CSRF and XSS

with a practical example using Burp Suite

Seminararbeit

Ausgewählte Kapitel der IT-Security

Author:
Nathalie Muehlberger

student identification number
1710475083

Handed in:
08.01.2020

Abkürzungsverzeichnis

CSRF	Cross-Site Request Forgery
XSS	Cross-Site Scripting
RFI	Remote File Inclusion
LFI	Local File Inclusion

Schlüsselbegriffe

CSRF
XSS
RFI
LFI

Contents

Chapter 1

Introduction

As the internet is used nowadays for fields like e-commerce, finance and healthcare, web applications need to become more and more secure to prevent attackers from exploiting any vulnerabilities so the user's credentials can be kept safe. As some vulnerabilities are not as widely known as they should be, although they are rather common, this paper deals with the two vulnerabilities Cross-Site Request Forgery(CSRF) and Cross-Site Scripting(XSS). Both work through injecting HTML or similar code with malicious code, but while CSRF exploits the vulnerabilities based on the website trusting in the user's requests, XSS exploits vulnerabilities based on the user trusting in the website's integrity. This paper will start with CSRF, its vulnerabilities, how an CSRF attack is done and some defense mechanisms are also being explained in the following chapter. Chapter 3 deals with XSS and focuses especially on the different types of XSS attacks, namely Reflected XSS attacks, Stored XSS attacks and DOM-based XSS attacks. Furthermore does this chapter contain some detection and prevention techniques for XSS attacks to offer an overview of how it could be possible to defend oneself against XSS attacks. This part of the chapter also shows against which of the attack types which mechanism offers detection and/or even prevention. The fourth chapter gives a step-per-step instruction on how a CSRF vulnerability can be detected using Burp Suite. In the last chapter one can find the conclusion.

Chapter 2

Cross-Site Request Forgery

Cross-Site Request Forgery is an attack that tricks the end user to execute unwanted actions on a web application, like submitting a malicious request, in which they are authenticated at. In a browser request for most sites the user's session cookie, IP address and Windows domain credentials, needs to be included. Because of that, it is impossible for the site to distinguish between a forged request of an attacker sent by the victim and a legitimate request of the victim. CSRF attacks target specifically state-changing requests, because the response to the forged request cannot be seen by the attacker. State-changing requests can be transferring funds, changing the e-mail address or password, purchasing items or if the victim is an administrative account, the entire web application can be compromised. "Stored CSRF flaws" are CSRF attacks that are stored on a vulnerable site itself. This can be done using a more complex cross-site scripting attack (XSS) or by storing an IMG or IFRAME tag in a field that accepts HTML. Such an attack amplifies the severity of a CSRF attack, because it is more likely that the victim will be authenticated to the site already. [1]

2.1 Synonyms

- XSRF.
- Sea Surf.
- Session Riding.
- Cross-Site Reference Forgery.
- Hostile Linking.
- One-Click attack.

[1]

2.2 File Inclusion

Allowing an attacker to include a file, because of the use of user-supplied input without proper validation is called File Inclusion vulnerability. This vulnerability can lead to code execution on the web server or the client-side like JavaScript, which can then lead to attacks like XXS, Denial of Service or Sensitive Information Disclosure.[2]

2.2.1 Remote File Inclusion

With Remote File Inclusion remote files are included through exploiting vulnerable inclusion procedures implemented in the application. This can happen, when a page receives a path to a file that has to be included as input and this input is not properly checked allowing external URL injection. [3]

2.2.2 Local File Inclusion

If files that are already locally present on the server are included through the exploiting of vulnerable inclusion procedures implemented in the application, this process is called Local File Inclusion. This can happen, when a page receives a path to a file that has to be included as input and this input is not properly checked allowing directory traversal characters like dot-dot-slash to be injected. [2]

2.2.3 Difference between RFI and LFI

Attackers can with LFI directly upload the malware to the target network as opposed to RFI where the victim downloads the malware himself, by using an external referencing function from a remote location that has been tampered with by the attacker. [4]

2.2.4 Solution against File Inclusion Attacks

Avoiding to pass on user-submitted input to a filesystem or framework API is the most effective solution against file inclusion attacks. If this cannot be done, another good solution is to maintain a white list of files and an identifier to only grant access to the selected files. This way malicious requests containing invalid identifiers will be rejected and there is no attack surface left for attackers to manipulate the path. [2]

2.3 Vulnerability

CSRF has several different categories of vulnerabilities such as HTTP Post handling error, Off-By-One-Error or unrestricted upload attachments.

In the figure below the CSRF vulnerabilities, that have been identified, are shown, with the according percentages these vulnerabilities have been encountered.[5]

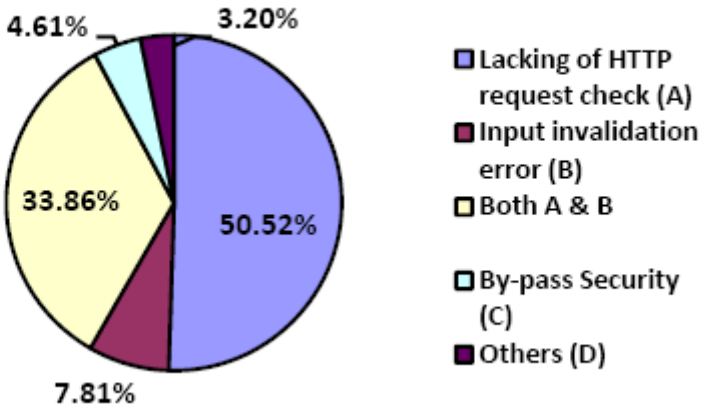

Figure 2.1: Vulnerabilities

Some vulnerabilities attackers could use would be unrestricted authentication area weakness, in which the attacker might be able to access the logged-in users or insufficient session expiration weakness, which might allow the attacker to reuse old session credentials or IDs for authorization. Another weakness an attacker could exploit would be the critical state data vulnerability, which would allow the attacker to modify the state information of the target website without detection. The possibility on some Webservers for users to create and customize their own content, gives attackers the opportunity to conduct CSRF attacks, with which they then can change data that is saved on the server. Incorrect security restrictions on scripts can lead to attackers abusing this restriction to upload malicious scripts like CSRF worms to the target Web server. If restrictions for upload attachments are lacking, malicious HTML or JavaScript attachments could be uploaded, which could lead to an attacker being able to access user cookies or perform other restricted actions. Another weakness occurs if a Web application uses highly predictable URL structures. An attacker could then have enough required information to craft a malicious link.

If a Website has an insufficient session expiration vulnerability, an attacker could use old cookies and could steal the user's session tokens, with which he could then predict the CSRF random tokens via the insufficiently random number weakness. Even if tokens are used as a defense mechanism against CSRF, these tokens could be stolen by someone using XSS. The unexpired cookies that could be then captured provide the most convenient, hidden, effective and durable means of exploiting session fixation vulnerailities. The attacker would then be able to set a user's session cookie value to a value he controls or he could use passive network eavesdropping to read cookie values

or he could guess session IDs by brute force if the session ID is low in entropy. He could even convert HTTP Post Requests to GET Request although the target website utilizes HTTP Post method for their sensitive traffic.[5]

2.4 Attack

A CSRF attack consists of the following participants: a targeted user, a trusted website, a malicious website and an attacker.[6]

This attack occurs when a malicious email, website, blog, instant message or program causes the web browser of the targeted user to perform an unwanted action on a trusted website on which the user is already authenticated on. A CSRF attack only works because browser requests contain credentials belonging to the site, like the user's session cookie and the IP address. Because of that the site cannot distinguish between a legitimate and a forged request.[7]

There are several possibilities to expose the targeted user to the malicious code. The attacker can either trick the user to visit a malicious website or he can distribute the payload of the attack with social networking applications or by email. Today, most websites use cookies to identify authenticated users. These cookies will be given to a user, as soon as he successfully authenticated himself once at the Webserver. He will then get an identity login cookie to remember the logged-in-status. Because of that, the user can visit this website as often as he wants to without having to authenticate himself again for as long as he does not close the browser or logs himself out. The Browser will just automatically put the identity login cookie in the request. An attacker can use this window to make the user's browser perform actions without the consent of the user. The National Vulnerability Database (NVD) documents more than 200 cases of exploited CSRF vulnerabilities against real world Web applications. Some of those attacks were used to modify system configurations, create super administrator accounts or to perform unauthorized actions like post/delete data on a forum or change the user's profile. These attacks are not easy to detect a lot of the remediation are only possible after the incident happened.[5]

> Jeremiah Grossman To Web servers, one request looks more or less just like another. The unintended request is legitimate (not hacked up), the user is the right user (authenticated), and the request is being made directly to the real Web site (no man-in-the-middle). The only problem is the victim did not intend to make the request, but the Web server does not know that. And the scary part is the vast majority of Web sites are vulnerable with users having very little ability to defend themselves against CSRF

CSRF attacks are divided into two categories: reflected and stored, depending on the mechanism the CSRF code is delivered. If the CSRF code is triggered within the domain of the targeted website, the attack is called stored CSRF. Reflected CSRF is meant if the attack is triggered from a different domain.

After all the, on the NVD published, CSRF attacks have been reviewed, it was found out, that more than half of the attacks have been reflected CSRF attacks.

Steps for a successful CSRF attack: The first step is to review the key functionality within the target Web application. This can be done using Web spidering or manual browsing. Some free tools for Web spidering are Paros, Burp Spider and WebSarab. The next step will be to find a function of this application that can be used to perform a sensitive function on behalf of the victim and which employs request parameters that can in advance be fully determined. For this step Wireshark can be used to capture and analyse the packets that are sent between the client and the server. The third step needs to be to create a malicious link that will execute actions like transferring money or changing a password. The last step is for the attacker to convince the target who is logged into the target website to click on this malicious link.

For reflected CSRF attacks, the malicious link has to be included on the attackers own website, where the target either needs to click on the malicious link or where the link is automatically executed by using a XMLHTTPRequest object.

When using a stored CSRF attack instead, the attacker does not have to convince the target to click on the link. With this attack the attacker has to embed the malicious link into the target website or execute a stored XSS attack on a website, so that the XMLHTTPRequest object will automatically execute the attack as soon as the user visits this website.[5]

2.4.1 Attackers Goals

There are three major goals that have been identified based on the incidents that have been published on NVD, which can vary depending on the motives of the attacker.

- Attacking and challenging: Motives for this goal could be to build cornerstones for another more destructive attack such as SQL Injection and DoS or just to challenge their hacking skills and show their abilities by revealing vulnerabilities.
- Pure entertainment: Motives for this goal could be to change the victims preferences, to become famous or if the attacker is lonely to make new friends.
- Money: The motivation for this goal is financial profit. Most of the attackers with this goal are experts and some of them even live of the money they can make by posting advertisements for products or by directly transferring money on their own account.[5]

2.4.2 Attack Tree Model of CSRF Attacks

An attack tree is a tree structure. The root node contains the attacker's goal and the leaf nodes contain the different ways of achieving this goal. This structure provides a formal, methodical way of the security of a system based on the planned attacks the attacker wants to use.[5]

2.5 Defense

To prevent a CSRF attack, the website would need to be able to distinguish between a forged and a legitimate request. This can be done by sending additionally a token or identifier, which is not accessible for the attacker and because of that cannot be sent with a forged request.[7]

Listed below are some of these defensive techniques against CSRF and another additional one.

Token Based Mitigation
Token Based Mitigation is one of the most recommended and popular methods to defend against CSRF. It can either have a state (Synchronizer Token Pattern) or it can be stateless (encrypted/hash based token pattern). Both forms need to adhere general security principles and strong encryption/HMAC functions. Furthermore token lifetime policies and a strict key rotation should be maintained. The following part of the paper deals with three different types of Token Based Mitigation patterns.

- Synchronizer Token Pattern:
With this method a secure random token is required for every state chanigng operation. This token needs to be a large random value, unique per user session and be generated using a Cryptographically Secure Pseudo-Random Number Generator. When using this token it is important that it is not leaked in the URL or Server Logs. If this token is not leaked, it provides security, because the server rejects any action that cannot be validated with the token. The token can either added in the header, which is considered as more secure, or in the hidden field of a parameter. This method follows the follwing steps to provide security: a random token is generated for each session of a user. Then this token is inserted into all of the HTML forms and calls associated with sensitive server-side operations. The server has then the responsibility to verify the existence and correctness of the token. With this methods CSRF attacks can be mitigated as an exploitation can only be done, if the attacker knows the randomly generated token for the targeted session of his or her victim. Even more secure are per-request tokens, but as one token is used for every request, the usability could be hindered, as the "Back"Button for example could lead to a false CSRF attack positive event, as the new token will not work for the page before anymore.

- Encryption based Token Pattern:
This method is often used if the state does not need to be maintained at the server side. With this method the token is not compared, instead the server validates the user by encrypting and decrypting the token. The token consists of the session ID of the user and a timestamp to prevent replay attacks, which are encrypted using an unique key. AES256 with GCM mode/GCM-SIV is recommended for encryption. After generating the encrypted token, it is returned to the client, then embedded in a hidden field and then sent with the request. The server then decrypts the token using the same key used to create the token and if this can be done, the session ID is then furthermore compared with the currently logged-in user, after timestamp and session ID have been validated.

- HMAC based Token Pattern:
This mitigation is similar to Encryption based Token Pattern, but has the following minor differences:
Instead of an encryption function a strong HMAC function like SHA256 or stronger is used to generate the token. The token consists of the HMAC encyrpted session id and timestamp and the same timestamp is also appended to the encrypted part of the token. When the server receives the token, it is re-genereated with the same key. The parameters of the key, session ID and timestamp, can be found in the request (session ID) and the received token(timestamp). If the same HMAC is generated as the one received, the timestamp is checked to see if it is less than the defined token expiry time and therefore still valid.[7]

The Referer Privacy Guard
This guard produces a constant flow of random HTTP requests to prevent an attacker from figuring out the user's browsing trends.[8]

Chapter 3

Cross Site Scripting

Web applications have several vulnerabilities, Cross Site Scripting(XSS) being one of major ones. [9] While with CSRF the vulnerabilities are based on the website trusting in the user's requests, XSS vulnerabilities are based on the user trusting in the website's integrity. [6] XSS attacks target web applications, by injecting malicious script code either trough the client-side or the browser-side.

This chapter deals with the different kinds of XSS attacks and lists some defense mechanisms against XSS attacks.[9]

3.1 Attack

During a XSS attack a malicious code is executed in the victim's browser to steal credentials like credit card numbers or passwords. An attack follows the following process: A malicious script code is injected into a vulnerable web page, this infected web page is then visited by the victim, the malicious code is sent to the victim as part of the HTML body of the requested page and personal data like passwords or credit card details can be stolen as soon as the malicious script code is executed inside the victims browser.

XSS attacks can be classified into Reflected, Shared and DOM-based attacks. Reflected and Shared XSS atacks make use of server-side vulnerabilities, while Dom-based XSS attacks exploit client-side vulnerabilities. These classifications are being discussed in more detail in the next part of this chapter.

3.2 Reflected

Reflected XSS attacks are used to steal the session cookie of a user and is a non-persistent attack. The steps for a Reflected XSS attack are shown in Fig.3.1 and are as follows:
- a malicious script code is sent as part of a link via for example email to the user
- this malicious code is sent to server without detection from the web application, as soon as the victim clicks on the link
- the server then sends a HTTP response containing the malicious script code
- the attacker receives the user's cookies, as soon as the script contained in the HTTP response is executed
- the cookies can then be stored for future use by the attacker

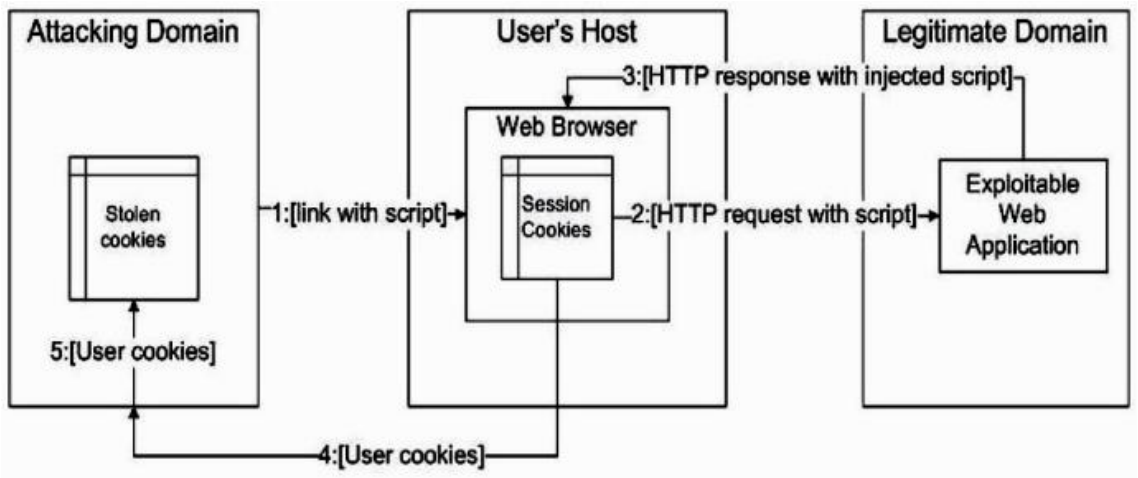

Figure 3.1: Reflected XSS Attack process
Source: [9]

3.3 Stored

Stored XSS attacks often target social networks or similar applications. The steps for a Stored XSS attack are shown in Fig.3.2 and are the following:
- a malicious script code is inserted into a vulnerable web application
- the web page content including the malicious code is accessed, as soon as the user sends a HTTP request
- the HTTP response to the user then contains the malicious script.
- the session cookies are sent to the attacker, following the script being executed by the user's web browser
- the stolen cookies are then stored on the attacker's domain

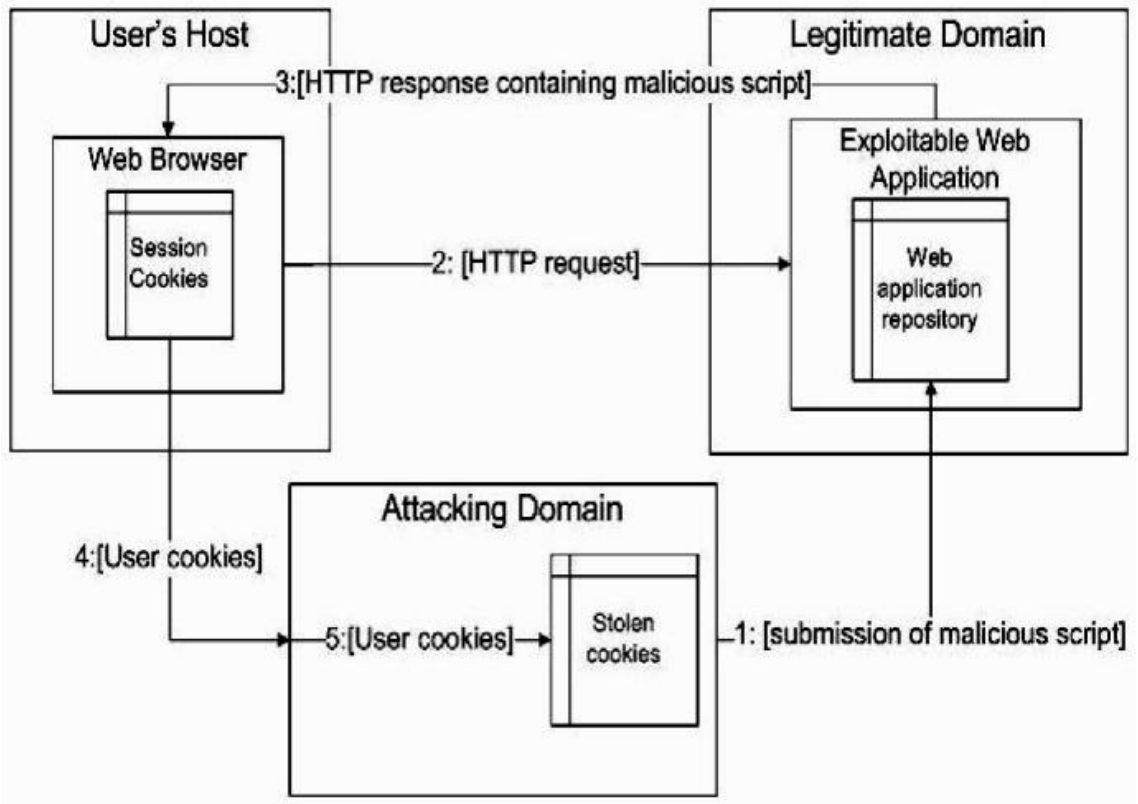

Figure 3.2: Stored XSS Attack process
Source: [9]

3.4 DOM-based

DOM-based XSS attacks are considered a client-side vulnerability, because the attack does not occur in the HTML code, it occurs in the Document Objet Model(DOM). The steps for these attacks can be seen in Fig.3.3 and are also described below:
- a link containing malicious script code is sent to the user via email or similar webpages
- this link is used by the victim, but neither HTTP request nor response contain malicious script
- the user' s cookies are sent to the attackers domain by executing the malicious code at the client
- the attacker is the able to store these stolen cookies for further use later on

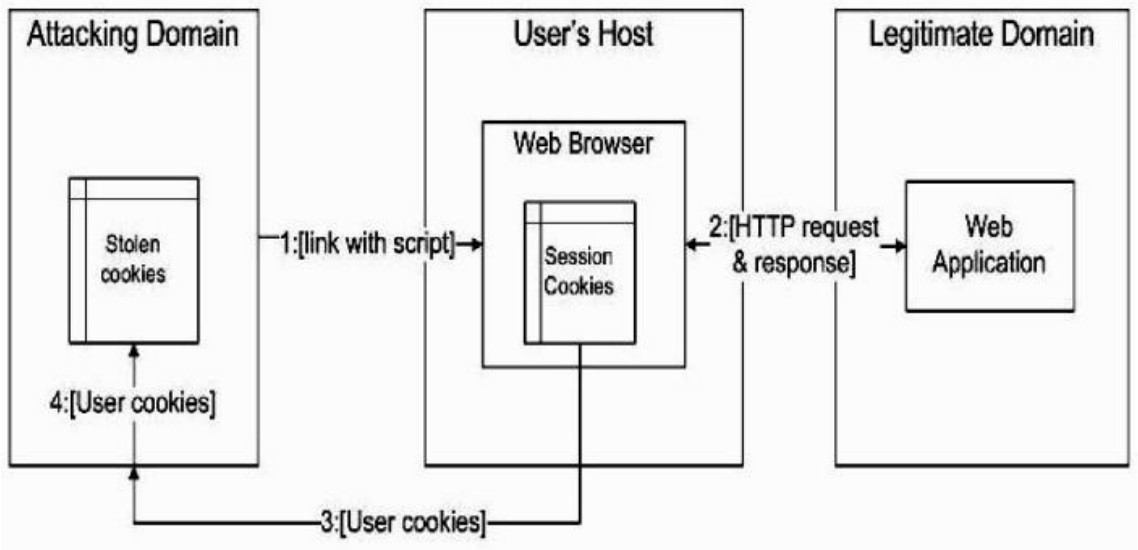

Figure 3.3: DOM-based XSS Attack process
Source: [9]

3.5 Defense

There have been devoloped several detection and prevention techniques for XSS. Some of those techniques can be seen in Fig.3.4, which also shows which of these techniques can detect and prevent, which type of XSS attacks. For anyone interested in the strengths and weaknesses of each of these techniques have a look at the paper [9].

Authors	Exploitation location	Year	Stored XSS	Reflected XSS	DOM-based XSS	Detection	Prevention	Server-side	Client-side
Shahriar and Zulkernine	Web server	2011	✔	✔	✗	✔	✗	✔	✗
Shahriar and Zulkernine	Server-Side	2011	✔	✔	✗	✔	✗	✔	✗
Barhoom and Kohail	Server-Side	2011	✔	✗	✗	✔	✔	✔	✗
Shar and Tan	server-side	2012	✔	✔	✗	✔	✔	✔	✗
Gundy and Chen	Web browser and web server	2012	✔	✔	✗	✔	✔	✔	✗
Huajie Xu et al.	Client side	2013	✗	✗	✔	✔	✔	✗	✔
Parameshwaran et al.	Client side	2015	✗	✗	✔	✔	✗	✗	✔
Guptaa and Gupta	Server side	2015	✔	✔	✗	✔	✗	✔	✗
Gupta and Gupta	Cloud	2016	✗	✔	✗	✔	✗	✔	✗
Gupta and Gupta	Cloud	2016	✗	✔	✔	✔	✔	✔	✔
Khan et al.	Client side	2017	✗	✗	✔	✔	✔	✗	✔

Figure 3.4: Detection and Prevention Techniques for XSS attacks
Source: [9]

Chapter 4

Practical CSRF Example using Burp Suite

First one needs to download Burp Suite from the following page: [10]

The next step is to download OWASP's Broken Web Application Project to be able to test for CSRF Vulnerabilities. As this Web Application has a lot of vulnerabilities, it should only be started using VMNets with „host only“ or „NAT“ networks for security reasons. The download is available at: [11]

To run Burp Suite, one has to start it, then select „Temporary project“ (as this is the only option when only having the community version) and choose „Load fromconfiguration file“ for this example and select the file from [11].[12]

One then has to configure the browser to work with Burp Suite. As I am using Firefox, the next part of the setup is according to Firefox, for everyone using another Browser, the setup can be found on the following page: [13]

When using Firfox, one has follow the following steps:

- Menu - Preferences - Options - General - Network Proxy - Settings.
- Here one has to select „Manual proxy configuration“
- Next the „HTTP Proxy“ field should be filled out with the Burp Proxy Listener address chosen when starting Burp Suite.(default(127.0.0.1))
- The „Port“ field has to be filled out with the Burp Proxy Listener port also chosen before.(default(8080))
- Check the „Use this proxy server for all protocols“ box
- Delete if necessary any information in the „No proxy for“ field and click „OK“[14]

To be able to open https-Websites while running Burp Suite, one has to install Burp's CA Certificate. Again, here is only shown how this is done with Firefox, the installation for any other browser can be found here: [15]

With Firefox, Burp Suite needs to be running. One has to enter „http://burp“ in his or her browser and click on the „CA Certificate“ Button on the uppermost right corner

to download the certificate. No one should forget the location where the certificate is saved, as it is needed again shortly. Afterwards the steps below need to be executed to be able to acces https-Websites while running Burp Suite:

- Menu - Preferences - Options - Privacy and Security - Certificates - View Certificates - Authorities - Import
- now one has to select the Burp CA certificate file, that he or she just downloaded and click „Open“
- the box „ Trust this CA to identify web sites“ has to be checked, when the dialog box pops up and the selection is saved by clicking on „OK“
- after restarting Firefox it should be possible to open https-Websites while running Burp Suite[16]

After this initial setup, one has to ensure that in the „Intercept“ tab of the „Proxy“ tab in Burp Suite „Intercept is off“ is chosen.

To test the „GETBOO“ website of [11] for CSRF vulnerabilities one then has to follow steps below:

- authenticate into the website with the credentials user:user
- in the tab „Settings - - Modify account information“ the value of the field „Email“ should be changed
- in Burp Suite in the „Intercept“ tab of the „Proxy“ tab „Intercept is on“ needs to be chosen.
- for Burp Suite to be able to capture the data, the request need to be submitted
- in the „Proxy“ tab right click on the raw request - Engagement tools - Generate CSRF PoC
- in the „CSRF PoC generator“ the value of email is changed to „attacker@malicious.com“
- in the same window one has then to click „Copy HTML“
- this HTML then has to be copied into a text editor and saved as a HTML file in this case with the name „CSRFTry“
- select in the „Intercept“ tab of the „Proxy“ tab in Burp Suite „Intercept is off“ again
- afterwards the „CSRFTry“ HTML file should be opened using the same browser as before
- sometimes the request is submitted automatically, sometimes one needs to submit the request manually

If the data is sucessfully changed, the attack has been successful and the website is vulnerable to CSRF Attacks.[12]

Chapter 5

Conclusion

Although CSRF and XSS are rather common web application vulnerabilities, they are not as widely known, as one would think. Therefore this paper gave an overview of both of these vulnerabilities and showed a practical example of how to find a CSRF vulnerability using Burp Suite. In chapter 2 the steps of a typical CSRF attack have been discussed and several defense mechanisms were shown as well. XSS was the topic of chapter 3 and the different types of XSS attacks were especially disussed . After both of the vulnerabilities have been explained theoretically, a practical example for finding CSRF vulnerabilities with a step-per-step instruction was given in chapter 4. After reading this paper everyone should be able to differentiate between CSRF and XSS and explain what each of this attacks do. Furthermore one should be able to find CSRF vulnerabilities using the example shown in chapter 4.

Appendix A

Lists

List of Figures

Bibliography

[1] "Cross Site Request Forgery," https://www.owasp.org/index.php/Cross-Site_Request_Forgery, eingesehen am 24.11.2019. [Online]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery 2

[2] "Testing for Local File Inclusion," https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion, eingesehen am 24.11.2019. [Online]. Available: https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion 3

[3] "Testing for Remote File Inclusion," https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion, eingesehen am 24.11.2019. [Online]. Available: https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion 3

[4] "Remote File Inclusion (RFI)," https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/, eingesehen am 24.11.2019. [Online]. Available: https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/ 3

[5] X. Lin, P. Zavarsky, R. Ruhl, and D. Lindskog, "Threat Modelling for CSRF Attacks," *IEEE*, 2009. 4, 5, 6

[6] M. S. Siddiqui and D. Verma, "Cross Site Request Forgery: A common web application weakness," *IEEE*, 2011. 5, 9

[7] "Cross Site Request Forgery Prevention Cheat Sheet," https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html, eingesehen am 24.11.2019. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html 5, 7, 8

[8] T. Alexenko, M. Jenne, S. D. Roy, and W. Zeng, "Cross Site Request Forgery: Attack and defense," *IEEE*, 2011. 8

[9] S. K. Mahmoud, M. I. Roushdy, M. Alfonse, and A.-B. M. Salem, "A comparative analysis of cross-site scripting(xss) - detecting and defense techniques," *IEEE*, 2017. 9, 10, 11, 12, 13

[10] "Download burp suite community edition," eingesehen am 08.01.2020. [Online]. Available: https://portswigger.net/burp/communitydownload 14

[11] "Owasp broken web application project," eingesehen am 08.01.2020. [Online]. Available: https://sourceforge.net/projects/owaspbwa/files/ 14, 15

[12] "Using burp to test for cross-site request forgery(csrf)," eingesehen am 08.01.2020. [Online]. Available: https://support.portswigger.net/customer/portal/articles/1965674-using-burp-to-test-for-cross-site-request-forgery-csrf- 14, 15

[13] "Configuring your browser to work with burp," eingesehen am 08.01.2020. [Online]. Available: https://support.portswigger.net/customer/portal/articles/1783055-Installing_ConfiguringyourBrowser.html 14

[14] "Configuring firefox to work with burp," eingesehen am 08.01.2020. [Online]. Available: https://support.portswigger.net/customer/portal/articles/1783066-Installing_ConfiguringyourBrowser-FF.html 14

[15] "Installing burp's ca certificate in your browser," eingesehen am 08.01.2020. [Online]. Available: https://support.portswigger.net/customer/portal/articles/1783075-Installing_InstallingCACertificate.html 14

[16] "Installing burp's ca certificate in firefox," eingesehen am 08.01.2020. [Online]. Available: https://support.portswigger.net/customer/portal/articles/1783087-Installing_InstallingCACertificate-FF.html 15