Website Cross-Site Scripting (XSS) Attack via Windows Terminal

What is xssbase?

XSSbase, developed by ByteBreach, is a robust command-line tool meticulously crafted for probing Cross-Site Scripting (XSS) vulnerabilities within web applications. This tool streamlines the XSS testing process by automating the injection of payloads into input fields, ensuring a comprehensive assessment of potential security weaknesses.

With a focus on user convenience and efficiency, XSSbase caters specifically to Windows environments, providing a reliable platform for security professionals and developers alike to bolster the integrity of their web applications. Its intuitive command-line interface facilitates seamless interaction, allowing users to execute targeted XSS tests with precision.

The dedication to enhancing web security is evident in XSSbase's capability to identify and report XSS vulnerabilities effectively. This tool not only identifies areas of concern but also empowers users with actionable insights, thereby fostering a proactive approach to web application security.

By attributing the development of XSSbase to ByteBreach, a reputable name in cybersecurity, users can trust in the tool's quality, reliability, and commitment to addressing XSS vulnerabilities comprehensively. You can access XSSbase on GitHub at https://github.com/ByteBreach , where the community can contribute to its ongoing evolution and enhancement.

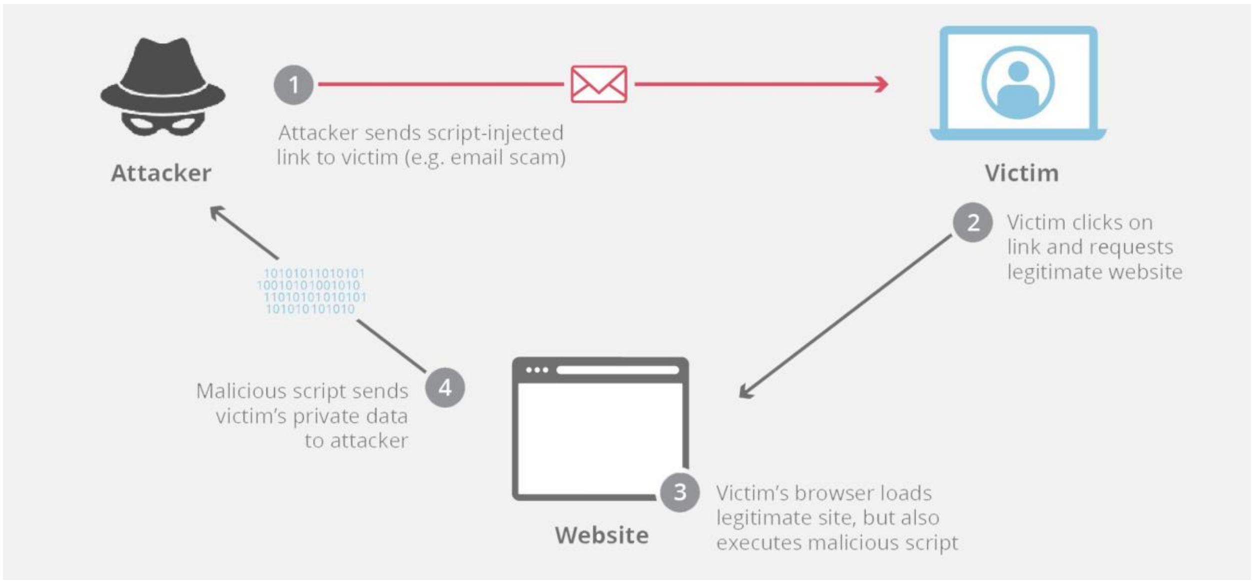
What is XSS?

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

Most popularly, it is either added to the end of a url or posted directly onto a page that displays user-generated content. In more technical terms, cross-site scripting is a client-side code injection attack.

How does xss work?


Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
3
Victim's browser loads legitimate site, but also executes malicious script
Website
4
Malicious script sends victim's private data to attacker
10101011010101
10010101001010
11010101010101
101010101010

Below are two examples :

Reflected XSS Example :

Link : http://example.com/search?q=<script>alert('XSS');</script>

In this example, the search query parameter q includes a script tag that would trigger an alert if the server reflects this input back into the web page without proper encoding.

Stored XSS Example :

Link : https://example.com/comments?post_id=123

Assume that a comment containing a malicious script like <script>alert('XSS')</script> is posted. When users view the post with ID 123, the script executes.

About ByteBreach

ByteBreach: Our purpose transcends mere protection; it’s about empowering individuals and organizations to navigate the digital realm with confidence and peace of mind. We believe that in a world where cyber threats loom large, knowledge is power. That’s why, in addition to our cutting-edge security solutions, we’re dedicated to education and awareness initiatives, equipping people with the tools and know-how to stay one step ahead of cybercriminals.

But our purpose extends beyond defense; it’s about harnessing the transformative potential of technology to create a better world. We’re not just safeguarding data; we’re fueling innovation, driving progress, and laying the foundation for a more connected, efficient, and secure future. ByteBreach isn’t just a cybersecurity firm; we’re architects of a new digital era, where trust, transparency, and resilience are the cornerstones of the online experience. Join us as we forge ahead, building bridges to a brighter tomorrow in the ever-expanding landscape of cyberspace.

Let’s see how to install this tool on Windows

Installation

```
$ pip install xssbase
```

If you have Python installed on your computer, you can easily run it

Python Installation Link : Click

You can also test your running website locally using this tool

With this you can check the vulnerabilities of most sites and you can also try it locally.

To run this tool you need payload if you don’t have payload you can download it by clicking below link

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

If you are reluctant to install the payload, you can use some of the default payloads

How to use is given below :

```
$ xssbase –url https://example.com/
```

- By running the above command we will perform an xss attack on that site
- If you run the above command you don’t need to put your payload there
- The above command will test the website with the default payloads
- You can also try your localhost
- $ xssbase –url https://127.0.0.1:8080

Works

How this xssbase works is given below to give you an idea of how the xssbase works

Vulnerable Code Explanation

Code :

```
<!DOCTYPE html>

<html>

<head>

  <title>XSS Vulnerable Form</title>

</head>

<body>

  <form action=”#” method=”post”>

    <label for=”name”>Enter your name:</label><br>

    <input type=”text” id=”name” name=”name”><br>

    <input type=”submit” value=”Submit”>

  </form>


  <div>

    Welcome, <?php echo $_POST[‘name’]; ?>!

  </div>

</body>

</html>
```

How an Attacker Exploits this Vulnerability

1) **Submit Malicious Input**: An attacker could submit the following input in the *name* field
- <script>alert('XSS Attack!');</script>
2) **Server-Side Code Execution**: When the form is submitted, the server processes the input and includes it directly in the HTML output without sanitization.
3) **Execution in Browser**: The user's browser receives the HTML with the embedded script tag and executes it, resulting in a JavaScript alert popping up:
- Welcome, <script>alert('XSS Attack!');</script>!

How xssbase Could Be Used

1) **Scanning for Vulnerabilities**: XSSBase scans the web application for common XSS vectors. It might identify that user input from the *name* field is being directly included in the HTML output.
2) **Testing for XSS**: xssbase attempts to inject various payloads into the name field to see if they are executed. One such payload might be:
- <script>alert('XSS Base Test');</script>
3) **Identifying Vulnerability**: If XSSBase successfully injects and executes this payload, it confirms that the application is vulnerable to XSS.

Usually we type the payload in the input box. But with this tool, all these are done automatically : xssbase --url https://127.0.0.1:8080

Example of Stored XSS Vulnerability

Vulnerable Code

```
<!DOCTYPE html>
<html>
<head>
  <title>Comment Section</title>
</head>
<body>
  <form action=”submit_comment.php” method=”post”>
    <label for=”comment”>Enter your comment:</label><br>
    <textarea id=”comment” name=”comment”></textarea><br>
    <input type=”submit” value=”Submit”>
  </form>

  <h2>Comments:</h2>
  <div id=”comments”>
    <?php
    // Assuming $comments is an array fetched from the database
    Foreach ($comments as $comment) {
      Echo “<p>{$comment}</p>”;
    }
    ?>
  </div>
</body>
</html>
```

Vulnerable submit_comment.php

```
<?php
// Assuming a database connection is already established

If ($_SERVER['REQUEST_METHOD'] === 'POST') {
   $comment = $_POST['comment'];
   $sql = "INSERT INTO comments (comment) VALUES ('$comment')";
   // Execute the query to store the comment
   $db->query($sql);
}

// Redirect back to the comment page
Header('Location: comment_page.php');
?>
```

How an Attacker Exploits Stored XSS

1) **Submit Malicious Comment**: An attacker submits a comment containing a script tag:
- <script>alert('Stored XSS Attack!');</script>
2) **Stored in Database**: The comment is stored in the database without sanitization:
- INSERT INTO comments (comment) VALUES ('<script>alert('Stored XSS Attack!');</script>');
3) **Displayed to Users**: When users visit the comments page, the script is executed in their browsers:
- <p><script>alert('Stored XSS Attack!');</script></p>

How xssbase Can Identify Stored XSS

1) **Scanning Input Points**: xssbase scans the web application to identify input fields that store data in a persistent manner (e.g., comment sections, profile updates).
2) **Injecting Payloads**: xssbase injects various XSS payloads into these fields and monitors if the payloads appear in the rendered output when the data is retrieved from the storage.
3) **Detection**: If the injected payloads are executed when the stored data is displayed, xssbase identifies and reports the stored XSS vulnerability.

Command-Line Usage Example

Basic Command Format:

```
$ xssbase –url https://example.com –payload payload.txt
```

Command Breakdown:

- `xssbase`: The tool's executable command.
- `--url https://example.com`*:* The target URL to test for XSS vulnerabilities.
- `--payload payload.txt`: A file containing various XSS payloads to test against the target.

Example Scenario

Vulnerable Web Page (example.com/comment_page.php)

```
<!DOCTYPE html>
<html>
<head>
  <title>Comment Section</title>
</head>
<body>
  <form action=”submit_comment.php” method=”post”>
    <label for=”comment”>Enter your comment:</label><br>
    <textarea id=”comment” name=”comment”></textarea><br>
    <input type=”submit” value=”Submit”>
  </form>

  <h2>Comments:</h2>
  <div id=”comments”>
    <?php
    // Fetch comments from the database
    $comments = getCommentsFromDatabase();
    Foreach ($comments as $comment) {
      Echo “<p>{$comment}</p>”;
    }
    ?>
  </div>
</body>
</html>
```

Payload File (payload.txt)

<script>alert(‘XSS Attack’);</script>

<img src=x onerror=alert(‘XSS Attack’)>

Using xssbase

1) **Prepare Payload File:**

- Create a file named payload.txt with your payloads:

<script>alert(‘XSS Attack’);</script>

<img src=x onerror=alert(‘XSS Attack’)>

2) **Run XSSBase Command:**

- Execute the following command in your terminal:

-
```
$ xssbase –url https://example.com/comment_page.php --payload payload.txt
```

3) **Expected Output:**

- xssbase will attempt to inject each payload from payload.txt into the input fields of https://example.com/comment_page.php. If a payload successfully triggers an XSS vulnerability, xssbase will report it. The output might look something like this:

```
[INFO] Testing XSS vulnerabilities on https://example.com/comment_page.php
[INFO] Using payload file: payload.txt
[SUCCESS] Payload ‘<script>alert(‘XSS Attack’);</script>’ triggered an alert!
[SUCCESS] Payload ‘<img src=x onerror=alert(‘XSS Attack’)>’ triggered an alert!
[INFO] Vulnerability report saved to xssbase_report.txt
```

Remember this tool is mainly designed for developers and never for cybercriminals

Thank You

Thank you for reading this document. I hope you found the information helpful and insightful. If you have any questions or feedback, please don't hesitate to reach out.

Team : ByteBreach

Enjoy experimenting with xssbase