original credentials. XSS exploit can be used to reveal the session ID. Attacker misuses the fact that web applications cannot differentiate between session ID from an authorized user ID and stolen session ID. An attacker can misuse authenticated users session to perform fraud transactions, password change or any other illegal activity[1].

A typical GET request for a $500 bank transfer :

**GET HTTP://bank.com/transfer.do?acct=PersonB&amount= $500 HTTP/1.1**

A hacker can modify this request and make this transfer to his own account.

**GET HTTP://bank.com/transfer.do?acct=AttackerA&amount=$500 HTTP/1.1**

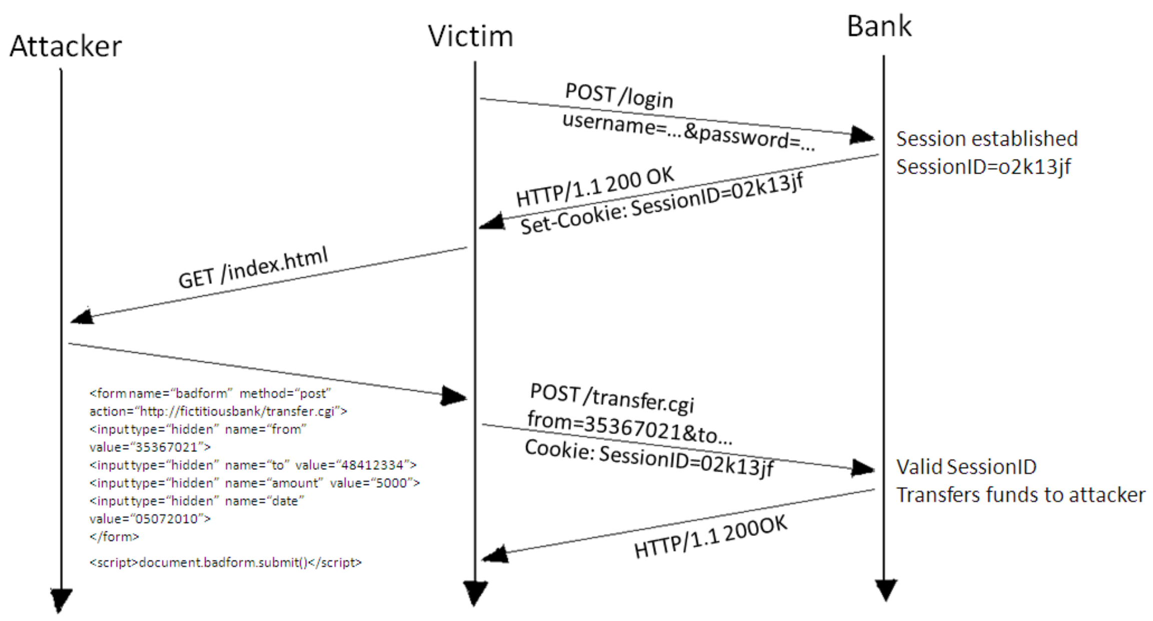
This malicious link can be embedded into any simple looking hyperlink or it can be embedded into an image link also :

**<a href="HTTP://bank.com/transfer.do?acct=AttackerA&amount=$500">Read more!</a>**

**<img src="HTTP://bank.com/transfer.do?acct=AttackerA&amount=$500">**

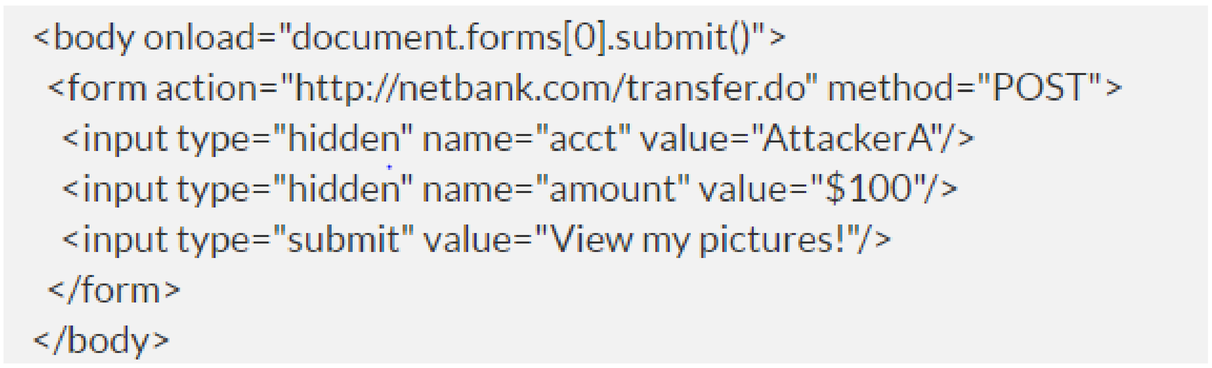
If a bank is only using POST requests, then malicious code snippet can be executed using form tag, with automatic implementation of JavaScript functions :

Figure 2.1 and 2.2 demonstrate the CSRF attack mechanism using POST request.



```
<form name="badform" method="post"
action="http://fictitiousbank/transfer.cgi">
<input type="hidden" name="from"
value="35367021">
<input type="hidden" name="to" value="48412334">
<input type="hidden" name="amount" value="5000">
<input type="hidden" name="date"
value="05072010">
</form>
<script>document.badform.submit()</script>
```

Figure 2.1: CSRF Attack



```
<body onload="document.forms[0].submit()">
 <form action="http://netbank.com/transfer.do" method="POST">
  <input type="hidden" name="acct" value="AttackerA"/>
  <input type="hidden" name="amount" value="$100"/>
  <input type="submit" value="View my pictures!"/>
 </form>
</body>
```

Figure 2.2: POST request malicious code [1]

When user click on such malicious links, while he is logged into his bank account. Then these scripts can be executed successfully using authenticated sessions.

## 2.2 Survey on Existing CSRF defense Techniques

1. Using POST requests :

   Common CSRF prevention practice is to use POST request instead of GET request. As this attack takes advantage of the default behavior of the HTML parser in the victims browser, this approach is not enough. As discussed in the previous survey, using JavaScript attacker can perform CSRF attacks even when POST requests are used [3].

2. Captchas :

   Adding captchas will force the user to interact with the browser ( application ) till the last and final request. The web application can not include captchas to each and every operation that is being performed. Thus, it is not an ideal solution[2].

3. Referer header check :

   The HTTP referer header is a field that shows the location of the webpage that linked to the source being requested. It is the address from where the request is being originated. The legitimacy of the referer can be checked. The browser can maintain white-list of accepted referer headers. But sending this referer header can result in leaking the sensitive information to unauthorized parties. So, referer headers can be empty. Which can make impossible to detect the CSRF attack. An attacker can use browser specific settings to trigger CSRF exploit using empty referer headers[4].

4. Proxy based solution :

   Separate module is included in the existing system with minimal attempt. Proxy is placed at the server side, between the target application and web server [1]. Proxy inspects and modifies the requests. Proxy modifies the response, in a way that future requests will contain valid tokens. Take countermeasures when an invalid token is encountered. Proxy associates the users session to the valid token and maintains the token table.

# Chapter 3

# CSRF Prevention approach in GWT (Google Web Toolkit)

## 3.1 Google Web Toolkit

### 3.1.1 GWT

Google Web Toolkit is used to create complicated JavaScript front-end web applications in JAVA. GWT builds a web application by compiling Java code base into JavaScript front end code. Vulnerability assessment on GWT applications is not feasible because interdependency between the Java code base and compiled JavaScript front end code is tough to detect. Java source code can be analyzed to detect potential vulnerabilities.

### 3.1.2 Current Approach

XSRF tokens were used previously in GWT. These XSRF tokens are generated randomly and are difficult to predict. An attacker can obtain important parameters by analyzing or using other exploits and can forge the request. Adding XSRF tokens is an effective method.

NoForge is a server-side agent, which executes the token mechanism and rewrites the URL [5]. Dynamically created links cannot be handled by this mechanism. Two-level token defense mechanism can also be used, to harden the security. The first level token is used for base security and for higher security second level token is used. Transactions requests only proceed further if they contain the correct token.

Correct CSRF token does not mean that sent data is trustable. But it means that it

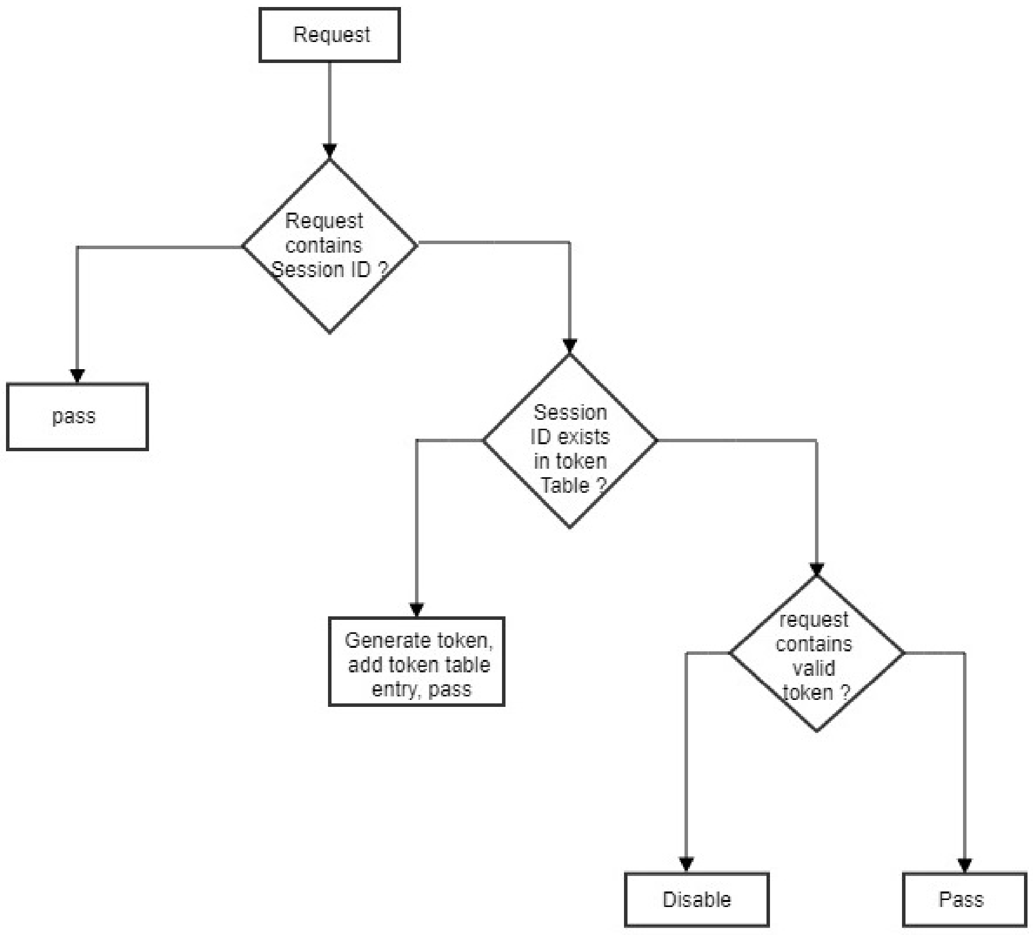

Request
Request contains Session ID ?
pass
Session ID exists in token Table ?
Generate token, add token table entry, pass
request contains valid token ?
Disable
Pass

Figure 3.1: The XSRF token approach in GWT

was the users intention to send the data. Token Table is maintained [3.1].

1. Every session ID is associated with a certain token number.
2. Token is checked for every request. If a request has token, the corresponding session ID is checked in the token table. If a request contains valid token, then request is declared as valid.
3. Proper token is attached along with the response. Classes used for vulnerability protection :
   - Com.google.gwt.safehtml.shared :

     Creates safe and encoded HTML content
   - SimpliSafeHtmlrenderer :

     Safey renders HTML code

- XsrfProtectedService

# Chapter 4

# Scanning the GWT code for vulnerabilities

## 4.1 GWT vulnerable APIs

1. Text and HTML

   This classes exposes HasHTML and SetHTML methods, which can inject malicious JavaScript code when it is called with untrusted data[6].

2. URLs

   Anchor and HyperLink uses setHref methods and accepts the URL. which is not trustworthy if URL is not encoded[6].

3. HTTP Request and cookies

   RequestBuilder class allows direct changes in request header and URLs. Response class allows accessing response headers. Cookies class grants access to browser cookies on the client machine. Data should not be manipulated using Getters and setters used for cookies [6].

4. Server communication

   RemoteServiceServlet and AsyncCallback classes are used to process the request and send the response to the matching client. This also uses XsrfProtectedService and XsrfProtectedServiceServlet classes to protect the application against CSRF[6].

5. Dynamic scripts

RemoteServiceServlet and AsyncCallback classes are used to process the request and send the response to the matching client. This also uses XsrfProtectedService and XsrfProtectedServiceServlet classes to protect the application against CSRF[6].

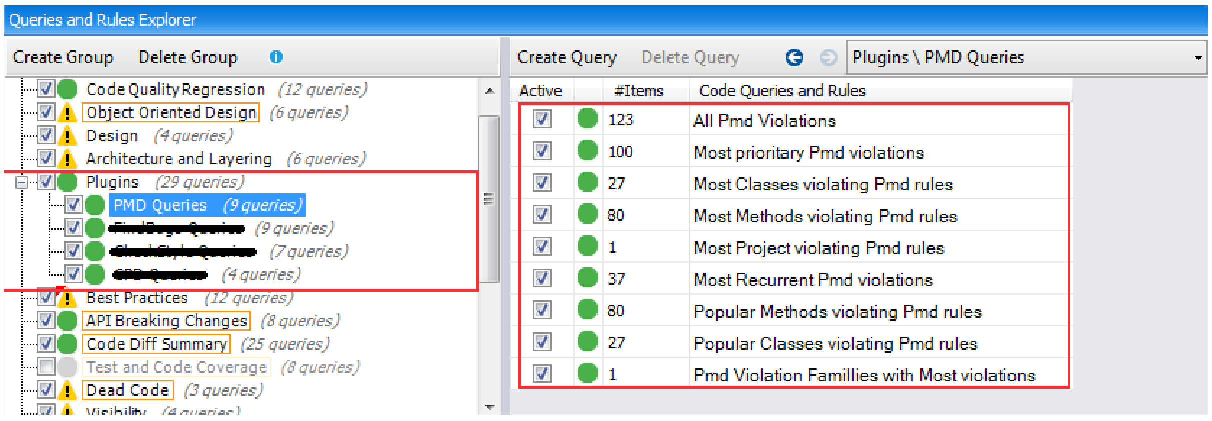
6. Parsing

   JSONParser and XMLParser accepts the data in JSON or XML formats. This data not be malicious[6].

## 4.2 Scannig the GWT code

To check instances of vulnerable APIs analysis is needed. Instead of checking it manually, the static analysis tool was used. Fortify is one of the tools used to check the vulnerability in the code. But this tool can only scan java code JSP script. And this did not work in GWT [6]. FindBugs and PMD tools were used to analyze the GWT code. These tools do not contain default rules for analysis, but custom rules can be written. PMD runs the analysis on parsed Java code base.

Custom PMD rules can be created to analyze the use of vulnerable APIs. These rules are created according to the listed APIs in the previous section. These rules were used to check the use of vulnerable APIs. Potential vulnerabilities are counted per line of code. Figure 4.1 shows the PMD tool processing for GWT code scanning.



- Code Quality Regression (12 queries)
- Object Oriented Design (6 queries)
- Design (4 queries)
- Architecture and Layering (6 queries)
- Plugins (29 queries)
  - PMD Queries (9 queries)
  - [redacted] (9 queries)
  - [redacted] (7 queries)
  - [redacted] (4 queries)
- Best Practices (12 queries)
- API Breaking Changes (8 queries)
- Code Diff Summary (25 queries)
- Test and Code Coverage (8 queries)
- Dead Code (3 queries)
- Visibility (4 queries)

| Active | | #Items | Code Queries and Rules |
|---|---|---|---|
| ✓ | ● | 123 | All Pmd Violations |
| ✓ | ● | 100 | Most prioritary Pmd violations |
| ✓ | ● | 27 | Most Classes violating Pmd rules |
| ✓ | ● | 80 | Most Methods violating Pmd rules |
| ✓ | ● | 1 | Most Project violating Pmd rules |
| ✓ | ● | 37 | Most Recurrent Pmd violations |
| ✓ | ● | 80 | Popular Methods violating Pmd rules |
| ✓ | ● | 27 | Popular Classes violating Pmd rules |
| ✓ | ● | 1 | Pmd Violation Famillies with Most violations |

Figure 4.1: GWT code scanning using PMD

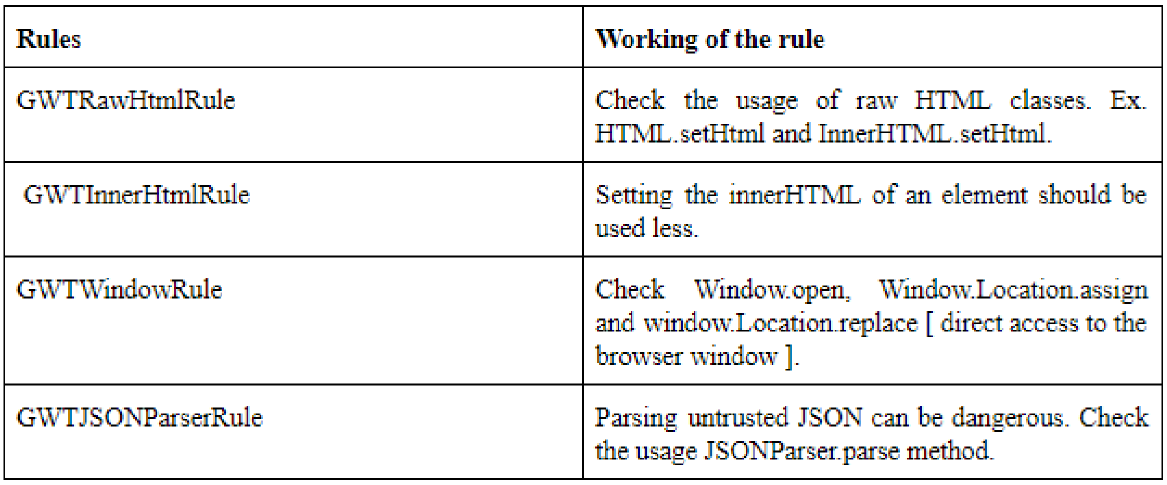
**Existing PMD Rules :**

- GWTSetTextRule
  - looks for setText and hasText methods. Texts ( from an untrusted source ) should be encoded.

- GWTUrlRule
  - All the URLs provided by user should be URIEncoded. Looks for setHref and setURL methods.
- GWTRequestBuilderRule
  - This rule counts the use of RequestBuilder, JsonRequestBuilder and XML-HttpRequest.
- GWTCookieRule
  - Looks for setCookie and getCookie methods.
- GWTRemoteServiceRule
  - Check if remote services are XSRF protected or not. Checks for the usage of RemoteService and XsrfProtectedService,
- GWTScriptRule
  - Checks the usage of scriptElement and scriptInjector.
- GWTXMLParseRule
  - Parsing malicious XML can be risky. This rule counts the use of XML-Parser.parse.

**New PMD Rules created :**

- Figure 4.2 shows the new GWT rules created.



| Rules | Working of the rule |
|---|---|
| GWTRawHtmlRule | Check the usage of raw HTML classes. Ex. HTML.setHtml and InnerHTML.setHtml. |
| GWTInnerHtmlRule | Setting the innerHTML of an element should be used less. |
| GWTWindowRule | Check Window.open, Window.Location.assign and window.Location.replace [ direct access to the browser window ]. |
| GWTJSONParserRule | Parsing untrusted JSON can be dangerous. Check the usage JSONParser.parse method. |

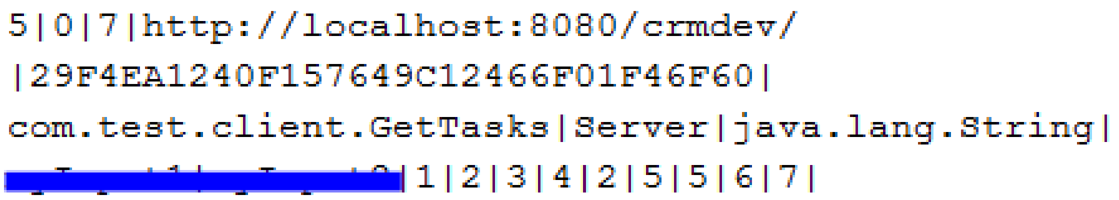
Figure 4.2: New GWT rules

## 4.3 GWT RPC calls

GWT is used to create AJAX enable Java web applications. GWT allows developers to assemble the components in Java and Then the JAVA code is compiled into advance JavaScript for browsers to run.

GWT RPC calls allow sending JAVA custom objects (data) from front end JavaScript code to JAVA back end server code. Whenever the AJAX call is made, serialized JAVA objects are streamed.

RPC call is sent as HTTP POST request from the browser. The serialized stream is in plain text and separated by vertical lines as shown in Figure 4.3.



```
5|0|7|http://localhost:8080/crmdev/
|29F4EA1240F157649C12466F01F46F60|
com.test.client.GetTasks|Server|java.lang.String|
[redacted]|1|2|3|4|2|5|5|6|7|
```

Figure 4.3: GWT RPC call stream

This stream is made of three different parts :

- Header
- String table

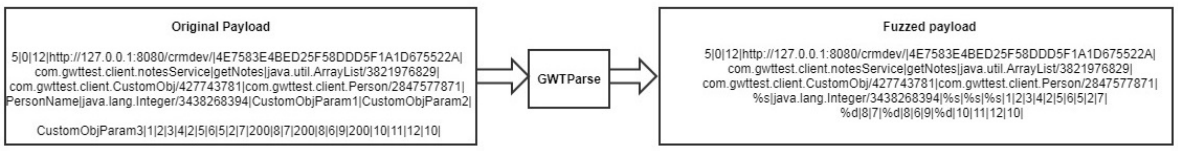
- Payload

### 4.3.1 Fuzzing GWT RPC calls

In GWT RPC calls, there are payload values that can be manipulated and are fuzzable. GWT Parse tool is used to automate the process of extracting the values within an RPC call payload that can actually be changed and are vulnerable. Security bugs are identified in this application using this tool.

GWT front end code is too obfuscated to read, so it is tough to recognize all the fuzzable values passed in the request by only viewing the JS code.

This is a command line tool which creates new payload with all fuzzable value identified. As shown in Figure 4.4 default output will replace fuzzable string value with %s and numeric value with %d.



```
5|0|12|http://127.0.0.1:8080/crmdev/|4E7583E4BED25F58DDD5F1A1D675522A|
com.gwttest.client.notesService|getNotes|java.util.ArrayList/3821976829|
com.gwttest.client.CustomObj/427743781|com.gwttest.client.Person/2847577871|
PersonName|java.lang.Integer/3438268394|CustomObjParam1|CustomObjParam2|

CustomObjParam3|1|2|3|4|2|5|6|5|2|7|200|8|7|200|8|6|9|200|10|11|12|10|
```

```
5|0|12|http://127.0.0.1:8080/crmdev/|4E7583E4BED25F58DDD5F1A1D675522A|
com.gwttest.client.notesService|getNotes|java.util.ArrayList/3821976829|
com.gwttest.client.CustomObj/427743781|com.gwttest.client.Person/2847577871|
%s|java.lang.Integer/3438268394|%s|%s|%s|1|2|3|4|2|5|6|5|2|7|
%d|8|7|%d|8|6|9|%d|10|11|12|10|
```

Figure 4.4: Fuzzing the GWT RPC calls

### 4.3.2 Enumerating GWT RPC calls

Enumeration is used to reduce the complexities exposed by GWT code. GWT obfuscates the client-side JavaScript code. Which makes difficult to enumerate the GWT RPC calls.

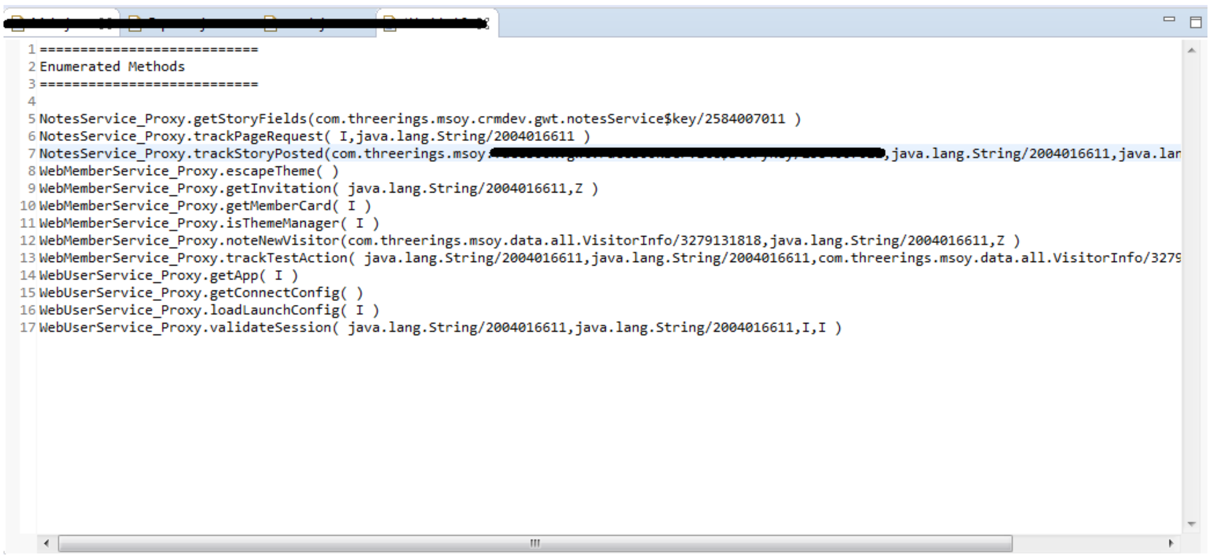
GWT Enum tool is used to list non-functioning vulnerable methods that may not normally be seen usually. GWT JavaScript code is obfuscated to enumerate all exposed method calls.

The tool results are shown in Figure 4.5.

## 4.4 JAVA code scanning using Fortify

Fortify is used to recognize and handle security vulnerabilities in software to reduce security risks. Fortify SCA is a static analysis tool processes the code in a similar manner to code compiler.

Fortify uses an intermediate tool that runs on the code base and converts into compiled code that is optimized for the fortify-analysis.



[redacted]

```
==========================
Enumerated Methods
==========================

NotesService_Proxy.getStoryFields(com.threerings.msoy.crmdev.gwt.notesService$key/2584007011 )
NotesService_Proxy.trackPageRequest( I,java.lang.String/2004016611 )
NotesService_Proxy.trackStoryPosted(com.threerings.msoy[redacted],java.lang.String/2004016611,java.lar
WebMemberService_Proxy.escapeTheme( )
WebMemberService_Proxy.getInvitation( java.lang.String/2004016611,Z )
WebMemberService_Proxy.getMemberCard( I )
WebMemberService_Proxy.isThemeManager( I )
WebMemberService_Proxy.noteNewVisitor(com.threerings.msoy.data.all.VisitorInfo/3279131818,java.lang.String/2004016611,Z )
WebMemberService_Proxy.trackTestAction( java.lang.String/2004016611,java.lang.String/2004016611,com.threerings.msoy.data.all.VisitorInfo/3279
WebUserService_Proxy.getApp( I )
WebUserService_Proxy.getConnectConfig( )
WebUserService_Proxy.loadLaunchConfig( I )
WebUserService_Proxy.validateSession( java.lang.String/2004016611,java.lang.String/2004016611,I,I )
```

Figure 4.5: Enumerating the GWT RPC calls

This tool also uses Fortify customized Rules to analyze the code base, if there are any security violations of secure coding practices. An audit workbench is available for viewing the results and analyzing the issues.

Figure 4.6 describes working of fortify tool and Figure 4.7. shows the scanned results.

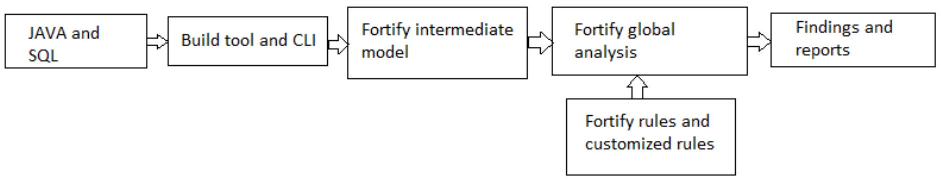
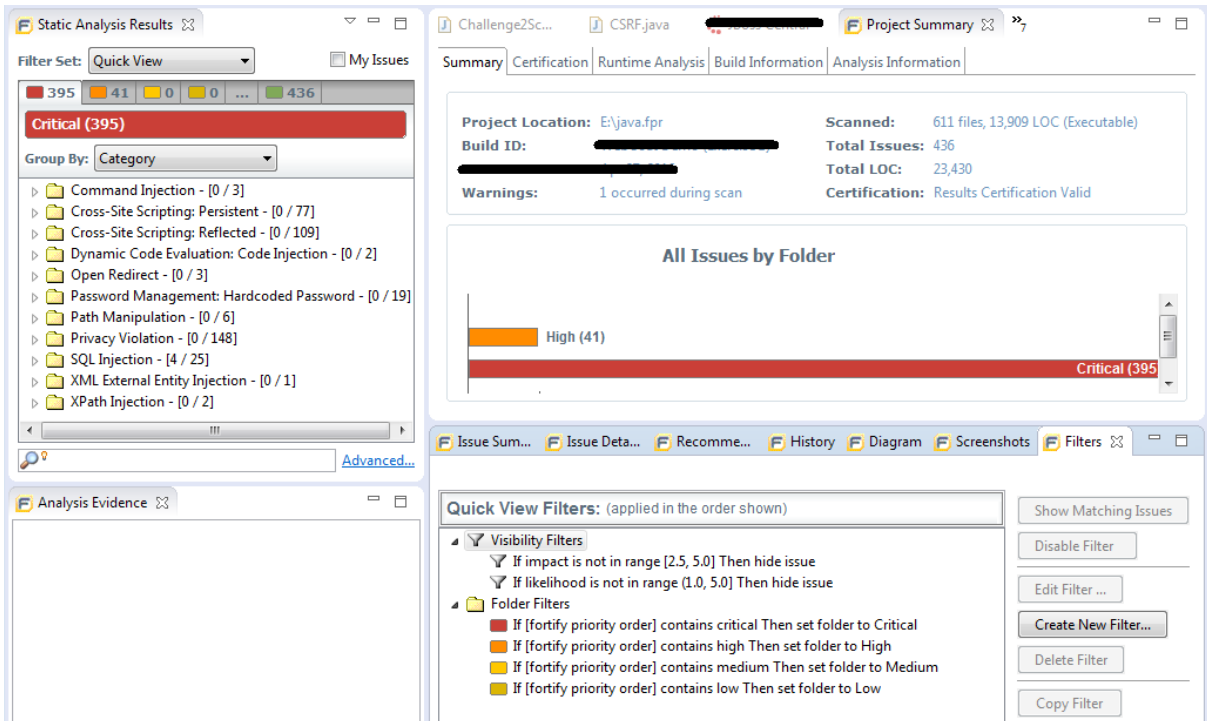


Figure 4.6: Fortify process flow


Static Analysis Results

Filter Set: Quick View    My Issues

🟥 395 🟧 41 🟨 0 🟨 0 ... 🟩 436

Critical (395)

Group By: Category

- Command Injection - [0 / 3]
- Cross-Site Scripting: Persistent - [0 / 77]
- Cross-Site Scripting: Reflected - [0 / 109]
- Dynamic Code Evaluation: Code Injection - [0 / 2]
- Open Redirect - [0 / 3]
- Password Management: Hardcoded Password - [0 / 19]
- Path Manipulation - [0 / 6]
- Privacy Violation - [0 / 148]
- SQL Injection - [4 / 25]
- XML External Entity Injection - [0 / 1]
- XPath Injection - [0 / 2]

Advanced...

Analysis Evidence

Challenge2Sc... | CSRF.java | [redacted] | Project Summary

Summary | Certification | Runtime Analysis | Build Information | Analysis Information

| | | | |
|---|---|---|---|
| Project Location: | E:\java.fpr | Scanned: | 611 files, 13,909 LOC (Executable) |
| Build ID: | [redacted] | Total Issues: | 436 |
| [redacted] | | Total LOC: | 23,430 |
| Warnings: | 1 occurred during scan | Certification: | Results Certification Valid |

All Issues by Folder

High (41)

Critical (395

Issue Sum... | Issue Deta... | Recomme... | History | Diagram | Screenshots | Filters

Quick View Filters: (applied in the order shown)

- Visibility Filters
  - If impact is not in range [2.5, 5.0] Then hide issue
  - If likelihood is not in range (1.0, 5.0] Then hide issue
- Folder Filters
  - 🟥 If [fortify priority order] contains critical Then set folder to Critical
  - 🟧 If [fortify priority order] contains high Then set folder to High
  - 🟨 If [fortify priority order] contains medium Then set folder to Medium
  - 🟨 If [fortify priority order] contains low Then set folder to Low

Show Matching Issues | Disable Filter | Edit Filter ... | Create New Filter... | Delete Filter | Copy Filter

Figure 4.7: Fortify scanning results

# Chapter 5

# Using CSRF Guard in Oracle JET framework

## 5.1 OJET framework

JET framework is a collection of JavaScript libraries, which is used to develop client-side web applications. In this new framework, the CSRF guard is used in order to prevent CSRF attack.

## 5.2 How CSRF Guard works

CSRF guard has a unique design pattern to prevent attacks. A developer can customize the token injection strategy. CSRF guard can be configured based on the requirement, without compromising the usability of the application.

Figure 5.1 describes how CSRF Guard works to prevent CSRF attack. CSRF Guard analyze the incoming requests to protect the web resources. It injects a token to specified resources into the HTML source code and then verifies this token, when user request for the particular resource or HTML page. This validation is done on the server side.

Figure 5.2 shows the token validation process. CSRF Guard checks the page specific token in each request. If page specific token is no yet generated, then it verifies the request by checking per-session tokens and creates page specific tokens. This page specific token will be checked for all subsequent requests.

Figure 5.3 gives the details of token injection process. CSRF Guard initiate the random token and adds it to the web pages that needs a protection.

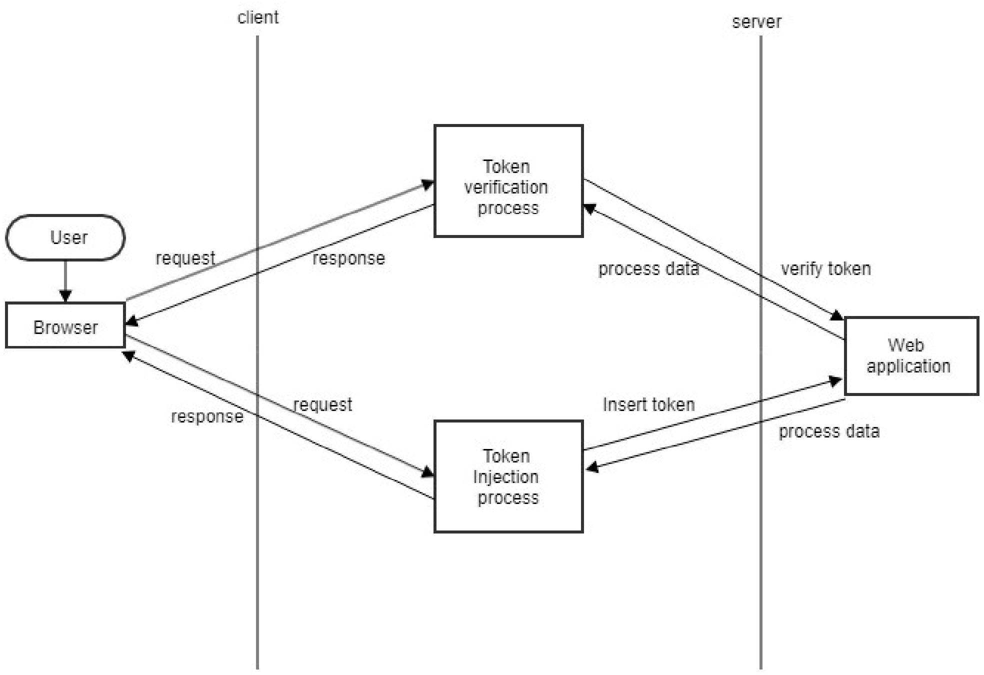


Figure 5.1: How CSRF Guard works

Figure 5.4 and 5.5 demonstrate that CSRF Guard attaches the CSRF token to every single web resources. Figure 5.6 and Figure 5.7 shows that CSRF Guard throws access denied message when unauthorized user access the resources.

## 5.3 Limitations of CSRF Guard

- CSRF token can be stolen By hijacking the session, man in the middle attack or XSS attack.
- Relevant pages for token injection needs to be analyzed. File configurations need to be set manually. But introducing scripts into pages needs manual work.
- Dynamically generated requests are the requests created by scripts while running on the web browsers. That cannot be accessed before running it. CSRF Guard fails to add a token to this type of dynamically created forms.

## 5.4 CSRF token injection approach

CSRF Guard injects the CSRF preention tokens within the HTML pages generated by the application. In an earlier version of CSRF Guard, token injection process was time-

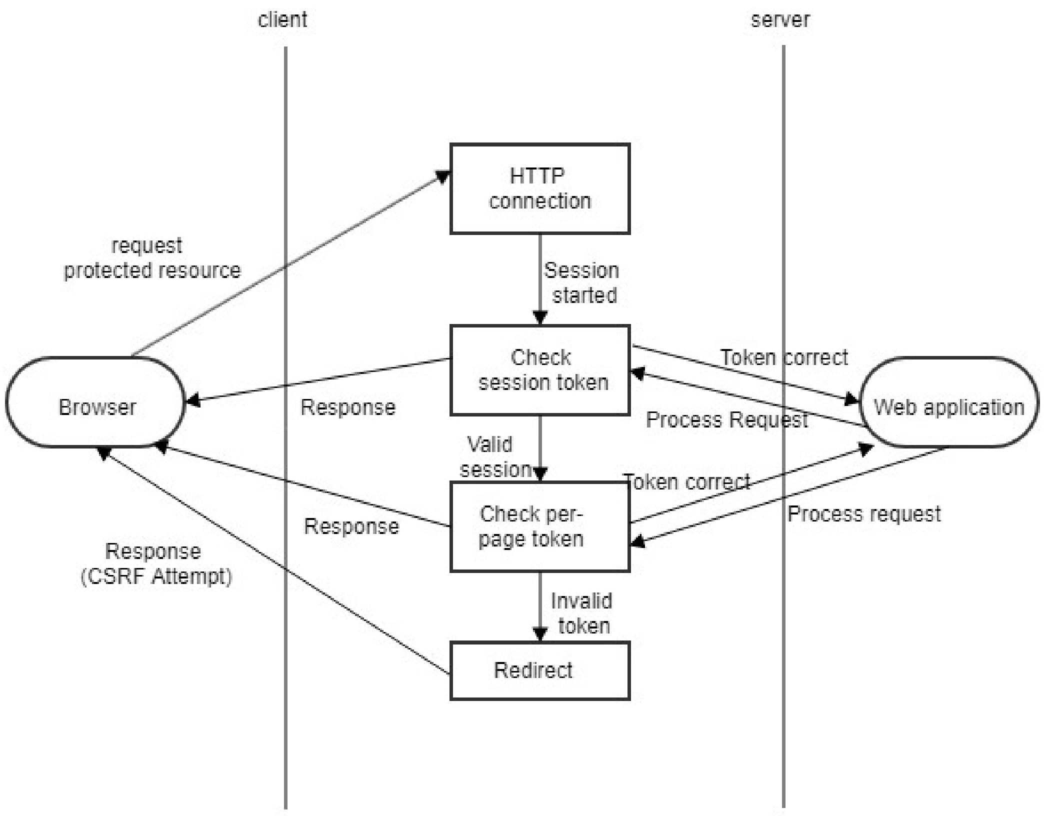


Figure 5.2: CSRF token validation in CSRF Guard

consuming, as file configuration needs to be set manually. and Relevant pages for token injection also needs to be analyzed.

CSRF Guard 3 has a better approach to inject CSRF tokens dynamically. JavaScript DOM manipulation JSP tag library strategies are used as CSRF token injection strategies. This technique requires minimum efforts to inject tokens.

### 5.4.1 DOM manipulation

This technique dynamically plants the CSRF tokens into the DOM that is currently loaded in the browser. Dynamic JS file needs to be added. This solution requires a servlet mapping and adds a JS HTML tags in all the pages sending requests to access protected web resources.

DOM manipulation will not work for a specific application context. JavaScript file used for token injection is created by the servlet file.

Owasp.CsrfGuard.jar this JAR file needs to be added in the application's classpath.

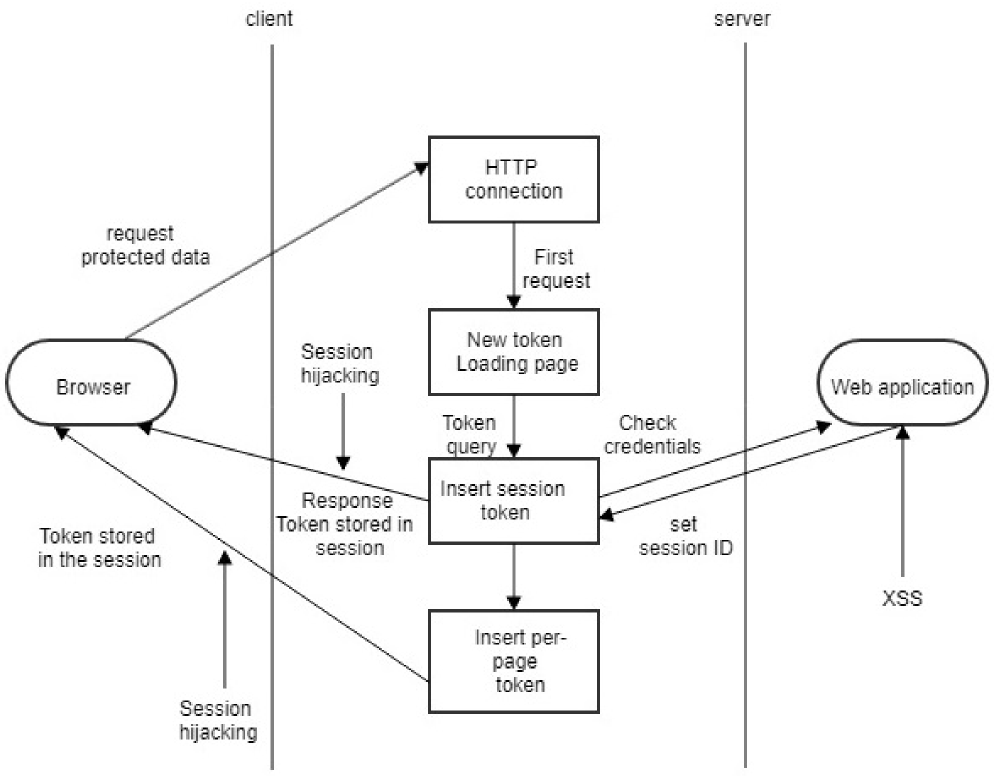


Figure 5.3: CSRF token injection in CSRF Guard

servlet class needs to be declared in the web.xml file. Figure 5.8 displays the servlet properties set. Figure 5.9 displays properties set on dynamically generated JS file.

**Initialization params supported by the JS servlet :**

- source-file

  location of JS template file that needs to be used to automate the process.

- domain-strict

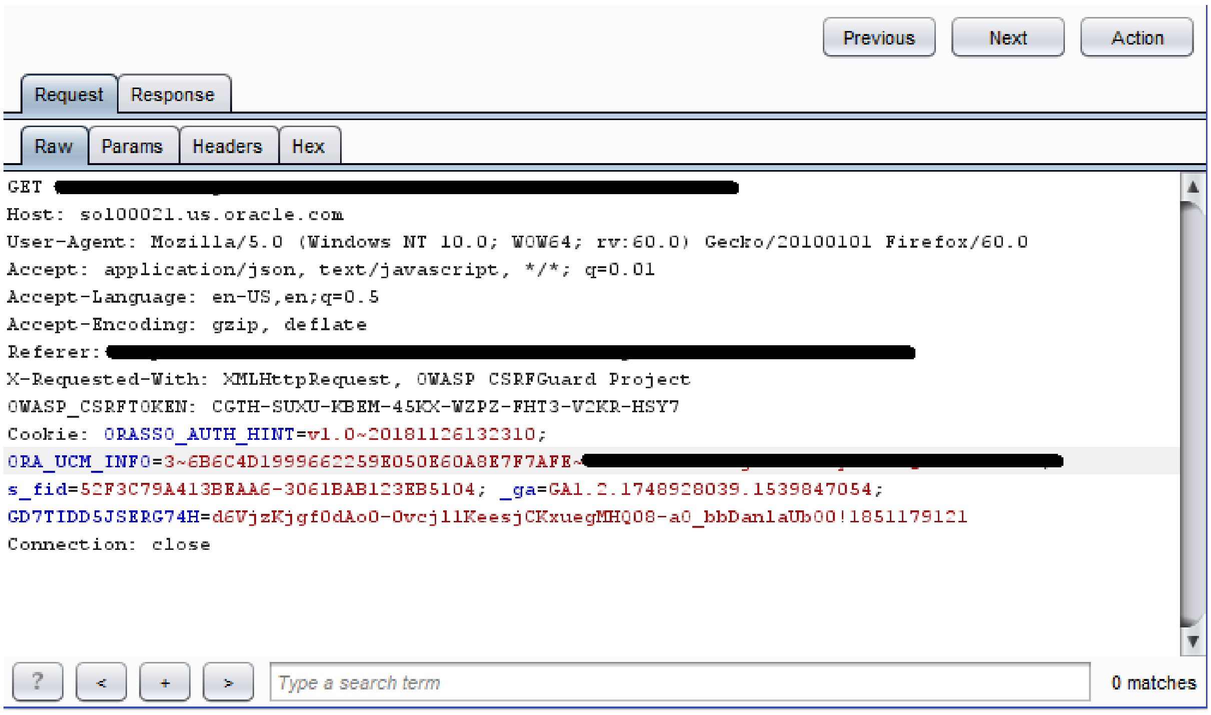
  Boolean value deciding whether or not the JavaScript code should add the prevention tokens in links that point to the same domain from which the HTML is originated.

- referer-pattern

  Requesting the servlet access pages needs to match this particular referer-pattern.

- cache-control

  caching the dynamic JS file reduces network traffic and increase performance.



```
GET [redacted]
Host: sol00021.us.oracle.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: [redacted]
X-Requested-With: XMLHttpRequest, OWASP CSRFGuard Project
OWASP_CSRFTOKEN: CGTH-SUXU-KBEM-45KX-WZPZ-FHT3-V2KR-HSY7
Cookie: ORASSO_AUTH_HINT=v1.0~20181126132310;
ORA_UCM_INFO=3~6B6C4D1999662259E050E60A8E7F7AFE~[redacted]
s_fid=52F3C79A413BEAA6-3061BAB123EB5104; _ga=GA1.2.1748928039.1539847054;
GD7TIDD5JSERG74H=d6VjzKjgf0dAo0-0vcj11KeesjCKxuegMHQ08-a0_bbDan1aUb00!1851179121
Connection: close
```

Figure 5.4: CSRF Guard

- inject-into-forms

  Boolean value describing weather token needs to be added as hidden fields to the HTML.

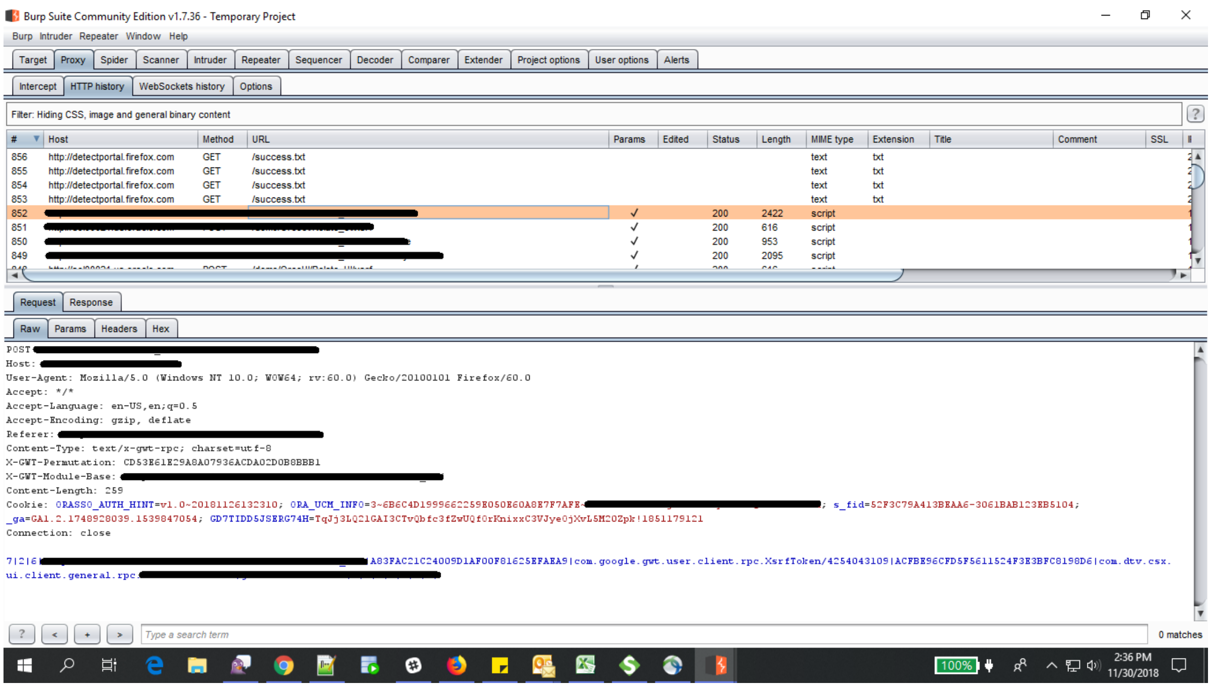
- inject-into-attributes

  determines weather JS should inject token to src or href attributes or not.

All requests originated from the HTML page needs to ensure that accurate CSRF token is submitted for the user's current session. Adding the dynamic JavaScript code does not protect the web page. Rather, the script ensures the CSRF token is transferred within all web requests created by the current page.

This JavaScript code will start an event handler with window.on load. Once the event is triggered, the code will analyze every HTML tag within the DOM looking for either form tags and or tags having href or src attributes.

According to defined params, forms are dynamically updated to include the CSRF-Guard tokens. Figure 5.10 explains the flow for token injection in CSRFGuard.



Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content

| # | Host | Method | URL | Params | Edited | Status | Length | MIME type | Extension | Title | Comment | SSL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 856 | http://detectportal.firefox.com | GET | /success.txt | | | | | text | txt | | | |
| 855 | http://detectportal.firefox.com | GET | /success.txt | | | | | text | txt | | | |
| 854 | http://detectportal.firefox.com | GET | /success.txt | | | | | text | txt | | | |
| 853 | http://detectportal.firefox.com | GET | /success.txt | | | | | text | txt | | | |
| 852 | [redacted] | [redacted] | [redacted] | ✓ | | 200 | 2422 | script | | | | |
| 851 | [redacted] | [redacted] | [redacted] | ✓ | | 200 | 616 | script | | | | |
| 850 | [redacted] | [redacted] | [redacted] | ✓ | | 200 | 953 | script | | | | |
| 849 | [redacted] | [redacted] | [redacted] | ✓ | | 200 | 2095 | script | | | | |

```
POST [redacted]
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: [redacted]
Content-Type: text/x-gwt-rpc; charset=utf-8
X-GWT-Permutation: CD53E61E29A8A07936ACDA02D0B8BBB1
X-GWT-Module-Base: [redacted]
Content-Length: 259
Cookie: ORASSO_AUTH_HINT=v1.0~20181126132310; ORA_UCM_INFO=3~6B6C4D199966225 9E050E60A8E7F7AFE~[redacted]; s_fid=52F3C79A413BEAA6-3061BAB123EB5104;
_ga=GA1.2.1748928039.1539847054; GD7TIDD5JSERG74H=TqJj3LQ21GAI3CTvQbfc3fZwUQf0rKnixxC3VJye0jXvL5M20Zpk!1851179121
Connection: close

7|2|6|[redacted]A83FAC21C24009D1AF00F81625EFAEA9|com.google.gwt.user.client.rpc.XsrfToken/4254043109|ACFBE96CFD5F5611524F3E3BFC8198D6|com.dtv.csx.
ui.client.general.rpc.[redacted]
```

Figure 5.5: CSRF Guard

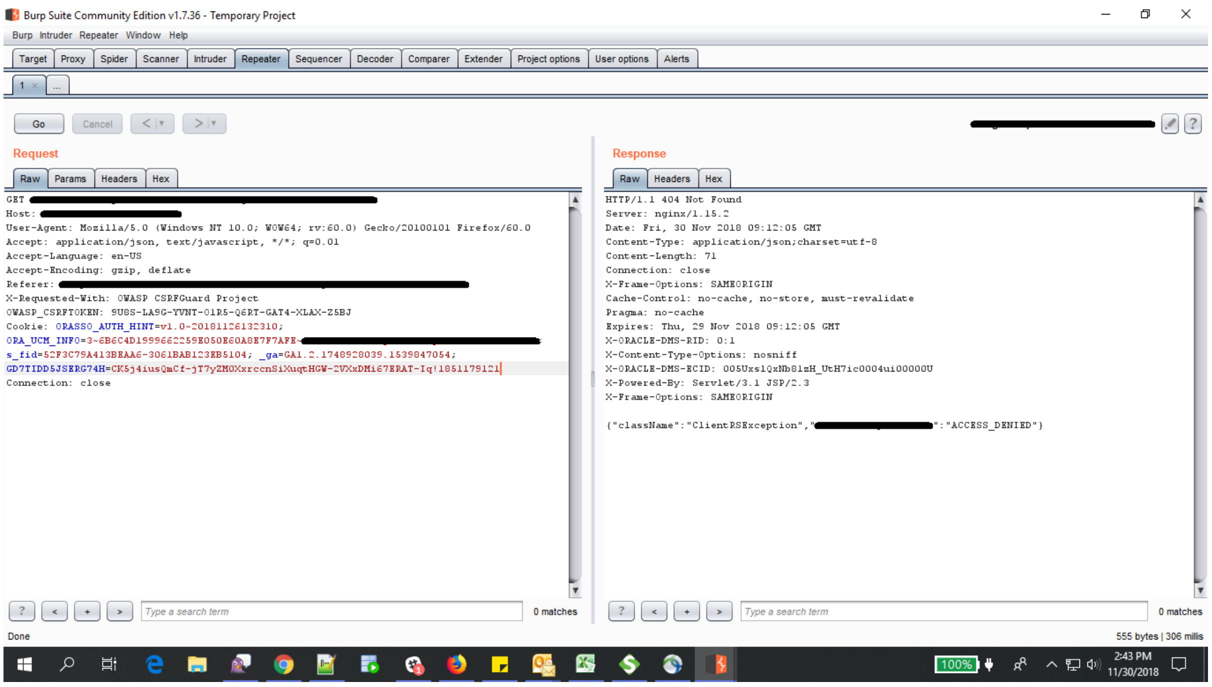
## 5.4.2 JSP tag library

This library allows using of JSP tags that give access to the token name, the token value and the token name-value pair. To use this library, JAR should be added to the classpath. Using this library, the Developer can have control over token injection process and can place the token at an appropriate location. This token details can be obtained using this library.

- Display token name [ Figure 5.11 ]
- Display token value [ token value should be used with URI attribute. The value of the URI attribute is the URI for which the token value will be posted ] [ Figure 5.12 ]
- Display token name and value pair.

**Generated form with the CSRF token :** This JSP library implements a pattern to generate HTML forms with the CSRF prevention token automatically embedded in it. This strategy makes it easy to integrate the CSRF token, for unique token per page model. proper validation and encoding need to be performed.

**Generated link with the CSRF token :** This library generates the HTML anchor tags with the CSRF prevention token automatically added as a query string parameter.


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go
Cancel
Request
Raw Params Headers Hex
Response
Raw Headers Hex
Type a search term
0 matches
Done
555 bytes | 306 millis

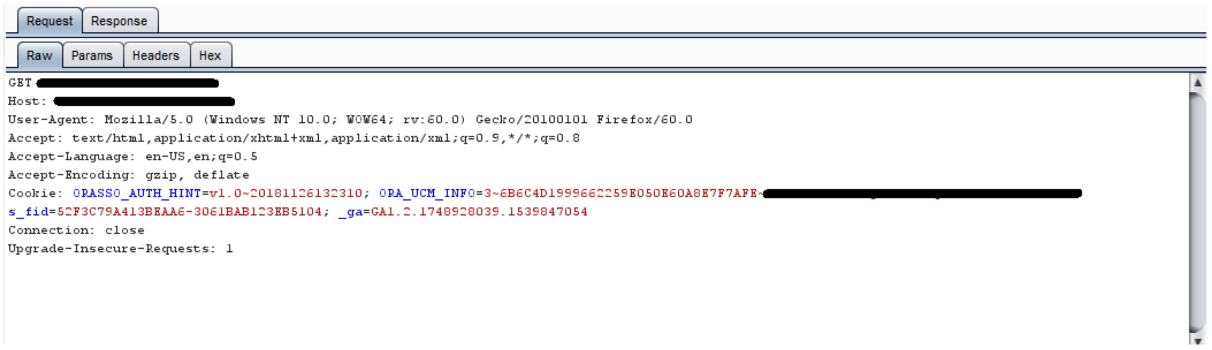
[redacted]

```
GET [redacted]
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US
Accept-Encoding: gzip, deflate
Referer: [redacted]
X-Requested-With: OWASP CSRFGuard Project
OWASP_CSRFTOKEN: 9U8S-LA9G-YVNT-01R5-Q6RT-GAT4-XLAX-Z5BJ
Cookie: ORASSO_AUTH_HINT=v1.0~20181126132310;
ORA_UCM_INFO=3~6B6C4D1999662259E050E60A8E7F7AFE~[redacted]
s_fid=52F3C79A413BEAA6-3061BAB123EB5104; _ga=GA1.2.1748928039.1539847054;
GD7TIDD5JSERG74H=CK5j4iusQmCf-jT7yZMOXxrccnSiXuqtHGW-2VXxDMi67ERAT-Iq!1851179121
Connection: close
```

```
HTTP/1.1 404 Not Found
Server: nginx/1.15.2
Date: Fri, 30 Nov 2018 09:12:05 GMT
Content-Type: application/json;charset=utf-8
Content-Length: 71
Connection: close
X-Frame-Options: SAMEORIGIN
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Thu, 29 Nov 2018 09:12:05 GMT
X-ORACLE-DMS-RID: 0:1
X-Content-Type-Options: nosniff
X-ORACLE-DMS-ECID: 005Uxs1QxNb81zH_UtH7ic0004ui00000U
X-Powered-By: Servlet/3.1 JSP/2.3
X-Frame-Options: SAMEORIGIN

{"className":"ClientRSException","[redacted]":"ACCESS_DENIED"}
```

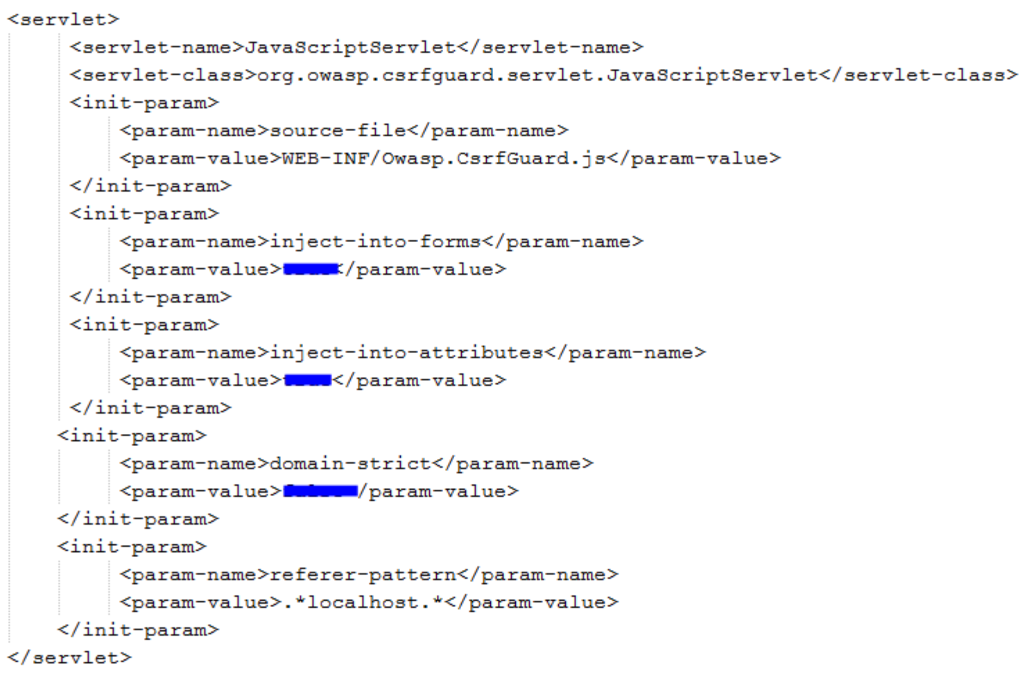
Figure 5.6: CSRF Guard



```
GET [redacted]
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: ORASSO_AUTH_HINT=v1.0~20181126132310; ORA_UCM_INFO=3~6B6C4D1999662259E050E60A8E7F7AFE~[redacted]
s_fid=52F3C79A413BEAA6-3061BAB123EB5104; _ga=GA1.2.1748928039.1539847054
Connection: close
Upgrade-Insecure-Requests: 1
```

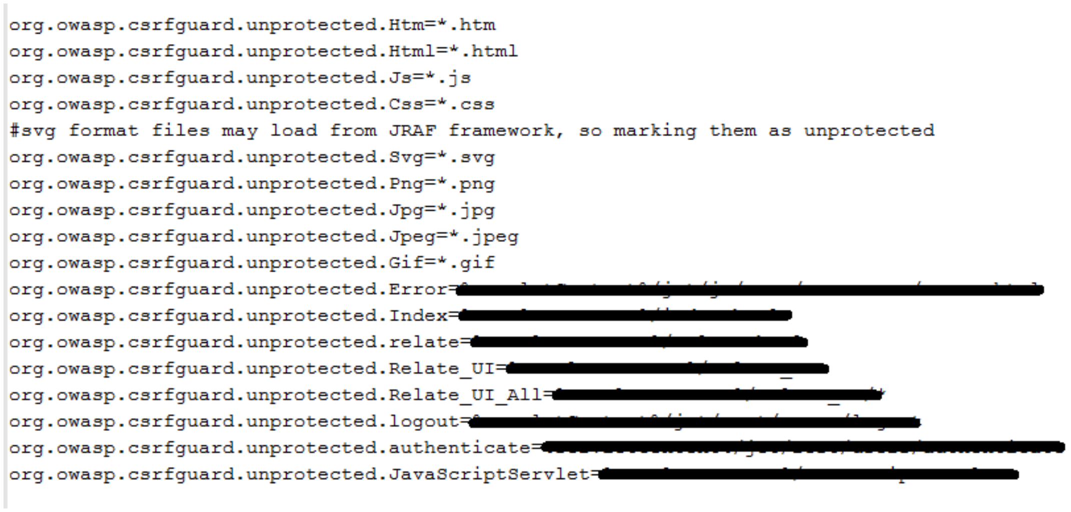
Figure 5.7: CSRF Guard

The tag accepts attribute name-value pairs and simply adds them to the page. As an outcome, you are free to use the same attribute values made available in a standard HTML anchor and the prevention tokens will be added in the link.



```
<servlet>
     <servlet-name>JavaScriptServlet</servlet-name>
     <servlet-class>org.owasp.csrfguard.servlet.JavaScriptServlet</servlet-class>
     <init-param>
         <param-name>source-file</param-name>
         <param-value>WEB-INF/Owasp.CsrfGuard.js</param-value>
     </init-param>
     <init-param>
         <param-name>inject-into-forms</param-name>
         <param-value>[redacted]</param-value>
     </init-param>
     <init-param>
         <param-name>inject-into-attributes</param-name>
         <param-value>[redacted]</param-value>
     </init-param>
    <init-param>
         <param-name>domain-strict</param-name>
         <param-value>[redacted]/param-value>
    </init-param>
    <init-param>
         <param-name>referer-pattern</param-name>
         <param-value>.*localhost.*</param-value>
    </init-param>
</servlet>
```

Figure 5.8: Servlet properties



```
org.owasp.csrfguard.unprotected.Htm=*.htm
org.owasp.csrfguard.unprotected.Html=*.html
org.owasp.csrfguard.unprotected.Js=*.js
org.owasp.csrfguard.unprotected.Css=*.css
#svg format files may load from JRAF framework, so marking them as unprotected
org.owasp.csrfguard.unprotected.Svg=*.svg
org.owasp.csrfguard.unprotected.Png=*.png
org.owasp.csrfguard.unprotected.Jpg=*.jpg
org.owasp.csrfguard.unprotected.Jpeg=*.jpeg
org.owasp.csrfguard.unprotected.Gif=*.gif
org.owasp.csrfguard.unprotected.Error=[redacted]
org.owasp.csrfguard.unprotected.Index=[redacted]
org.owasp.csrfguard.unprotected.relate=[redacted]
org.owasp.csrfguard.unprotected.Relate_UI=[redacted]
org.owasp.csrfguard.unprotected.Relate_UI_All=[redacted]
org.owasp.csrfguard.unprotected.logout=[redacted]
org.owasp.csrfguard.unprotected.authenticate=[redacted]
org.owasp.csrfguard.unprotected.JavaScriptServlet=[redacted]
```

Figure 5.9: CSRFGuard.js properties

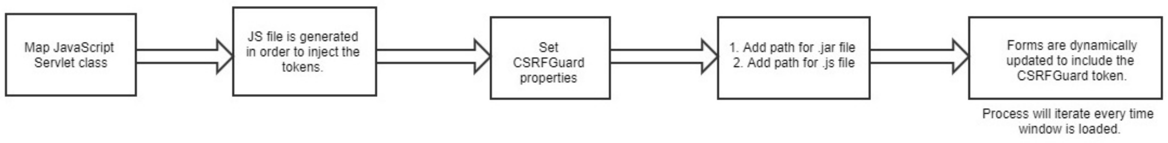
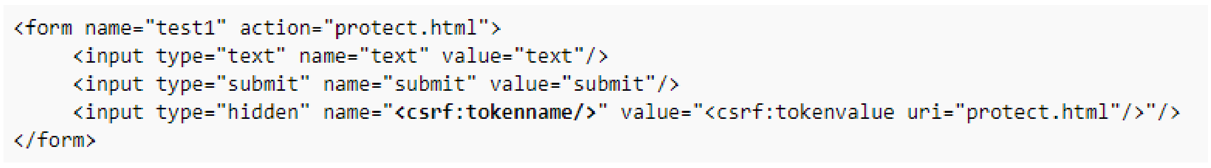
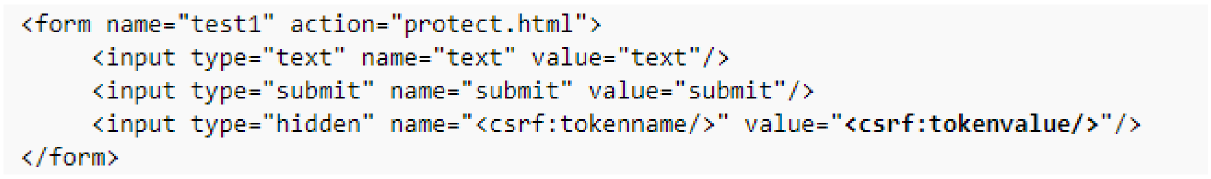

Map JavaScript Servlet class
JS file is generated in order to inject the tokens.
Set CSRFGuard properties
1. Add path for .jar file
2. Add path for .js file
Forms are dynamically updated to include the CSRFGuard token.
Process will iterate every time window is loaded.

Figure 5.10: Dynamic JS generation flow



```
<form name="test1" action="protect.html">
     <input type="text" name="text" value="text"/>
     <input type="submit" name="submit" value="submit"/>
     <input type="hidden" name="<csrf:tokenname/>" value="<csrf:tokenvalue uri="protect.html"/>"/>
</form>
```

Figure 5.11: Exposed token name using JSP tag library



```
<form name="test1" action="protect.html">
     <input type="text" name="text" value="text"/>
     <input type="submit" name="submit" value="submit"/>
     <input type="hidden" name="<csrf:tokenname/>" value="<csrf:tokenvalue/>"/>
</form>
```

Figure 5.12: Exposed token values using JSP tag library

# Chapter 6

# Conclusion

## 6.1 Conclusion

This project includes source code scanning and CSRF attack prevention approach. Existing PMD rules were used to scan the GWT code initially. New PMD rules are added to scan the front end GWT code thoroughly. New fortify rules are created to scan the back end JAVA code. XSRF token approach was used to reduce the CSRF attack risk in GWT application. CSRFGuard is used to prevent CSRF attack in a new JET application. A new technique for token injection is introduced using JSP tag library and DOM manipulation to overcome the limitations of manual CSRF token injection.

# Bibliography

[1] N. Jovanovic, E. Kirda, and C. Kruegel, “Preventing cross site request forgery attacks,” in *2006 Securecomm and Workshops*, pp. 1–10, Aug 2006.

[2] E. D. Alvarez, B. D. Correa, and I. F. Arango, “An analysis of xss, csrf and sql injection in colombian software and web site development,” in *2016 8th Euro American Conference on Telematics and Information Systems (EATIS)*, pp. 1–5, April 2016.

[3] J. You and F. Guo, “Improved csrfguard for csrf attacks defense on java ee platform,” in *2014 9th International Conference on Computer Science Education*, pp. 1115–1120, Aug 2014.

[4] B. Chen, P. Zavarsky, R. Ruhl, and D. Lindskog, “A study of the effectiveness of csrf guard,” in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pp. 1269–1272, Oct 2011.

[5] T. Alexenko, M. Jenne, S. D. Roy, and W. Zeng, “Cross-site request forgery: Attack and defense,” in *2010 7th IEEE Consumer Communications and Networking Conference*, pp. 1–2, Jan 2010.

[6] D. Larson, J. Liu, and Y. Zuo, “Analyzing the vulnerabilities in gwt code and applications,” in *2014 Second International Symposium on Computing and Networking*, pp. 525–530, Dec 2014.