International Journal of Engineering & Technology, 7 (4.15) (2018) 130-134

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Preventive Measures for Cross Site Request Forgery Attacks on Web-based Applications

Emil Semastin[1], Sami Azam[1]*, Bharanidharan Shanmugam[1], Krishnan Kannoorpatti[1], Mirjam Jonokman[1], Ganthan Narayana Samy[2], Sundresan Perumal[3]

[1] College of Engineering, IT and Environment, Charles Darwin University, Australia
[2]Advanced Informatics School, Menara Razak, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia
[3]Faculty of Science and Technology, Universiti Sains Islam Malaysia (USIM), Negeri Sembilan, Malaysia
**Corresponding author E-mail: sami.azam@cdu.edu.au*

Abstract

Today's contemporary business world has incorporated Web Services and Web Applications in its core of operating cycle nowadays and security plays a major role in the amalgamation of such services and applications with the business needs worldwide. OWASP (Open Web Application Security Project) states that the effectiveness of security mechanisms in a Web Application can be estimated by evaluating the degree of vulnerability against any of the nominated top ten vulnerabilities, nominated by the OWASP. This paper sheds light on a number of existing tools that can be used to test for the CSRF vulnerability. The main objective of the research is to identify the available solutions to prevent CSRF attacks. By analyzing the techniques employed in each of the solutions, the optimal tool can be identified. Tests against the exploitation of the vulnerabilities were conducted after implementing the solutions into the web application to check the efficacy of each of the solutions. The research also proposes a combined solution that integrates the passing of an unpredictable token through a hidden field and validating it on the server side with the passing of token through URL.

Keywords: *CSRF; CSRF Prevention; CSRF Tester; Hidden Token; Web Application Vulnerabilities*

1. Introduction

Securing information is a big challenge in web applications. A well-secured web application should never allow information to slip into the hands of an unauthorized user. In this modern era, information sharing over the Internet is every day's business. Keeping the data secure means preventing misuse due to unauthorized access. But this task alone is becoming harder to achieve every day due to the easy transferability of digital information [1].

Since Web Applications are widely used for both business and personal purposes, the security testing of such applications has a high importance. The primary benefit of using Web Applications in day-to-days business is that it makes the transactions considerably faster without disruptions [2]. Unfortunately, no standardized testing methodologies have been developed yet to test the security of these applications [3]. The research by White Hat claims that over sixty percent (60%) of the total web applications are exposed to multiple vulnerabilities which any attacker can take advantage of. These statistics are quite worrisome and it goes without saying that this rate must come down markedly [4]. Suitable and robust security policies should be implemented during development of any Web Application. Web Applications require more attention than normal computer applications because they are much more sensitive and highly prone to online attacks. A security loophole or a weakness in a Web Application, which a hacker can misuse to carry out an attack, is known as vulnerability [5]. Vulnerabilities are generally classified into two groups: Technical and Logical.

One of the best-known Web Application vulnerabilities is Cross Site Request Forgery (CSRF). The aim of this research is to analyze the different available tools for CSRF testing. The following section highlights the problem statement, followed by a comparison of the methods and in-depth analysis of the solutions available. On the basis of different tests and analysis, the most effective solution will be identified.

CSRF is an attack that transpires when an attacker forces an authenticated user of a Web Application to execute an unwanted operation in that application which results in exploiting the trust relation of a website. These attacks are known by many other names, such as Confused Deputy, One Click Attack, Sea Surf, Session Riding and XSRF [6].

Many Web Applications are not fully protected, especially against the CSRF vulnerability. Cross Site Request Forgery is one of the top 10 vulnerabilities selected by the OWASP and is ranked in the third in the sense of severity after SQL Injection and Cross Site Scripting [7]. Another interesting fact is that these attacks, if they are executed properly, can be very damaging for the Web Applications. The State Changing functionality remains the main focus of the attackers.

These strikes, which are mainly aimed towards the users of a web application rather than the web application itself, intend to obtain the sensitive data of the users. They are commonly known as social engineering attacks [8]. Most developers believe that normal security measures, such as adding cookies with the requests, can prevent Web Applications from these attacks. Unfortunately, this solution is not effective enough because the hacker can get access to all the sessions related information with the aid of different technologies currently available. As a result, full protection for state changing functionalities in Web Applications is not available. The best solution for each application should be selected according to the purpose of the application. This is the focus of this research [9].

Enterprise applications which allow sharing the real time data over the internet have been introduced as a new medium to promote the growth of companies. To increase the security, a number of subsystems need to be installed to protect the data across the communication. Organizations should regularly adopt new technologies to enhance their data security and protect their highly sensitive data from unauthorized access.

2. Cross Site Request Forgery Attacks

CSRF is often referred as the "Sleeping Giant" among the critical vulnerabilities found in Web applications [10]. These attacks can often cause havoc because the developers and a testing squad of a website neglect them and fail to put the protective measures against these attacks in place. The academic and technical discussions on these attacks are relatively limited and they have therefore not been a part of the threat classification of web security. Most web development companies take it for granted that when there is protection against Cross Site Scripting attacks, CSRFs are also prevented automatically. Unfortunately, numerous web developers are totally unaware that CSRF and XSS (Cross-Site Scripting) attacks are entirely different kinds of threats. Compared to other major attacks, CSRF is fairly easy to detect, exploit and prevent. The only requirement is that the developers should know how and where this attack happens.

The CSRF attack is executed by bypassing a request through the user's browser. The attacker misuses the user's belief in a website. The policy of the browser which handles the security allows it to forge a request to any website without problems. This gives the hacker the opportunity to control the browser according to his or her wish.

Even if the attacker is not able to send any request to the victim's web application, because the attacker is beyond the firewall of the user, he or she can still make the user send the request from the user's browser, which is in the firewall. Since each request carries the session information of the user, the attack will be successful even though the user is beyond the firewall [11].

An attacker to perform attacks varying in severity and target can use CSRF. The most popular type of attack is exploiting the valid session ID that a user has for a particular web application. Improper protection against CSRF from the web application results in these types of attacks. Even if the applications of companies of market moving capacity are protected against CSRF, there are incidents which indicate the security gaps are still present. Although CSRF attacks are popularly used to attack the Web Applications, hardware devices like printers, routers, switches etc. can also be compromised [12].

Attackers have exploited the CSRF vulnerability in the recent past. This happens only because the developers of the Web Applications are not cautious enough to implement preventive measures against CSRF when they develop the applications. Attacks, which occurred in 2016 with the Belkin routers and Agora Wallet, are examples of such instance [13, 14].

3. Critical Factors for a Successful Attack

The first and foremost condition for a CSRF attack to be successful is that the user should be authenticated to the victim's website. The next step is to provoke the client to visit the malicious website. This can be the attacker's own website or a website which is under the control of the attacker [15]. If the server of a website is vulnerable to CSRF and it accepts the GET request, the attack becomes much easier. The attacker can use a simple image tag to perform the operation. In the case of the POST requests, the image tag would not work but the attack could still be carried out with the help of a simple code in JavaScript that can be used to submit the FORM tag automatically [16].

The ranking of CSRF in OWASP top 10 vulnerabilities proves that the attack is still critical and the need for an effective solution is still a top priority. The threat classification released by the Web Application Security Consortium ranks CSRF 9th in the list of critical vulnerabilities [11].

The authentication methods of the Grid registering stage accept different security approaches. The general concept of resource centralization and authorization management does not fit into the concept of Service Oriented Architectures [11].

4. Comparison of CSRF Testing Tools

This section describes the criteria we use for comparison of the CSRF testing.

4.1. Creation of CSRF HTML File

For testing, we need to create an HTML file which contains the CSRF attack code. This HTML file can either be generated manually by the tester or with the help of a tool.

4.2. Proxy Listening

Proxy listening is the process of recording each and every request sent by the browser to the web applications. The tester cannot do this manually and hence requires the help of a tool. There are tools which can block the request and ask for confirmation from the tool to forward the request to the web application.

4.3. Auto Submission of Forms

The auto submission of the forms in the POST requests can be done with the help of JavaScript codes. The tester can create this code with the help of a tool or can make it manually.

4.4. Creation of Form in the Browser

Some tools create the forms in the POST requests in the browser during testing. This criteria is to check whether a particular tool creates the form in the browser. Based on the testing carried out with the selected tools, we found that the CSRF tester tool developed by the OWASP is the most effective based on the criteria described in Table 1.

Table 1: Comparison of Tools for CSRF Testing

Criteria / Tools	CSRF HTML file	Listen Proxy	Auto Submit Form	Form in the Browser
OWASP CSRF Tester	Yes	Yes	Yes	No
Burp Suite	Yes	Yes	No	No
OWASP ZAP	No	Yes	No	Yes
Pinata	Yes	No	Yes	No

Pinata and CSRF tester are tools, which can be used for the auto submission of the forms, but Pinata does not have the proxy listening functionality. Proxy listening is nearly a deciding factor in the testing process. The other two tools, Burp Suite and OWASP ZAP, are good tools, which can test for all vulnerabilities. However, they are not efficient as a CSRF tester tool because they cannot submit the forms automatically, even if they can perform the attack by manually submitting the form. Since most of the attacks are targeted on the POST requests, the submission of the form should be done automatically in order to test it. Therefore, this is a big drawback in these two tools. Based on the comparison, it is evident that the OWASP CSRF tester is more effective than other tools because, contrary to other tools, it satisfies the three main criteria (refer to Table 1).

5. Criteria for Comparison of Solutions

A comparison of CSRF solutions is made based on nine criteria, which are selected based on the literature. These criteria also depend on the mechanism of different solutions. These criteria are discussed in detail below.

5.1. Prevention against Get Requests

This criterion checks whether the solution gives protection for requests, which use the GET method. GET is one of the four types of methods used in requests. It involves reading data from the server by sending a request. The comparison table (Table 2), shows which solutions provide protection against GET requests and which ones do not.

5.2. Prevention against Post Requests

POST is another of the four main methods used to retrieve data from the server. In a POST requests a form is submitted to the server requesting the data. Once the form is filled in and submitted, the server returns the result. The comparison table shows which solutions provide protection and which ones do not against POST requests.

5.3. Prevention against Single Step Transactions

Most sensitive data handling requests these days uses multi requests. However, some web applications used single step or a single request for this kind of transactions in the past. This criterion tests whether a solution protects a Web Application which uses a single step for sensitive operations from the CSRF attacks, although, the use of single step requests for sensitive transactions is not recommended.

5.4. Usage of Random Value in the Code

Some of the solutions to prevent CSRF attacks deploy a pseudo random number for the implementation of the solution. Checking each solution against this criterion tests whether a particular solution uses a random number in the code itself or not.

5.5. Usage of Random Value in the URL

Passing the randomly generated token is one of the most effective solutions to protect a Web Application against CSRF attacks. Each solution in the comparison table is therefore checked so that solutions, which pass the pseudo random value through the URL, can be identified.

5.6. Usage of Random Value in the Cookie

The randomly generated token can also be passed with the cookie information. By checking each solution against this criterion, we can identify which solution uses this method. This is a potential solution even if all the information in the cookie is accessible to the attackers.

5.7. Random Value Encryption

The random value created using a pseudo random number generator can be encrypted and used as a preventive mechanism against the CSRF attacks. The solutions are checked against these criteria to understand whether they use the above mechanism to activate the protection. The encryption is normally done to deny the attackers access to the random number.

5.8. Browser Dependency

There are existing ways to prevent CSRF attacks in Web applications which are dependent on the browser. However, an effective solution should not depend upon Brower's capability to enhance its protection level.

5.9. Domain Dependency

The effective solutions to prevent a web application from the CSRF attacks should not be dependent on the domain. According to the literature, there are existing solutions which are dependent on the domain. This criteria is used to identify which solutions are dependent on a particular domain.

6. Comparison of Solutions to Prevent CSRF Attacks

A comparison of the CSRF solutions has been made, based on the criteria as listed in Table 2. This comparison helps us to understand the efficiency of each solution to prevent the CSRF attacks.
Passing a pseudo random token through the hidden field is considered the most efficient existing solution to prevent the CSRF attacks in web applications. At the server end, this token is validated against the token in the session. If both the tokens match, then it is understood that the request originated from the same origin. By passing the token through the hidden field, access of the attackers to the token is denied [17].
In POST requests, along with the normal form, the token is included as a hidden field [9]. Single step transactions are also protected, because the form submission automatically converts single step transactions into multistep transactions.
Since the pseudo random value is passed through the hidden field, the token is in the code. The token is not present in the URL or in the cookie. The token remains unencrypted in this solution. Another advantage is that it is not dependent on the browser or on the domain because the solution is implemented in the Web Application itself [11].
The second most effective solution is passing the unpredictable random token through the URL. This solution is mainly implemented in GET requests but it can also give protection to POST requests. Even if the token is accessible to the attacker, it is not useful because the validity of the token expires when the session expires [11]. A new token is generated at the start of each session. Single step transactions are also protected because the token is validated on the server side with each request. The token is not present in the code and in the cookie but it is available in the URL. Similar to the solution discussed before, this one is also independent of the browser and the domain [7, 18]. Of the two methods discussed above, the former one, passing a pseudo random token through the hidden field, is more powerful and more widely used.
Besides the above, checking the referrer header of a request is also a possible solution to prevent CSRF attacks. It helps to identify the origin of the request [19]. This is independent of the type of request. However, not all browsers have the option of sending the referrer header with each request. Even if the browser supports it, the option to turn off the referrer header is available. This means that this solution is dependent on the browser as well as the domain [20].
Submitting the cookie twice is simply placing the generated token in the cookie information itself and sending it. Since the cookie can be accessed by anyone, this solution does not provide proper protection. On the server side the value in the cookie is validated against the value in the session. The only advantage is that the solution is not dependent on the browser and the domain [7].
An effective method is encrypting the token which is generated randomly and passing it through the session. Rather than checking the token at the server side, the token is decrypted and compared with the generated token [21]. This is efficient but harder for the

developers as it requires coding to encrypt, decrypt and compare the token. Due to this difficulty, the process is seldom used. The advantage of only passing the encrypted token is Browser and Domain independence [7].
Similarly, the origin headers are also dependent on the browsers. Attaching the origin header with every request is a feature of the 'Firefox' browser, which was developed by a research group comprising Adam Barth, Collin Jackson and John C. Mitchell [2]. None of the other browsers have this functionality. The process is therefore dependent on a particular browser. Since there is no generation of a token, there is no presence of a random value in the code or URL or the cookie [20].

Table 2: Comparison of the Solutions to Prevent CSRF Threat

	Criteria → / Solution ↓	Protect GET requests	Protect POST requests	Protect single step operations	Token in the code	Token in the URL	Token in the cookie	Token Encryption	Depends Browser	Depends Domain
1	Pseudorandom value (synchronizer token pattern) as a hidden field in the form	Yes	Yes	Yes	Yes	X	X	X	X	X
2	Pseudorandom value (synchronizer token pattern) disclosed in the URL	Yes	Yes	Yes	X	Yes	X	X	X	X
3	Using only post requests in the applications	Yes	X	X	X	X	X	X	X	X
4	Multi step transactions	Yes	X	Yes	X	X	X	X	X	X
5	Double submitting cookies	X	Yes	Yes	X	X	Yes	X	X	X
6	Encrypted token pattern	Yes	Yes	Yes	X	X	X	Yes	X	X
7	Checking the referrer header	Yes	Yes	Yes	X	X	X	X	Yes	Yes
8	Checking the origin header	Yes	Yes	Yes	X	X	X	X	Yes	X
9	Challenge response (disadvantage: not user friendly)	Yes	Yes	Yes	X	X	X	X	X	X

Getting a response from the client by a challenge is one of the best ways to identify the origin of a request. In fact, it is a double authentication procedure for a request to be processed. It is the only solution which gives proper protection against CSRF attacks without the use of pseudo random numbers [22]. However, the disadvantage is that asking for re-authentication in each and every request is uncomfortable for the user. User-friendliness is crucial for any good web application. This Browser independent technique is only implemented in requests which are highly sensitive, to give extra security in addition to the token validation [7].

7. Solution and Recommendations

The proposed technique will present an effective solution, which can offer double-layered protection against CSRF. Figure 1 depicts the flowchart which demonstrates the general progression of the algorithm.
The OWASP CSRF tester is used in this case as well to generating the html file to execute the attack. The test has been carried out using our own dedicated private cloud. The file used for attacking purposes will register a new user to the application from an external source. An HTML file for the registration request using the CSRF tester tool has been generated. After that, the parameters of the form have been edited to convert it an attack file. The screenshot of this file is shown below (Figure 2). To ensure that the token is passed through both hidden field and the URL, please review the highlighted section. The same token is displayed in the URL and in the hidden field in the form.
Once the form was created using the tool, the parameters of the registration fields have been edited. If the attack executes successfully, a new user with username '34343' will be created in the Web Application. To execute the attack, the file is accessed in the same browser as where a user is logged into the application. The result for the test is shown below.
Once the attack is executed, the application rejected the request flagging a token error. This means that the token present in the attack file did not match the token in the session when the request was sent. As a result the execution of the attack was not successful and the user was not registered to the application. This shows that the solution provides protection to the application against CSRF attacks. With this suggested solution, even if the attacker gets access to the token in the URL, the attack will not be executed successfully because of the double validation (validated with the token in a hidden field and a token in the URL).
The result will be the same if the attacker obtains the token in the hidden field. This suggested solution provides more protection for web applications than the existing solutions.

8. Conclusion

CSRF attacks are riskier than they first appear to be because most web developers are unaware of CSRF and fail to provide proper protection. These attacks can be easily executed by the attackers by using straightforward techniques. A reliable mechanism for protecting a Web Application against these attacks is absolutely essential. There are many existing solutions which can be implemented in a Web Application to prevent CSRF attacks. Of these, passing an unpredictable token through a hidden field and validating it at the server side is the most effective solution. Passing the token through the URL is the next best solution. Ongoing research is currently carried out to develop even more stringent methods to foil the CSRF attacks. The suggested solution is a combination of the most effective existing technique and the second best option. By implementing this, a double validation takes place at the server side of the web application to ensure the prevention of CSRF attacks.

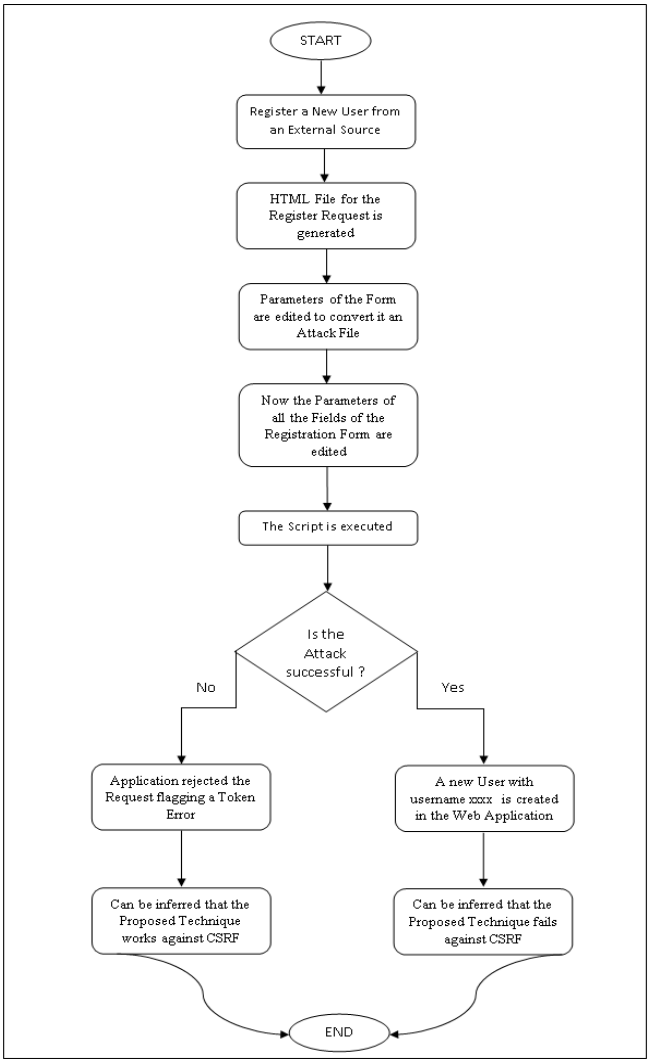

Fig. 1: Flowchart of the proposed process

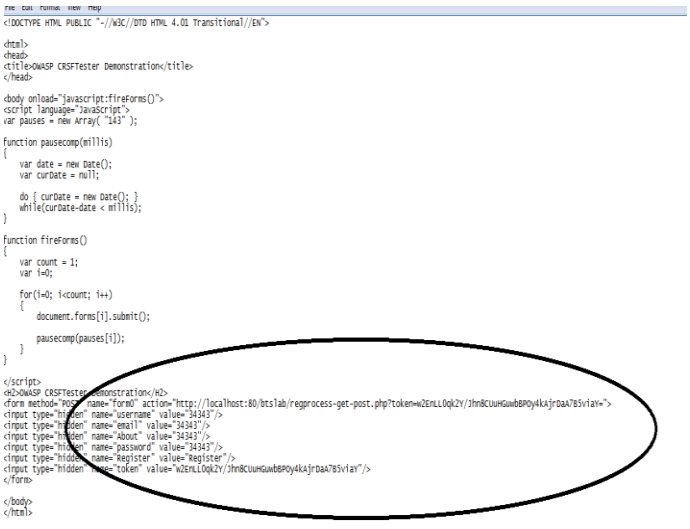

Fig. 2: Screenshot of the attack file created using CSRF Tester [7].

Currently, hackers do not seem to have devised a method to obtain the unpredictable token in the URL yet. However, if the attackers find a way to get this token in the future the suggested solution can be still used to prevent the attacks. Even if the attackers can get the token in the URL, they will not be aware of the token in the hidden field. Since the token is validated twice at server side, the attack request will be rejected. Contrary to other existing solutions, this solution provides protection against both GET requests and POST requests. The fact that the token passes through the code and the URL makes it preferable as well. Since the implementation of the solution is done in the Web Application, it is also independent of the browser. Considering all these properties, it would be expected that the suggested solution would be more effective to protect Web Applications from CSRF attacks than the existing solutions.

References

[1] Webappsec (2017). Webappsec resources. https://danielmiessler.com/projects/webappsec_testing_resources

[2] Caviglione, L., Merlo, A., & Migliardi, M. (2012). Green-aware security: Towards a new research field. Journal of Information Assurance and Security, 7(6), 338-346.

[3] Vala, R., & Jasek, R. (2011). Security testing of web applications. Proceedings of the Annals of DAAAM and Proceedings, pp. 1533-1535.

[4] Grossman, J. (2007). Whitehat website security statistics report. http://hhs.janlo.nl/articles/Whitehatstat.pdf.

[5] Ahmed, N., & Abraham, A. (2013). Modeling security risk factors in a cloud computing environment. Journal of Information Assurance and Security, 8, 279-289.

[6] Kafer, K. (2008). Cross site request forgery. Technical report, Hasso-Plattner-Institut.

[7] OWASP. (2017). CSRF prevention cheat sheet. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.

[8] Akanbi, O., Abunadi, A., & Zainal, A. (2014). Phishing website classification: A machine learning approach. Journal of Information Assurance and Security, 9(5), 222-234.

[9] Khurana, P., & Bindal, P. (2014). Vulnerabilities and defensive mechanism of CSRF. International Journal of Computer Trends and Technology, 13(4), 2231-2803.

[10] Jovanovic, N., Kirda, E., & Kruegel, C. (2006). Preventing cross site request forgery attacks. Proceedings of the IEEE Securecomm and Workshops, 2006, pp. 1-10.

[11] Zeller, W., & Felten, E. W. (2008). Cross-site request forgeries: Exploitation and prevention. The New York Times, pp. 1-13.

[12] Burns, J. (2005). Cross site request forgery-An introduction to a common web application weakness. Whitepaper.

[13] DeepDotWeb. (2015). Warning: New malicios JS using CRFST exploit via PM's on Agora. https: //www.deepdotweb.com/2015/06/11 /warning-new-malicious-js-using-csrf-exploit-via-pms-on-agora/.

[14] stack exchange (2017). astonishing recent belkin router auth bypass vulnerability: CSRF used to exploit? https://security.stackexchange.com/questions/100921/astonishing-recent-belkin-router-auth-bypass-vulnerability-csrf-used-to-exploit.

[15] Mansfield-Devine, S. (2008). Anti-social networking: Exploiting the trusting environment of Web 2.0. Network Security, 2008(11), 4-7.

[16] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext transfer protocol--HTTP/1.1 (No. RFC 2616).

[17] Menzel, M., Wolter, C., & Meinel, C. (2007). Access control for cross-organisational web service composition. Journal of Information Assurance and Security, 2(3), 155-160.

[18] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., & Stewart, L. (1999). HTTP authentication: Basic and digest access authentication (No. RFC 2617).

[19] Bojinov, H., Bursztein, E., & Boneh, D. (2010). The emergence of cross channel scripting. Communications of the ACM, 53(8), 105-113.

[20] Shaikh, R. (2013). Defending cross site reference forgery (CSRF) attacks on contemporary web applications using a Bayesian predictive model. https://sci-hub.tw/https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2226954.

[21] Barth, A., Jackson, C., & Mitchell, J. C. (2008). Robust defenses for cross-site request forgery. Proceedings of the ACM 15th ACM Conference on Computer and Communications Security, pp. 75-88.

[22] Jurcenoks, J. (2013). Owasp to wasc to cwe mapping correlating different industry taxonomy. Critical Watch, 7-11.