Network and Web Security

Tutorial 6: Client-Side Web Vulnerabilities - Part 1*

February 22, 2021

1 Cross-site scripting vulnerabilities in DVWA

In this tutorial, we'll continue using the copy of the *Damn Vulnerable Web Application (DVWA)* used for Tutorials 5. In modules 17-18 we saw that any scripts executing on a web page (e.g., those included with the `<script>` element) run in the same origin as the including page. This gives included scripts access to a wealth of valuable or sensitive data: the origin's cookies, Web Storage and IndexedDB databases can all be read and written, and arbitrary AJAX requests can be made to URLs in the same origin.

Unless a web application is carefully designed, it may be vulnerable to the various *cross-site scripting* (*XSS*) attacks discussed in modules 19-20. With an XSS an attacker manages to subvert the HTML document produced by the web application and served to users by injecting their own malicious client-side code into it that is then executed by the victim's web browser. Like the genuine scripts included on the page, this malicious code runs in the same origin as the web application, and may be able to exfiltrate sensitive data (such as the victim's session cookie) to a server under the control of the attacker, or perform privileged operations with the victim's authority (e.g., by initiating same-origin AJAX requests to the web application's API endpoints).

In this tutorial, we'll encounter two types of XSS vulnerability:

Reflected XSS. This occurs when a web application directly includes untrusted input from the URL's query string or a submitted form in the HTML document it outputs. If the untrusted input is not correctly sanitised before it is included in the HTML document, it may be possible to craft a malicious query string or form submission that causes the attacker's client-side code to be injected into the page and executed by the victim's browser.

Stored XSS. This occurs when a web application stores untrusted input from the URL's query string or a submitted form (e.g., in a server-side file or database table) and later uses this input in a HTML document that it outputs. As with reflected XSS, if this input is not correctly sanitised before it is read from storage and included in the output HTML document, it may be possible to inject a malicious client-side script into the HTML. Stored XSS attacks are often more damaging than reflected XSS attacks, since the malicious script is usually injected into the HTML document for *every* user who subsequently visits the page.

Two parts of DVWA are specifically designed to be vulnerable to XSS. Let's start with the reflected XSS category.

1. Start a "fake" web server on port 8000 on `kali-vm` using Ncat, a variant of Netcat created by the Nmap developers that makes it easier to listen for multiple and concurrent incoming connections:

   ```
   ncat -lkc "perl -e 'while (defined(\$x = <>)){ print STDERR \$x; last if \$x eq qq#\\r\\n
   # } print qq#HTTP/1.1 204 No Content\\r\\n#'" 8000
   ```

   When a web browser connects on port 8000, Ncat will spawn a Perl program that reads each line of the HTTP request, stops reading when the end of the request header is reached, and responds with a simple HTTP response (`204 No Content`).

2. Open either Chrome or Firefox. If you choose to use Chrome, you'll need to bypass its built-in XSS prevention mechanism: in Burp Suite, select the **Proxy** tab, then the **Options** tab, enable the **Disable browser XSS prevention** rule in the *Match and Replace* list, and enable Burp's proxy in Chrome. This will inject the `X-XSS-Protection: 0` header into all HTTP responses that pass through Burp, disabling Chrome's XSS Auditor (which blocks the execution of a script if its source code is found in the HTTP

*Thanks to Chris Novakovic `c.novakovic@imperial.ac.uk` for preparing this material.

request). You can turn off interception in Burp: the new header will still be injected into the response.

3. Visit `http://10.6.66.42/dvwa/` in your chosen browser.
4. Log in to DVWA with the user name `admin` and the password `password`.
5. From the left-hand menu, select **XSS (Reflected)**.
6. This part of DVWA allows you to enter your name into a text box. After entering your name and clicking the **Submit** button, DVWA greets you using the name you gave. (The PHP source code for this part of the page is shown when you click the **View Source** button.)
7. Craft a malicious URL that mounts an XSS attack against logged-in visitors to this page: when the URL is accessed, the resulting page should contain an injected script that causes the victim's web browser to leak their session cookie (`PHPSESSID`) to your "fake" web server by triggering a HTTP request to it that contains the value of that cookie. You'll know if your XSS attack has succeeded by looking at the output from `ncat` in the terminal: if your attack was successful, the web browser will have made a HTTP request to your "fake" web server that will be visible in the terminal, with the session cookie encoded somewhere in the request.

If you're stuck, you may find the *OWASP XSS Filter Evasion Cheat Sheet* (`https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`) helpful: it lists some of the common ways you can work around the presence of anti-XSS blacklists in web application code. It's also useful to bear in mind that HTML parsers in web browsers are incredibly forgiving of malformed HTML documents, and that an XSS attack may work even if you can't find a way to get well-formed HTML past the blacklist.

Move through the security levels as you find attacks that succeed. This category is completed when you successfully perform a reflected XSS attack against the code for the *high* security level.

After completing the reflected XSS category, move on to the stored XSS category.

1. From the left-hand menu in DVWA, select **XSS (Stored)**.
2. This part of DVWA behaves like a guestbook, and allows you to enter your name and a message that will be displayed to all visitors to the page. Enter your name and message into the boxes, and click the **Sign Guestbook** button to save your message in the DVWA database. (The PHP source code for this part of the page is shown when you click the **View Source** button.) From now on, every time the page loads, all previously-submitted messages are displayed further down on the page.
3. Submit a malicious name and/or message that gets stored in the DVWA database and causes the web browsers of future visitors to this page to leak their session cookie to your "fake" web server, as before.

Move through the security levels as you find attacks that succeed, making sure you reset the DVWA database after you complete each level so you begin the next level with a single benign guestbook message:

1. From the left-hand menu in DVWA, select **Setup / Reset DB**.
2. Click the **Create / Reset Database** button.

When you find a stored XSS attack that succeeds against the code for the *high* security level, you've completed DVWA's XSS challenges.

1. Which lines of the DVWA source code in each category attempt to prevent XSS attacks? Why are they inadequate?

2. How could these vulnerabilities in DVWA be fixed properly? Compare your answer with the source code for the *impossible* security level (which is invulnerable to XSS) in each category.