1 ATTACKING CLIENTS OF WEB APPLICATIONS

Web applications are used by variety of different types of clients. Window, Linux and Mac operating systems with varied browser possibilities expose broad attack surface for an attacker. Also web application users have different levels of security measures implemented in their systems ranging from non-existent to very good. Recent browser level security measures have improved the security of web application clients, but still attacking users is easier than attacking web application or server functionality.

By attacking web application's users malicious person can perform operations such as:

- Stealing user's session which allows him to impersonate user.
- Trigger web application functionality without client's consent.
- Steal sensitive data.
- Steal usernames and passwords.
- Perform virtual defacement

What the attacker can do is limited by his imagination. With client side attacks malicious person is usually able to execute JavaScript on user's browser and therefore he has control of the whole browser.

1.1 CROSS SITE SCRIPTING

Cross Site Scripting (XSS) is an attack where malicious person is able to use web application vulnerability to execute JavaScript on victim's browser. There are three types of XSS vulnerabilities, reflected, stored and DOM based.

The fundamental error what makes XSS possible is the insufficient user input sanitation. The web application uses unfiltered user input to generate a dynamic webpage (HTML). Since the web application user can insert special characters such as '<">', he can take control of the HTML representation of the dynamic webpage and insert <script> tags to run JavaScript. The attacker will take an advantage of this by crafting malicious requests that contain attack scripts. This request might be stored in the web server or a web application user can be tricked to make the request on the behalf of the attacker.

1.2 REFLECTED XSS VULNERABILITIES

In reflected XSS the attacker's malicious payload is not stored in web application but the payload is reflected to the victim. This means that the attacker will craft a special URL containing the attack that can be sent to the victim. The URL can be f.e. posted to a message board or emailed to the victim. Attacker can use URL shortening services to make this type of attack even more difficult to detect.

For an example consider a web application that uses a GET parameter to modify HTML content of a dynamic web page:

URL:	*http://www.example.com/index.php?name=Victim*
HTML:	*<p>Welcome to this nice site Victim</p>*

To run JavaScript following payload could be inserted instead of the real name:

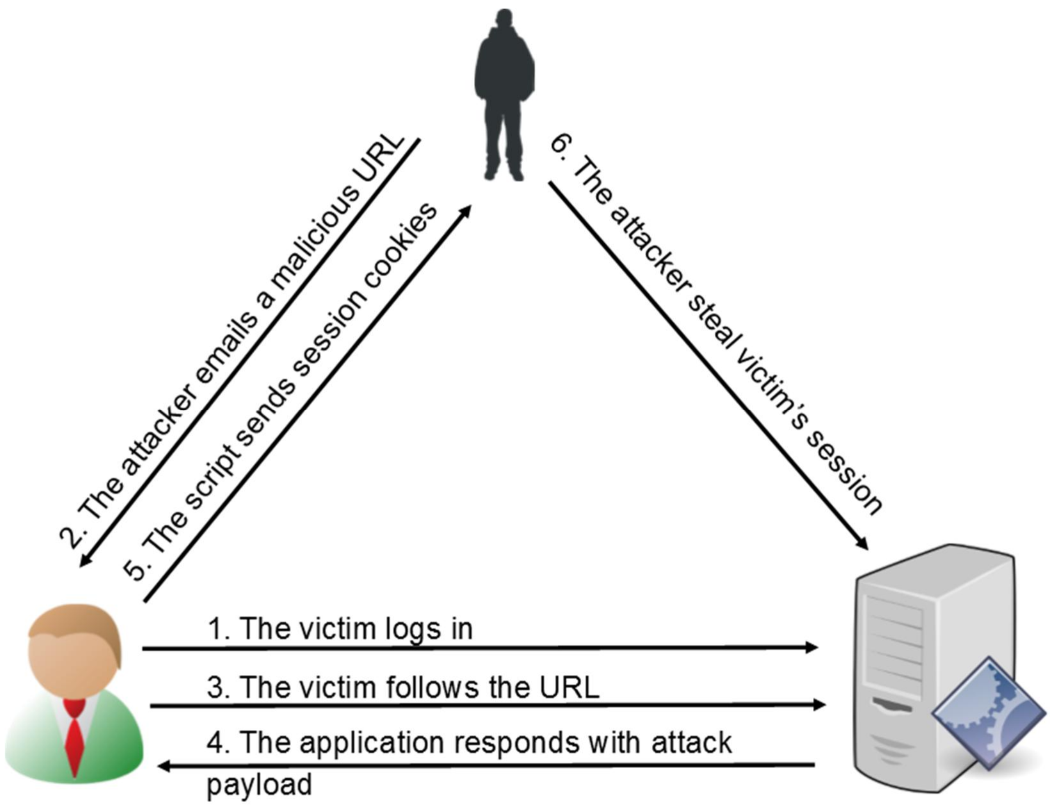
</p><script>alert("Hacked!!")</script><p>

This would have to be URL encoded before adding to a link:

http://www.example.com/index.php=name=%3C%2Fp%3E%3Cscript%3Ealert(%E2%80%9CHacked!!%E2%80%9D)%3C%2Fscript%3E%3Cp%3E

HTML representation: *<p>Welcome to this nice site </p><script>alert(“Hacked!!”)</script><p></p>*

This is a valid HTML and will create pop up screen with a “Hacked!!” text. Obviously instead of alert box the attacker could steal session cookies or other sensitive information. In figure X. it is shown how a reflected XSS attack could take place.


2. The attacker emails a malicious URL
5. The script sends session cookies
6. The attacker steal victim's session
1. The victim logs in
3. The victim follows the URL
4. The application responds with attack payload

Picture X. Reflected XSS attack scenario.

1.3 Stored XSS vulnerabilities

Functionality of a stored XSS vulnerability is similar to the reflected one. The difference is that the attack is stored in the web application. This makes the stored XSS vulnerability more dangerous since the attack requires no action from a victim to be executed. The attack will be executed if the victim visits the vulnerable page. Also usually the victim is already logged in the application when the stored XSS attack is executed. In reflected XSS user might not be logged in when the XSS is executed and thus the attack might fail because of this. In real cases stored XSS vulnerabilities can be found in message board and online commenting applications. Without prober input sanitation an user can insert HTML tags and execute scripts.

Stored XSS attack is executed in two phases. In first phase the attacker posts an attack payload to stored XSS vulnerable page. In second phase a victim visits the page and the attack payload will be executed. In figure X. is shown how this attack could take place.

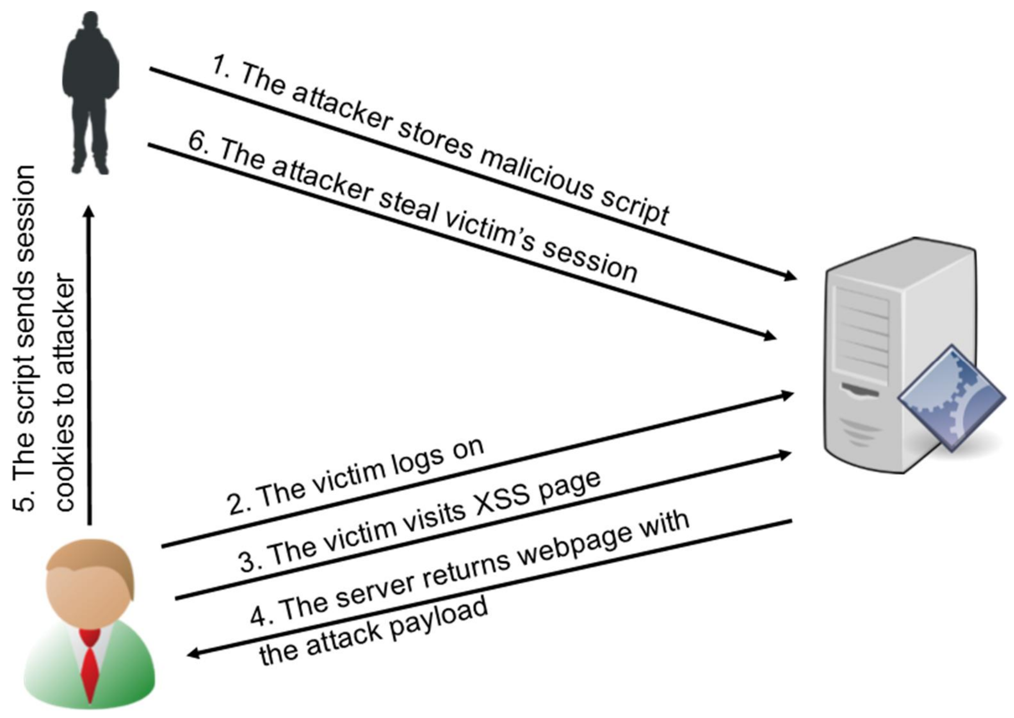

1. The attacker stores malicious script
6. The attacker steal victim's session
5. The script sends session cookies to attacker
2. The victim logs on
3. The victim visits XSS page
4. The server returns webpage with the attack payload

Figure X. Stored XSS vulnerability attack scenario.

1.4 DOM BASED XSS VULNERABILITY

In previous XSS types the server side code of a web application was adding the attack to HTML of the web page. In DOM based XSS vulnerability the web applications response doesn't contain the attack payload (script). The attack payload is contained in the URL and client side JavaScript of the web application uses parts of the URL to modify the Document Object Model (DOM). Therefore if URL is maliciously crafted the JavaScript can insert the malicious attack payload into the webpage.

DOM based XSS vulnerability requires victim's actions to be successful. Attackers use same methods as described in reflected XSS attacks to deliver the attack to the victim.

1.5 DETECTING XSS VULNERABILITIES

A web application generates dynamically content to be shown to the clients. The content generation is modified with parameters (GET/POST/Cookies) which are relayed to the web application. Therefore XSS vulnerabilities can be found by investigating parameters in detail. The basic detection procedure is following:

1. Locate all parameters used by a web application.
2. Insert a detection string in each parameter one by one.
3. Locate any appearance of the detection string in the application response.
4. If detection string appears in response, insert special characters (<>#') in the detection string.
5. If special characters are shown without filtering, the XSS vulnerability is confirmed.
6. If filtering is in place, apply filter evasion methods.

XSS vulnerabilities can be located in several places such as GET/POST-parameters, Cookies, URL or HTTP headers.

1.6 Defending against XSS attacks

CSRF

- Predictable URLs
- Defending with CSRF tokens