Identification and Illustration of Insecure Direct Object References and their Countermeasures

Ajay Kumar Shrestha
Department of Computer and Electronics Engineering
Kantipur Engineering College
Tribhuvan University, Nepal

Pradip Singh Maharjan
Department of Computer and Electronics Engineering
Kantipur Engineering College
Tribhuvan University, Nepal

Santosh Paudel
Department of Computer and Electronics Engineering
Kantipur Engineering College
Tribhuvan University, Nepal

ABSTRACT
The insecure direct object reference simply represents the flaws in the system design without the full protection mechanism for the sensitive system resources or data. It basically occurs when the web application developer provides direct access to objects in accordance with the user input. So any attacker can exploit this web vulnerability and gain access to privileged information by bypassing the authorization. The main aim of this paper is to demonstrate the real effect and the identification of the insecure direct object references and then to provide the feasible preventive solutions such that the web applications do not allow direct object references to be manipulated by attackers. The experiment of the insecure direct object referencing is carried out using the insecure J2EE web application called WebGoat and its security testing is being performed using another JAVA based tool called BURP SUITE. The experimental result shows that the access control check for gaining access to privileged information is a very simple problem but at the same time its correct implementation is a tricky task. The paper finally presents some ways to overcome this web vulnerability.

General Terms
Web Vulnerability; Authorization

Keywords
IDOR; Web Application; Authorization; Access control; Web exploit

1. INTRODUCTION
OWASP is the Open source Web Application Project which is the result of the survey work from a web application security team which comes up with most critical top ten web application security issues [1]. OWASP group maintains different kinds of security projects including documentation of the top ten web vulnerabilities, teaching tools such as WebGoat, proxies such as web scarab, secure development library such as OWASP Enterprise Security API (ESAPI) etc. In fact, the top ten web application vulnerabilities in the list are the very simple, well-known but dangerous and commonly exploited ones. It is usually considered that the attackers are very well equipped with security knowledge than the web developers. Therefore it is very important that the web developers should develop strong security knowledge on both education and using the tools. OWASP is the best site where the educational materials regarding the web application security can be obtained. And the OWASP ESAPI is the best tool which provides abundance APIs to prevent the new class of web exploits and vulnerabilities to some extent.

This paper presents one of the top ten web application vulnerabilities in terms of the challenges and the procedures to deal with it. The "Insecure Direct Object Reference (IDOR)", was listed as the fourth web vulnerability in the list for year 2013 [1].

The paper demonstrates the vulnerability of the IDOR via tools namely WebGoat and BURP SUITE. After the verification of the vulnerabilities, the real effect and the identification of the insecure direct object references are presented. Finally, it provides the feasible preventive solutions so as to make that the web applications free from direct object references which will be harder for attackers to manipulate.

2. BACKGROUND
2.1 Overview
Basically a direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key, as a URL or form parameter. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data [2]. So the insecure direct object reference simply represents the lacking of the authentication level checks resulting in the incorrect level of administrative access to the system data. This happens when the developers expose data objects via a web application with assumption that the users will always follow the application rules. The insecure direct object reference can be overviewed via the Table 1.

Let's take a case to understand the basic ideas about the insecure direct object reference. Suppose an organization provides financial data report via website to its users who are only authorized to see their own personal financial data report. The web page is designed with the assumption that no user will see other users' or organization's data. For this, it simply uses a report ID which is available in webpage URL (e.g. /accounts/viewDetail?id=0010) or in some hidden field. In this scenario, since report ID is obviously predictable, anyone can change the value of ID and resubmit the request in order to get the data of other users as well (like altering the content as /accounts/viewDetail?id=0012).

Table 1. Table captions should be placed above the table

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
_____	**Exploitability EASY**	**Prevalence COMMON**	**Detectability EASY**	**Detectability EASY**	_____
Consider the types of users of your system. Do any users have only partial access to certain types of system data?	Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user isn't authorized for. Is access granted?	Applications frequently use the actual name or key of an object when generating web pages. Applications don't always verify user is authorized for the target object. This result in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws and code analysis quickly shows whether authorization is properly verified.		Such flaws can compromise all the data that can be referenced by the parameter. Unless the name space is sparse, it's easy for an attacker to access all available data of that type.	Consider the business value of the exposed data. Also consider the business impact of public exposure of the vulnerability.

The report ID in this case is the direct reference to the object which is the record in the database table. This is the insecurity due to very basic of the direct object referencing when no protection mechanism is incorporated in the system. This is also termed Directory Traversal.

Basically the Directory Traversal and Open Redirects are the two classic examples of IDOR web vulnerabilities. For the Directory Traversal, usually a web application that allows a file to be rendered to a user is stored on the local machine. If no verification is done by the application about accessing a whichever file, an attacker can request other files on the file system and those will also be displayed. For instance, if the attacker notices the URL:

http://misc-security.com/file.jsp?file=report.txt

The attacker could modify the file parameter using a directory traversal attack by modifying the URL to:

http://misc-security.com/file.jsp?file=**../../../etc/shadow**

This causes the file.jsp to return and render /etc/shadow file thereby demonstrates the page is susceptible to a directory traversal attack. However, in the Open Redirects flaws, the web application with a parameter can allow the website to redirect the user somewhere else. If there is no proper implementation of parameter through a white list, attackers can use this in a phishing attack to lure potential victims to a chosen site.

2.2 Literature Review

With the insecure direct object references, all of the exposed web application frameworks are vulnerable to some sort of attacks [3]. Many of the poorly designed applications reveal the internal object references to the users. The references basically point to the file systems and databases. An attacker can use the tampering process to change the references and exploit the access control mechanism. Just take an example, if the code allows user input to specify paths or filename, any attacker may then jump out of the directory of that application and can access the other resources. The code as shown below can be attacked using a null byte injection via a string like "../../../etc/passwd%00". The primary objective of this attack is to access any resources on the file system of the web server [3].

```
<select name="language"><option value="fr">Français
</option></select>
…
require_once ($_REQUEST['language']."lang.php");
```

N. Antunes et al. mentioned in their paper about the SQL injection and cross-site scripting (XSS) vulnerabilities as the two most common risks in the Web application [4]. SQL injection basically enables the attackers to alter SQL queries sent to a database. XSS vulnerability on the other hand exists when an application sends user-supplied data to a Web browser without first validating or encoding that content. These web vulnerabilities allow attacker to access the critical data and resources.

Similarly, R. Eran et. al presented a method for detecting security vulnerabilities in a web application [5]. Their paper included an analysis of the client requests and server responses so as to discover pre-defined elements of the application's interface with external clients and the attributes of those elements. The client requests were then mutated as per the pre-defined set of mutation rules and then the unique exploits were generated. Overall, they used the exploits to attack the web application and the results of the attack were evaluated for anomalous application activity.

L. Shar et al. in their paper had proposed the use of a set of hybrid of the static and dynamic code attributes which characterize input validation and input sanitization code patterns [6]. They are in fact expected to be the most important indicators of web application vulnerabilities. And most importantly, both techniques can be used to extract the proposed attributes in an accurate and scalable way since the static and dynamic program analyses complement each other.

A paper [7] by N. ElBachir El Moussaid et al. provided a survey of web application attacks and vulnerabilities. To improve the web application security, they proposed a method using intrusion detection system (IDS) and scanners based on machine learning and artificial intelligence. They emphasized

an idea "when it comes to vulnerability; it is also an attack which exploits this vulnerability", so they presented web IDS which was based on detection of web vulnerabilities. They used HTTP simulations in a network and the corresponding responses of HTTP requests after sending them to a bunch of websites and applications in order to get experimental results which they used to test the efficiency of their IDS.

C. Yang et al. in their paper [8] had presented a work to provide the ability of web attack detection for IDS namely Snort, by implementing the web attack detection engine using the Core Rule Sets of an open source Web Application Firewall "ModSecurity". They modify the Snort IDS to load Core Rule to detect web attack behaviors viz. Insecure Direct Object References.

Furthermore, M. Jensen et al. performed exemplary attacks on widespread Web Service implementations as a proof of the practical relevance of the threats via a survey of vulnerabilities in the context of Web Services [9]. In addition to that, they also discussed a general countermeasure for the possible prevention and mitigation of such attacks.

However, our paper apparently demonstrates the identification and illustration of the typical web vulnerability which is an insecure direct object references and then it provides the feasible countermeasures such that the web applications will not allow direct object references to be manipulated by attackers. The experiment of the insecure direct object referencing and its corresponding security testing was performed using the JAVA based tools.

3. PROCEDURES AND RESULTS

The practical work was carried out using WebGoat and BURP SUITE together to verify the vulnerability with the insecure direct object reference. WebGoat is the deliberately insecure J2EE web application designed to teach web application security concepts. It has built-in tools that can be used to test the experiment for the Insecure DOR. BURP SUITE is also a Java based application used for performing security testing of web applications.

3.1 Verifying the Vulnerability

BURP SUITE was first configured by going into the connection setting and then the HTTP Proxy was manually set as the local IP address of the machine, which was 127.0.0.1 and also Port value was set to 8080. Then Intercept tab was pressed to see its status. It was at ON state (by default). It is better to change the state to OFF mode at first. Then the task of breaking the access control mechanism was performed in order to access the resource, which was not in the listed directory of the WebGoat.

As a guide, WebGoat said that one xml file "Tomcat/conf/tomcat-users.xml" could be the possible options. But it is just the guess that could work here if there was not any access control check in the web application. In fact, the user of the WebGoat has the access to all the listed files in the lesson_palns/English directory as shown in Figure 1. In our case, the file named "NewLesson.html" was chosen to see the result. The "NewLesson.html" file was selected and then the "View File" tab was pressed. The result obtained was shown in Figure 2. It was a comment in the red showing the hint about the location of the chosen file, which was:

/owaspbwa/owaspbwa-svn/var/lib/tomcat6/webapps/WebGoat/lesson_plans/English/NewLesson.html

This apparently shows that the file has the ninth directory beforehand.

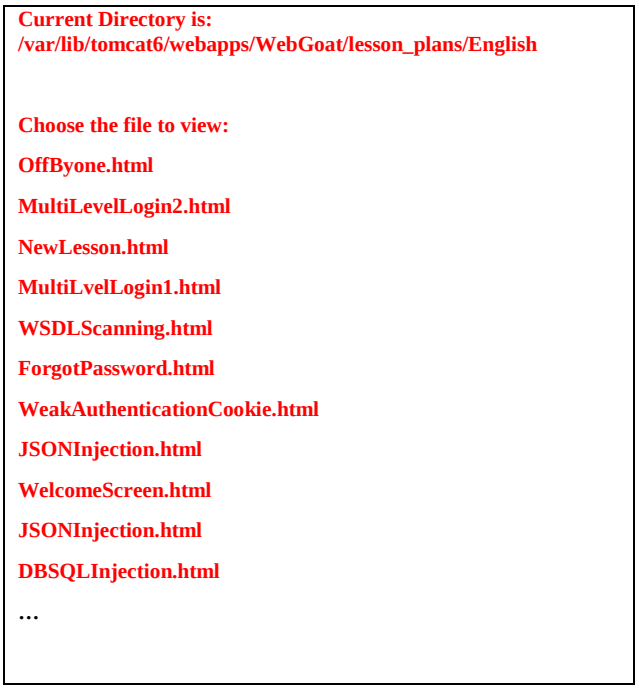

Figure 1: WebGoat Access field

Figure 2: Result on WebGoat page

Then some intercepting attempt was carried out. The intercept tab of BURP SUITE was turned on and again the same process of viewing the NewLesson.html file was carried out. The BURP SUITE was opened to see the result as shown in Figure 3. The File field had the value "NewLession.html".

Now one can just guess and try to add some other directory level at the File field to see the possible error options (not sure) that may be helpful to achieve the target. As shown in Figure 4, the "../" was added randomly to see the page for one directory beforehand and forward the intercept to get the result.

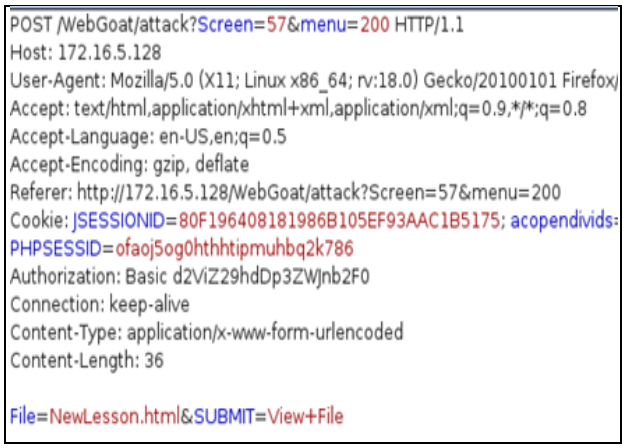

Figure 3: Result page in BURP SUITE

```
POST /WebGoat/attack?Screen=57&menu=200 HTTP/1.1
Host: 172.16.5.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox/
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
Referer: http://172.16.5.128/WebGoat/attack?Screen=57&menu=200
Cookie: JSESSIONID=80F196408181986B105EF93AAC1B5175; acopendivids=
PHPSESSID=ofaoj5og0hthhtipmuhbq2k786
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

File=../&SUBMIT=View+File
```

Figure 4: Result page in BURP SUITE

Solution Videos **Restart this Lesson**

The 'webgoat' user has access to all the files in the lesson_plans/English directory. Try to break the access control mechanism and access a resource that is not in the listed directory. After selecting a file to view, WebGoat will report if access to the file was granted. An interesting file to try and obtain might be a file like tomcat/conf/tomcat-users.xml. Remember that file paths will be different if using the WebGoat source.

***Access to file/directory "/owaspbwa/owaspbwa-svn/var/lib /tomcat6/webapps/WebGoat/lesson_plans" denied**

Figure 5: Result of the WebGoat page

It was a good guess as it displayed the access denied error message for the previous directory "lesson_plans" as shown in the Figure 5, which means that it was the eight directories at that point and with the one directory afterward, we got that we were in the ninth directory level. And since the page was accessed, it showed that there was nothing like access control check in that web app.

So, again another guess was made to get the file "tomcat-users.xml". For this, BURP SUITE was opened again and nine directories were added before etc/tomcat6/ which is the default address for the tomcat-users.xml [10] and it was forwarded as shown in the Figure 6.

```
POST /WebGoat/attack?Screen=57&menu=200 HTTP/1.1
Host: 172.16.5.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox
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
Referer: http://172.16.5.128/WebGoat/attack?Screen=57&menu=200
Cookie: JSESSIONID=80F196408181986B105EF93AAC1B5175; acopendivids
PHPSESSID=ofaoj5og0hthhtipmuhbq2k786
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

File=../../../../../../../../../etc/tomcat6/tomcat-users.xml&SUBMIT=View+File
```

Figure 6: Result page in BURP SUITE

```
Viewing file:/owaspbwa/owaspbwa-svn/etc/tomcat6/tomcat-users.xml

<xml version='1.0' encoding='utf-8'?>

<tomcat-users>
<role rolename="webgoat_basic"/>
<role rolename="webgoat_admin"/>
<role rolename="server_admin"/>
<role rolename="webgoat_user"/>
<role rolename="tomcat"/>
<role rolename="role1"/>
<role rolename="standard"/>
<role rolename="manager"/>
<role rolename="admin"/>


<user username="root" password="owaspbwa" roles="manager,admin,webgoat_admin"/>
<user username="server_admin" password="owaspbwa" roles="server_admin"/>
<user username="admin" password="owaspbwa" roles="admin,manager"/>


<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" fullName=""/>
<user username="role1" password="tomcat" roles="role1"/>


<user username="guest" password="guest" roles="webgoat_user"/>
<user username="user" password="user" roles="webgoat_user"/>
<user username="webgoat" password="webgoat" roles="webgoat_admin"/>
<user username="basic" password="basic" roles="webgoat_user,webgoat_basic"/>
```

Figure 7: tomcat-users.xml file

The attempt was successful. The tomcat-users.xml file was then chosen to see its contents which are shown in Figure 7. It proved that in the absence of the access control check, the content which is supposed to be not available to any user, can actually be achieved without authorization by manipulating the direct object references.

3.2 Identifying and Preventing Vulnerability

Current vulnerability prediction techniques rely on the availability of data labelled with vulnerability information for training. For many real world applications, past vulnerability data is often not available or at least not complete. The usual approaches to verify the security are the automated and the manual approaches. The automated approach with the use of vulnerability scanning tools have difficulty in identifying which parameters should be given an access control check before being implemented and to exploit this vulnerability, it is not enough to only identify the flawed interface but it is also necessary to predict the pattern in order to identify a secure object like Filename etc. Thus identifying this vulnerability is slightly more difficult than other vulnerabilities by using the automation tools. But the manual approach involves the code review process that can identify the critical parameters and then verify that the object is not directly accessible based on predictable factors like User Id, Customer Id, Email Id , Predictable File names or obvious object names like Financial Reports named with organization or client name. But these are all time consuming approaches.

So in order to prevent Insecure Direct Object References, it is very essential to minimize user ability to predict object IDs/Names by using the hard-to-guess numbers. It is always better not to expose the actual ID/name of objects assuming that attackers will find all sensitive pages and folders. And most importantly make the user authenticate each time when sensitive objects/files/contents are accessed. For example, an attacker can retrieve some contents from Database for a particular customer using the query as shown below [3]:

```
SELECT * FROM FinancialReports WHERE CustomerID=
"123"
```

If the user can manipulate the CustomerID from end user interface then she may pass a different ID to access reports of other customers. Developer may easily add validation in SQL by checking the user authorization as shown below:

```
SELECT * FROM FinancialReports INNER JOIN
ReportAccessControlbyOrg On
ReportAccessControlbyOrg.OrgId = FinancialReports.OrgId

WHERE CustomerID="123″ AND
ReportAccessControlbyOrg.OrgId="loggedInUser_OrgId"
```

However if contents are stored outside the Database e.g. File System, then developer may also require to ensure that other methods of file retrieval are also protected since the user may get access to the objects by FTP, Path Traversal vulnerabilities, direct HTTP requests.

Another option is to use an Indirect Reference Map so that developer can create an alternative ID for server side object such as GUID (Globally Unique Identifier) what can be exposed to the outsiders in a safe way because of its complexity. For implementing the indirect reference, the original easy to guess IDs are mapped to GUID, lookup is stored in a dictionary and the dictionary is stored in a session variable. This basically ensures that the mapping is available in the current session and only for the current user. This will not expose the exact ID of the object to the end user.

There is also an open source mapping interface called "AccessReferenceMap" class in OWASP ESAPI. The object of this class is used to create and store the dictionary in the session, and enables us to add or delete the items from the dictionary, and to translate the IDs into the GUID and vice-versa. Since the object ID may be anything like file name or an internal user ID, Indirect Reference Map always maintains a non-sequential & random identifier for a server side resource. Therefore the end user may only see the alternate ID (GUIDs) but not the actual object's ID. This sort of mapping can either be stored temporarily in server cache or permanently in a mapping table. What basically happens in this approach is that when the user tries to retrieve the information from the IDs, the HTTP Post, for example in the AJAX call, contains the alternate ID (indirect reference GUID), which is mapped to the ID value before retrieving any data and return it to the caller. If the call was an attack then the service might not be able to access the mapped value from the AccessReferenceMap. Thus the genuine calls and the attacks can be distinguished.

4. CONCLUSION

It is clear that the attackers can exploit the web vulnerability on insecure direct object reference by simply requesting the content after manipulating the object reference. The insecure direct object reference occurs in web pages when the application uses the actual name or the key of an object and the application doesn't authenticate the user for the target object. So the attackers may try a portion of the parameters such as URI and manipulate it to detect the flaws and then observe the results. They can also play with the http request itself that includes the cookies, forms fields including hidden fields.

It is also seen that the common implementation areas where the objects may be exposed are URL & Links, Hidden Variables, Drop down List box, Unprotected View State (ASP.NET), JavaScript Array and client side objects like Java Applets.

By using an alternate or a random object ID, the developer can just minimize the ability of user to predict resource identifiers. This will certainly reduce capability of any end user to attack but it will not completely avoid the attack. If the attackers may get knowledge of the alternative object ID (such as IDs from browser history on a shared computer) then they can send resource request in legitimate manner to attempt an exploit. So it is very important to check and confirm the authenticity of user while requesting the resources. It is usually easier to implement such scenarios with database based validations than with an application code.

And for the system implementing the AccessReferenceMap, even if the attackers requested for another value by substituting the user ID, their attempt to retrieve data would be unsuccessful. This is because the indirect references are being passed in this mapping but not direct references. Even if they knew the ID for another user, they still had to pass a valid GUID, which must not only exist in the session level dictionary, and must also specifically relate to the legitimate user. However, as far as using the AccessReferenceMap (indirect reference map) is concerned, it requires more coding and needs to pass some extra information to the application so as to maintain the access to the reference maps. Further research can be performed in future on the "per-page authorization" that may not require extra bit of coding and may not need to pass the information around the application for maintaining the access to the reference map. Basically per-page authorization is similar to the AccessReferenceMap but here the appropriate authorization should already be made on the master page [11]. The benefit with this approach is that when the end users or attackers tamper the data and send to the detail page, they will still be unable to see the other's data.

As the direct object access is a significant issue in many of web applications today, it can simply be corrected. Insecure direct references occur when the end users find out the internal conventions and request the page with the manipulated parameters. To overcome this, the conventions are to be made very tough to guess and manipulate and most importantly all the sensitive requests should be placed behind access control check.

5. ACKNOWLEDGMENTS

The authors gratefully acknowledge helpful discussion with C. McLean.

6. REFERENCES

[1] Owasp.org, 'Category:OWASP Project - OWASP', 2015. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_ Project. [Accessed: 20- Sep- 2014].

[2] Owasp.org, 'Top 10 2010-A4-Insecure Direct Object References - OWASP', 2015. [Online]. Available:

https://www.owasp.org/index.php/Top_10_2010-A4. [Accessed: 20- Sep- 2014].

[3] Owasp.org, 'Top 10 2007-Insecure Direct Object Reference - OWASP', 2015. [Online]. Available:

https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference. [Accessed: 20- Sep- 2014].

[4] N. Antunes and M. Vieira, 'Defending against Web Application Vulnerabilities', *Computer*, vol. 45, no. 2, pp. 66-72, 2012.

[5] R. Eran, El. Yuval, R.Gil and T. Tom, 'System for determining web application vulnerabilities', US 6584569 B2, US 09/800,090, 2003.

[6] L. SHAR, L. Briand and H. Tan, 'Web Application Vulnerability Prediction using Hybrid Program Analysis and Machine Learning', IEEE Trans. Dependable and Secure Comput., pp. 1-1, 2014.

[7] N. ElBachir El Moussaid and A. Toumanari, 'Web Application Attacks Detection: A Survey and Classification', International Journal of Computer Applications, vol. 103, no. 12, pp. 1-6, 2014.

[8] C. Yang and C. Shen, 'Implement Web Attack Detection Engine with Snort by Using Modsecurity Core Rules', The E-Learming and Information Technology Symposium Tainan, Taiwan, 1 April, 2009.

[9] M. Jensen, N. Gruschka and R. Herkenhoner, 'A Survey of Attacks on Web Services', Computer Science – Research and Development, vol. 24, no. 4, pp. 185-197, 2009.

[10] Wiki.archlinux.org, 'Tomcat - ArchWiki', 2015. [Online]. Available: https://wiki.archlinux.org/index.php/Tomcat. [Accessed: 20- Sep- 2014].

[11] J. Melton, 'The OWASP Top Ten and ESAPI – Part 4 – Insecure Direct Object Reference : John Melton's Weblog', Jtmelton.com, 2015. [Online]. Available: http://www.jtmelton.com/2010/05/10/the-owasp-top-ten-and-esapi-part-5-insecure-direct-object-reference/.