An intro to XSS attacks and the security of a system

4 May 2021

By: Joshua Doucet and Trung Nguyen

Abstract

In this research cross-site scripting attacks are known as XSS. We learned these attacks can be dangerous and security measures are needed to prevent such attacks. This paper lays out more on what XSS attacks are, the problems they cause, and why they are so common. To further help understand and prevent XSS attacks the paper also goes over how they are deployed and the different types of XSS attack methods. Most of the paper uses JavaScript as a prime example of one of the script languages used to perform these attacks. The security of the OS is difficult to hijack using an XSS attack, but it is not impossible, and some methods are explained below. Lastly, the paper provides learning resources on how to prevent XSS attackers from breaching sensitive information that resides within web applications. There are many tools that can be used to test and detect XSS attacks.

Section 1: Introduction

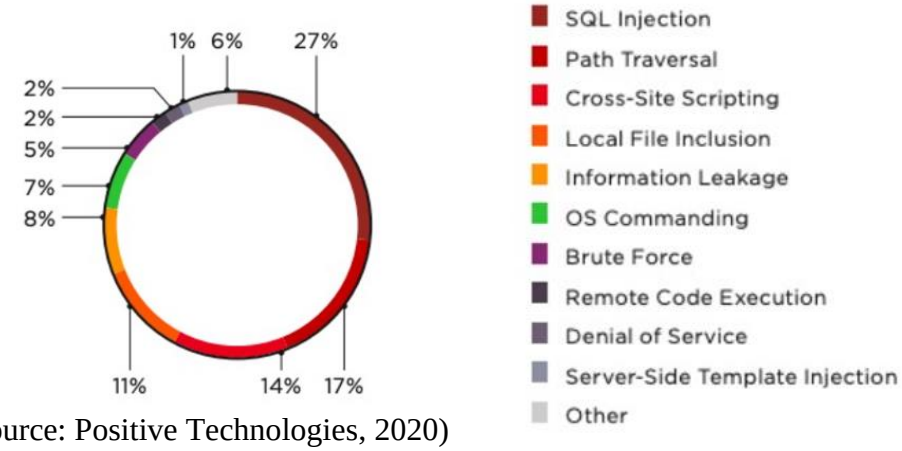
The computing world is full of vulnerabilities that leave computer systems open to information leakage as well as information modification among other concerns. Cross-site scripting known as XSS is a method of compromising information through malicious scripts that are injected into web applications with the intent to run these scripts on end-user devices running a web browser (Positive Technologies, 2020).

Section 2: Problem

What is the problem with XSS attacks and why are they so common?

XSS attacks typically target web servers that utilize JavaScript because of its tight integration with almost all web browsers used today. XSS attacks are common and dangerous because controlled scripting languages like JavaScript are widely used in web clients which interact with web servers holding precious data. “An XSS attack can turn a web application or website into a vector for delivering malicious scripts to the web browsers of unsuspecting victims” (Positive Technologies, 2020). Furthermore, XSS attacks can be used to create memory corruption errors in web applications to leak address information which can be sent back to a web client over a network by using corrupted status pointer (Szekeres et all, 2013). More generally XSS attacks can be used to read data from web servers, intercept user input, deface web pages, and inject malicious code into web servers. These attacks can allow actors to hijack an online account, spread worms, access browser history/cookies and clipboard content, control a browser remotely, and inspect/exploit web applications. Having access to a device’s clipboard can allow an attacker to obtain all sorts of valuable information such as personally identifiable information, login passwords, and financial information (Veracode, 2021). A 2019 analysis found that 14% of web application attacks were the result of XSS attacks, thus making XSS the third most common web application vulnerability. Also, a similar analysis the same year found that 77% of sampled web application tested positive for XSS vulnerabilities (Positive Technologies, 2020).

Known Web Application Attacks by Vulnerability Type



(Source: Positive Technologies, 2020)

Deploying an XSS attack

An XSS attack is triggered when a user visits a site or clicks on an advertisement for example that triggers a malicious script. This script may notice that the user has their bank information open in one of their tabs and attempt to transfer money out of their account. Thankfully, browsers are attempting to prevent this scenario by implementation SOP or the Single Origin Policy. What this does is ensures that scripts on a page only have access to the data that is meant for that page. Unfortunately, cybercriminals have found ways to bypass the SOP by spoofing their network identities (Positive Technologies, 2020).

So how does a site become embedded with an XSS vulnerability? There are two major steps that an attacker must accomplish to deploy a successful XSS attack. Step 1 is embedding user data onto a web page. This means attackers need to embed pieces of malicious code into web page content delivered to end users. Malicious code can be injected when web application inputs are not validated prior to being used by the web server (Positive Technologies, 2020). For example a user (perhaps an attacker) types <script> alert("Access Granted")</script> into a search field. Either the script will be ignored because the web app developers checked the input and the application realized that it should not execute scripts, or the developers may have overlooked the input field and the script would be executed assuming that the user input is echoed back to the user. The user will be presented with an alert box that says "Access Granted." This scenario depicts unwarranted scripts being executed by the web server, and vulnerabilities like this can be used to execute scripts that can do a lot of harm.

Once an attacker has found a way to execute scripts on the server, they will need to embed malicious code into the web application. A relatively harmless example is depicted below where a script is executed to append a name to the web page that is returned by a web server. This code can be placed into an input field on a web page that does not check for input validation, and results in unintended content being displayed on a web client.

```
protected void doGet(HttpServletRequest request, HttpServletResponse resp) {
    String firstName = request.getParameter("firstName");
    resp.getWriter().append("<div>");
    resp.getWriter().append("Search for " + firstName);
    resp.getWriter().append("</div>");
}
```

(Positive Technologies, 2020)

Although the above code appears harmless, the code depicts that an end user/attacker can access information stored on the server. This means that sensitive information could be obtained by typing scripts into an input field on a web page. Furthermore, when input is not validated on a page, the scripts that are entered into the input fields could be stored as persistent data on the web server's database and be used at a later time to attack other users of the web site. Scripts injected into the web server in this way have access to domain cookies, API calls, and other data (Positive Technologies, 2020).

An XSS attack is primed once an attacker embeds attack scripts into a web app. The next step to deploying an XSS attack is having the infected website attack the users that are making requests to the web server. When users navigate to the compromised page, the malicious code may execute when a certain event occurs. For example, an XSS attack script may be injected by an attacker typing JavaScript into an online discussion form. If the discussion message is not validated, the script may be stored in a database that holds the message contents. Then, when other users navigate to the message board, the server will provide all the messages in that thread, and then the browser will display each message. If the messages are not sanitized, then the messages including JavaScript may be executed on the browser client of each user that visits the discussion board (Academind, 2020). These scripts could be used to collect personal information about users or redirect users to a URL that downloads malware to their host machines.

Types of XSS attacks

There are three main types of XSS attacks that include: reflected (non-persistent), stored (persistent), and DOM-Based (Document Object Model). Reflected attacks target the client machine receiving an HTTP request. The client device sends the attack to the web server and the server reflects it back to the HTTP client for execution. So, if a web server echos user input, then an attacker could place malicious code into a user input field to retrieve some server info by utilizing the below JS function.

```
protected void info(HttpServletResponse resp, String info) {
    resp.getWriter().append("<h4>Info</h4>");
    resp.getWriter().append(info);
}
```

(Positive Technologies, 2020)

The second type of XSS is the stored attack. These attacks persistently store attack vector code on the server side of HTTP requests. An attacker can insert html and JavaScript into databases as raw data, and when a client requests that raw data it may get executed on the client browser.

```
protected void doGet(HttpServletRequest rq,
HttpServletResponse resp) {
    String name = rq.getParameter("NAME");
    StringBuffer res = new StringBuffer();
    String query = "SELECT fullname FROM emp WHERE name = '" +
name + "'";
    ResultSet rs = DB.createStatement().executeQuery(query);
    res.append("<table class=\"table\"><tr><th>Employee</th>
</tr>");
    while (rs.next()) {
        res.append("<tr><td>");
        res.append(rs.getString("fullname"));
        res.append("</td></tr>");
    }
    res.append("</table>");
    resp.getWriter().append(res.toString());
}
```

(Positive Technologies, 2020)

The third type of XSS vulnerability is the DOM based attacks which causes scripts to execute in the server context. Below, the variable named message could be regular data, but it could also be html and JavaScript since the html() function does not sanitize its inputs.

```
<div id="message-text">This is a warning alert</div>

  function warning(message) {

      $("#message-text").html(message);

      $("#message").prop('style', 'display:inherit');

  }
```

(Positive Technologies, 2020)

XSS and the Security of an OS

Web browsers typically run JavaScript in a controlled environment that gives the code minimal access to the host machines operating system, thus it is difficult for an XSS attack to directly compromise a machine's OS (Acunetix, 2020). However, XSS attacks can be used to leak memory addresses which can be used to deploy more severe server attacks. Getting address information used to be difficult, but modern applications such as web browsers PDF viewers and more use controlled scripts like JavaScript, VBScript, and ActionScript. These controlled scripts can be used to snag address information to construct malicious payloads that can execute during runtime on the target machine. JavaScript can be used to leak memory contents by overwriting the length field of something like a string before reading its contents. The extra length exposes other data in memory (Szekeres et all, 2013).

Furthermore, XSS attacks can be used to manipulate users into downloading and installing malware which can directly compromise an OS. Also, XSS attacks can be used to obtain copious amounts of data including login information from the comprised web application's users (Acunetix, 2020). And with these logins, attackers then have the keys to other web applications, databases, or operating system environments. Overall, there are many ways that XSS vulnerabilities can be used to compromised information and computing systems.

Section 3: Solution

Testing and Detecting XSS vulnerabilities

If the source code of an application is available then static code analysis can be used to detect certain breeds of XSS vulnerabilities. This implies that you manually inspect the web application code to see if inputs can be used to deploy XSS vulnerabilities. Automated XSS vulnerability detection can be used as well, but the quality of XSS vulnerability detection is reliant on the type of scanner used. Keep in mind that these scanners can also produce false positives. Also, certain XSS attacks can also be written in a way that tricks scanners to bypassing the exploit and not flagging it as a vulnerability. When looking for XSS vulnerabilities there are 4 structures where scripts are commonly injected: Markdown editors that allow users to insert custom html, Text-to-emoji converters, Text to link converters, and Text-to-picture converters. Furthermore, injected script tags, onload attributes, img, elements, input elements, and div elements can all be used to inject an XSS vulnerability into a web application. Each of these elements should be inspected to determine if an XSS attack is possible based on user input or database output.

Preventing XSS attacks

The best measures that can be in place to prevent XSS vulnerabilities within web applications is to validate user input and sanitize database output. Data must be checked to see if it contains html or JavaScript whenever a user can insert data into a web application. Similarly, data that is being fetched from a database should be sanitized to ensure that it does not contain html or JavaScript before it is included in the DOM structure of a web page.

Section 4: Related Work

Tools for Preventing XSS Attacks

XSS vulnerability testing tools are not created equal, and all have their advantages and disadvantages. These testing tools typically fall into 2 categories: open source and enterprise. The pros of open source XSS scanners are: being free of charge and have many options to choose from. The cons of open-source scanners are: they can be low quality, have minimal support, and may integrate poorly with other tools.

On the other hand, there are enterprise XSS scanners. The pros include smooth integration with other application tools, they include vendor support with regular updates, and the scanners tend to perform better than open-source variants. The cons of enterprise scanners include high price tags, and some scanners may be cumbersome and produce many false positive threat results.

Some XSS Vulnerability Testing Tools

XSSer – This tool uses an automatic framework to detect XSS vulnerabilities and tries to bypass various methods used to inject code into the application. https://github.com/epsylon/xsser

XSSsniper – This tool is used to scan URLS for GET parameters and injects a XSS payload into them. https://github.com/gbrindisi/xsssniper

XSStrike – This tool also scans an application to detect for vulnerabilities and exploits. It uses a fuzzing engine that increases its accuracy. https://github.com/s0md3v/XSStrike

More Tools - https://linuxhint.com/free_xss_tools/

Section 5: Conclusion

In conclusion, from what we have researched XSS attacks can be dangerous as they can reveal sensitive information within web applications and browsers. With most of today's web services relying on JavaScript, sites are more vulnerable to XSS attacks, especially when proper security measures are not in place. As time and technology progresses, so does software and the vulnerabilities that these software bring. To prevent those vulnerabilities, it is best to understand how they work and methods that can potentially prevent them. It is also good to be aware of what you choose to download and press while surfing online. With many tools available and being well educated in security, everyone can benefit from being prepared to prevent attackers who are looking to ruin someone else's day. Be alert and be aware, vulnerabilities and exploits are around every corner.

References

Academind. (2020, July 16). Running a xss attack + how to defend. Retrieved April 14, 2021, from https://www.youtube.com/watch?v=oEFPFc36weY

Acunetix. (2020, July 14). What is cross-site scripting and How Can you fix it? Retrieved March 30, 2021, from https://www.acunetix.com/websitesecurity/cross-site-scripting/

Positive Technologies. (2020, December 02). What is a cross-site scripting (XSS) ATTACK? Retrieved March 30, 2021, from https://www.ptsecurity.com/ww-en/analytics/knowledge-base/what-is-a-cross-site-scripting-xss-attack/

Szekeres, L., Payer, M., Wei, T., & Song, D. (2013). SoK: Eternal war in memory. Paper presented at the 48-62. https://doi.org/10.1109/SP.2013.13

Veracode. (2021). What is Cross-Site Scripting? Cross-site scripting prevention. Retrieved April 05, 2021, from https://www.veracode.com/security/xss