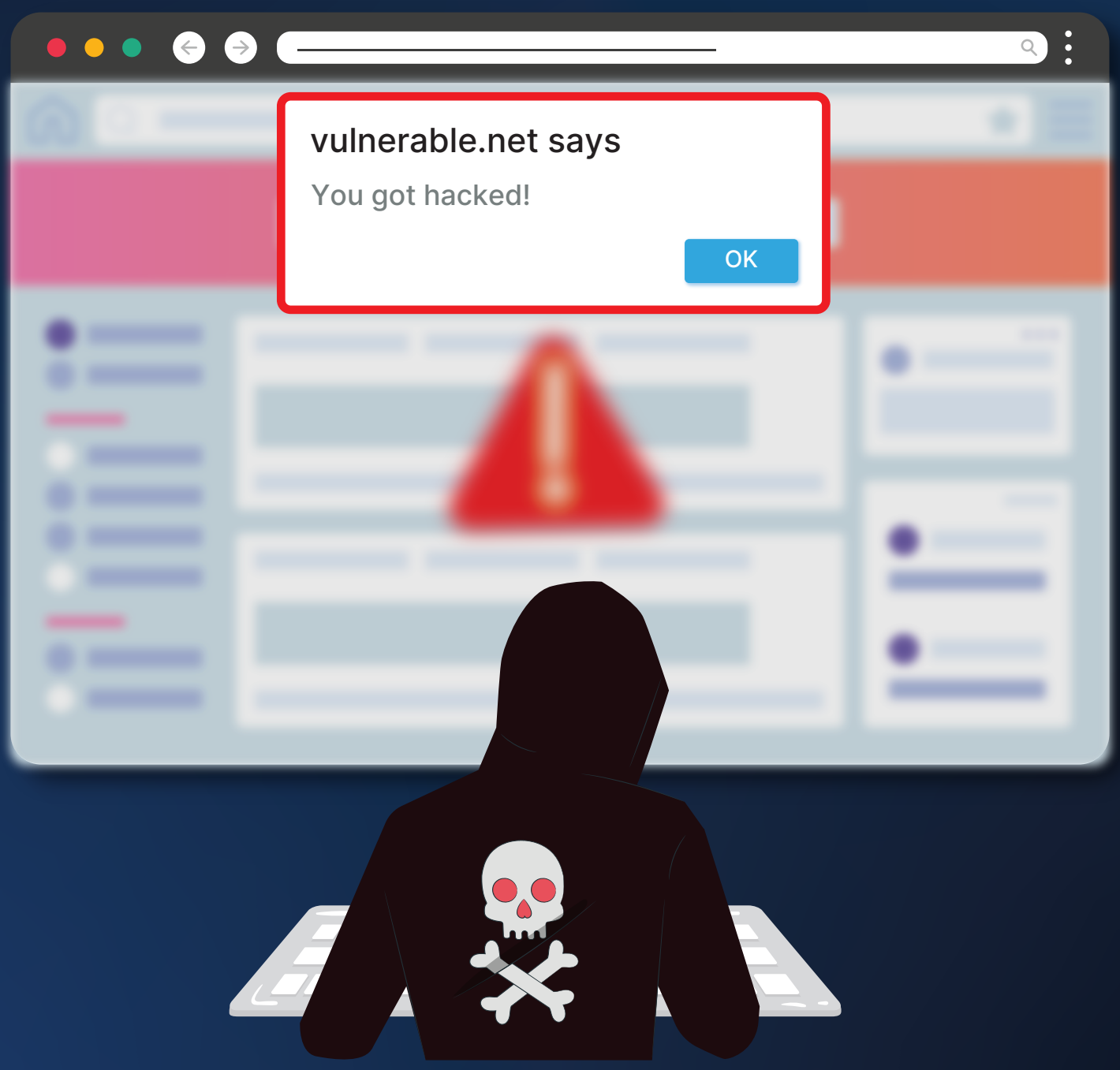
SECURITYBOAT
Frontline Of Your Business

CROSS SITE SCRIPTING HANDBOOK



www.securityboat.net

SECURITYBOAT
Frontline Of Your Business

Table of Contents

1. What is Cross Site Scripting? 01

2. XSS and Client-Side Code 01

3. How does XSS works? 02

4. Finding and Testing for XSS Vulnerabilities 03

   4.1 Automated Detection with Burp Suite 03

   4.2 Manual Testing for Reflected and Stored XSS 03

   4.3 Testing for DOM-based XSS 03

   4.4 Burp Suite's Advanced Capabilities 03

5. Cross Site Scripting Attack Techniques 04

   5.1 Stored XSS (Persistent XSS) 04

   5.2 Reflected XSS (Non-Persistent XSS) 04

   5.3 DOM-Based XSS (Client-Side XSS) 04

6. XSS Bypass 05

7. Impact 06

8. Mitigation 06

9. QR Code 07

1. What is Cross Site Scripting?

Cross-site scripting (XSS) is a client-side code injection attack where an attacker embeds malicious code onto a legitimate website, which then executes when a victim loads the site. This code can be injected in various ways, such as appending it to the end of a URL or posting it on a page that displays user-generated content. XSS attacks occur when a web application sends this malicious code, usually in the form of a browser-side script, to another user. These attacks exploit flaws in web applications that fail to properly validate or encode user input.

This vulnerability compromises user interactions with a vulnerable application by bypassing the same-origin policy, which is meant to isolate different websites from each other. It allows attackers to impersonate the victim user, perform any actions the user can, and access the user's data. If the victim has privileged access, the attacker may gain full control over the application's functionality and data.

2. XSS and Client-Side Code

Client-side code refers to JavaScript code that runs on a user's machine, typically executed by the web browser after loading a web page. This contrasts with server-side code, which runs on the web server. Client-side code enhances the interactivity of web pages by allowing interactions to occur directly on the user's device, reducing the need for constant communication with the web server. This results in faster and more reliable interactive content. A common application of client-side code is in browser-based games, which benefit from smooth operation regardless of connectivity issues.

Client-side code is a cornerstone of modern web development, utilized by most contemporary websites. However, its widespread use has also led to increased vulnerabilities, particularly cross-site scripting (XSS). XSS has become one of the most frequently reported cybersecurity issues, affecting major platforms such as YouTube, Facebook, and Twitter.

3. How does XSS works?

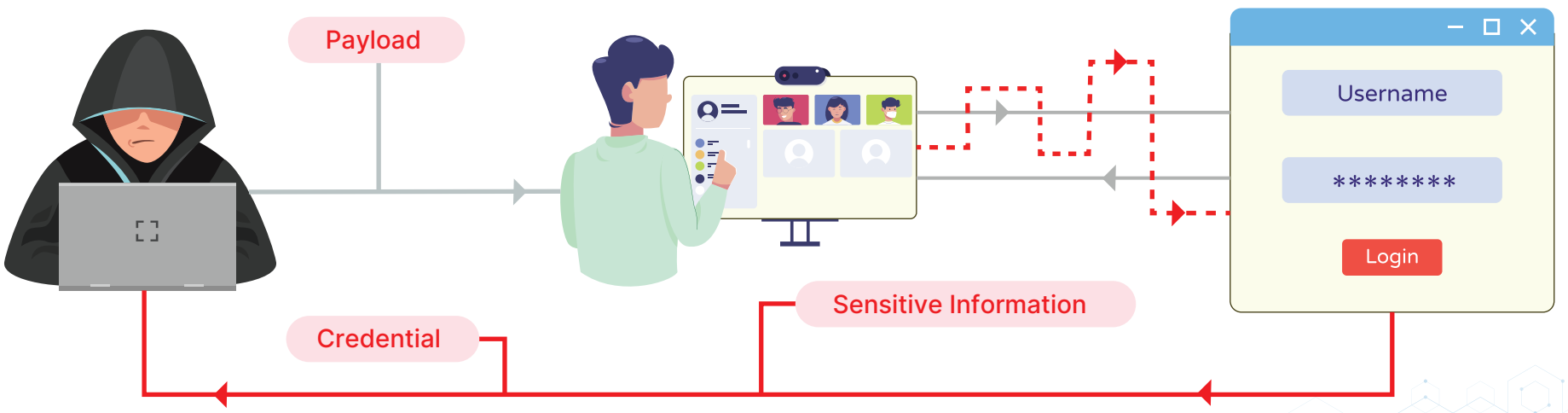
1. **Injection of Malicious Code:** The attacker identifies a web application vulnerability that fails to properly validate or encode user input. They then inject malicious code, often in the form of JavaScript, into the application. This can be done through various means such as submitting a form, adding a comment, or manipulating a URL.

2. **Delivery to the Victim:** The malicious code is embedded within the website and delivered to the user's browser. This can happen when the user visits a compromised page or clicks on a malicious link.

3. **Execution in the Browser:** When the victim's browser loads the compromised web page, it executes the malicious script as if it were legitimate. Because the script runs in the context of the trusted site, it gains access to any data or functionalities that the user has on that site.

4. **Exploitation:** The attacker can now use the script to perform a variety of malicious activities. These may include:Stealing cookies, session tokens, or other sensitive information.
   - Logging keystrokes to capture passwords or other confidential data.
   - Redirecting the user to phishing sites or other malicious sites.
   - Performing actions on behalf of the user, such as sending messages, making purchases, or changing account settings.

5. **Circumventing Security Policies:** XSS attacks can bypass the same-origin policy, which is designed to prevent different websites from accessing each other's data. By injecting code into a trusted site, attackers can effectively masquerade as the user and interact with the site's data and functionality.



4. Finding and Testing for XSS Vulnerabilities

Discovering XSS vulnerabilities involves both automated tools and manual testing methods to comprehensively assess web applications:

4.1 Automated Detection with Burp Suite

Burp Suite's web vulnerability scanner provides a quick and reliable method to identify the majority of XSS vulnerabilities. It automates the process by scanning web applications for common XSS flaws in various input fields and parameters.

4.2 Manual Testing for Reflected and Stored XSS

- **Reflected XSS:** Manually test by entering unique input (e.g., alphanumeric strings) into each entry point of the application. Identify where this input is reflected in HTTP responses and test each location individually to see if crafted input can execute JavaScript.
- **Stored XSS:** Similarly, input data into fields that store information (e.g., comments, user profiles). Check how this stored data is retrieved and displayed to users. Test if injected scripts can execute when users view the affected pages.

4.3 Testing for DOM-based XSS

- **URL Parameter-based XSS:** Inject unique input into URL parameters. Use browser developer tools to inspect the DOM and test each location to determine if it's exploitable.
- **Non-URL-based DOM XSS:** Review JavaScript code manually to identify vulnerabilities related to document objects like `document.cookie` or functions like `setTimeout.` This process can be intensive due to the need for detailed code analysis.

4.4 Burp Suite's Advanced Capabilities

Burp Suite's web vulnerability scanner integrates static and dynamic analysis of JavaScript, enabling automated detection of complex DOM-based XSS vulnerabilities that are challenging to identify manually. This tool combination enhances efficiency and reliability in XSS vulnerability detection.

5. Cross Site Scripting Attack Techniques

5.1 Stored XSS (Persistent XSS)

Stored XSS (Persistent XSS) occurs when a malicious script is stored on the target server, such as in a database, message forum, visitor log, comment field, etc. When a victim requests the stored information, they retrieve and execute the malicious script. For instance, if an attacker submits a script in a comment section like `<script>alert('XSS Attack');</script>`, every user who views that comment will see an alert box with "XSS Attack". This demonstrates that the script is running in their browser context, affecting all users who access the infected page.

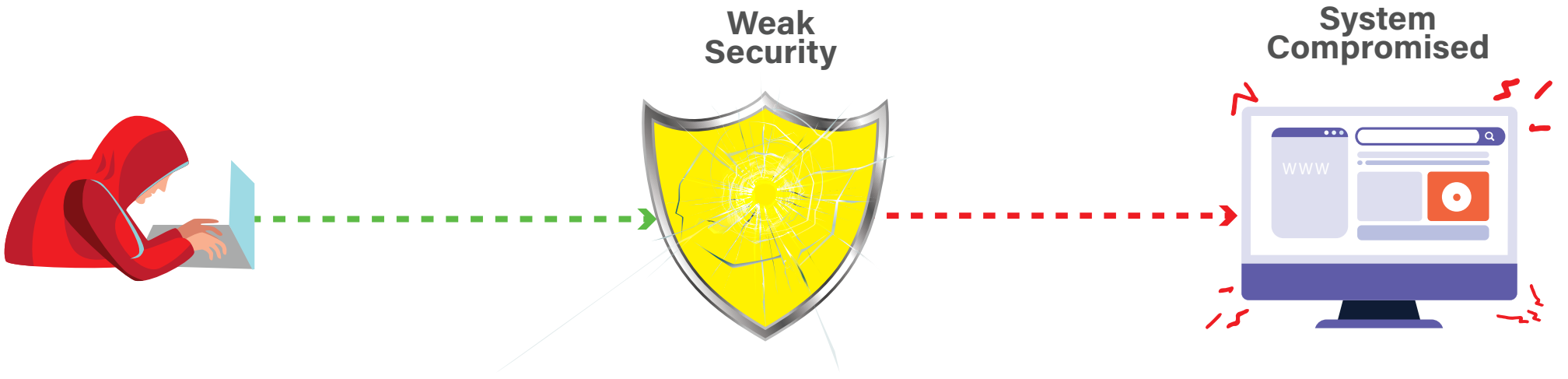
5.2 Reflected XSS (Non-Persistent XSS)

Reflected XSS (Non-Persistent XSS) involves a malicious script being reflected off a web application and executed in the victim's browser. This type of attack is typically delivered to the victim through another route, such as an email or another website. For example, an attacker might send a link containing a malicious script to a victim. When the victim clicks on the link, like `http://example.com/search?q=<script>alert('XSS Attack');</script>`, the script is reflected off the server's search results page and executed in the victim's browser. This results in the script running in the victim's browser context, demonstrating the vulnerability.

5.3 DOM-Based XSS (Client-Side XSS)

DOM-Based XSS (Client-Side XSS) refers to vulnerabilities in client-side code, where the malicious script is executed by modifying the DOM environment in the victim's browser. This occurs when a web application writes user input directly to the DOM without proper sanitization, allowing an attacker to manipulate the input to execute a script. For instance, if a web application contains JavaScript code like `document.write("User input: " + location.hash);`, an attacker can craft a URL such as `http://example.com/page#<script>alert('XSS');</script>`. When the victim visits this URL, the script in the hash fragment is executed, demonstrating the vulnerability and impact of the attack.

6. XSS Bypass



- **HTML Entity Encoding:** Using HTML entities to encode characters so that they are interpreted by the browser as part of the HTML content instead of executable code. For example, <script> can be encoded as `<script>`.
- **URL Encoding:** Encoding characters in URLs to bypass filters. For instance, <script> can be encoded as `%3Cscript%3E`.
- **Double Encoding:** Applying multiple layers of encoding to evade detection. For example, `%253Cscript%253E` represents a double-encoded <script> tag.
- **Base64 Encoding:** Encoding the script in Base64 and using `eval(atob('...'))` to execute it.
- **Event Handlers:** Injecting scripts through event handlers such as onload, onclick, onerror, etc. For example, `<img src=x onerror=alert('XSS')>`.
- **Nested Tags:** Embedding one tag inside another to bypass filters. For example, `<svg><g onload=alert(1)></g></svg>`.
- **Tag Injection:** Using tags that are less commonly filtered, like `<svg>`, `<math>`, or `<iframe>`. For example, `<svg onload=alert('XSS')>`.
- **Polyglot Payloads:** Creating payloads that can be interpreted as valid in multiple contexts (e.g., HTML, JavaScript, CSS). For example, `</script><svg onload=alert(1)>`.
- **Using Alternative Syntax:** Leveraging alternative ways to write JavaScript, such as using `String.fromCharCode` to construct strings.
- **Avoiding Common Keywords:** Obfuscating common JavaScript keywords like script, alert, etc., using different encodings or variable names. For example, using `window['alert']` instead of alert.

7. Impact

- **Data Theft:** Attackers can steal cookies, session tokens, and other sensitive information.
- **Account Hijacking:** Compromised sessions can lead to unauthorized access to user accounts.
- **Credential Theft:** Malicious scripts can capture login credentials.
- **Phishing Attacks:** Users can be redirected to fraudulent sites, leading to credential theft.
- **Malware Distribution:** XSS can be used to deliver malicious payloads to users.
- **Defacement:** Attackers can modify the appearance of a website.
- **Reputation Damage:** Successful XSS attacks can harm a website's reputation and user trust.
- **Financial Loss:** Exploits can result in financial theft or fraud.
- **Denial of Service:** XSS can be used to launch further attacks, including Denial of Service (DoS).

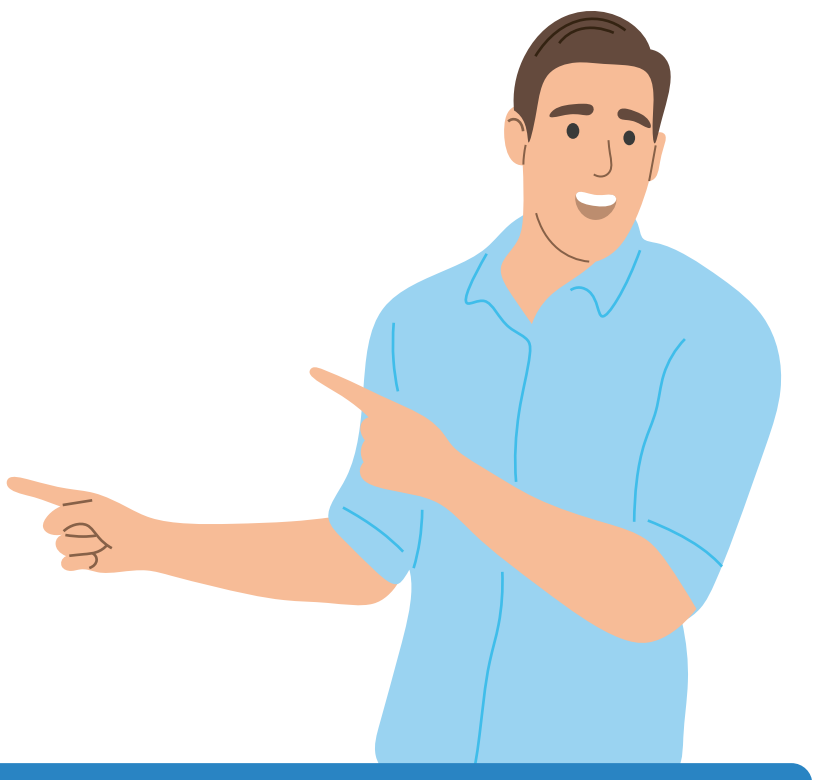
8. Mitigation

- Prevent users from posting HTML in form inputs to reduce the risk of persistent XSS attacks. Use markdown or WYSIWYG editors to allow users to create rich content without HTML.
- Implement rules to ensure only valid data is submitted. An input for "Last Name" should only accept alphanumeric characters. Validation rules should also reject tags and characters commonly used in XSS attacks, like `<script>`.
- Clean data after it's submitted but before it's displayed to another user. Use online tools to sanitize HTML and filter out malicious code injections.
- Set rules for cookie handling to prevent cookie theft via XSS. Tie cookies to specific IP addresses and block JavaScript from accessing cookies.
- Configure a WAF to enforce rules that block reflected XSS attacks. WAFs like Cloudflare's can provide turnkey protection against XSS, DDoS attacks, SQL injection, and other threats.

SECURITYBOAT

Frontline Of Your Business

CROSS SITE SCRIPTING HANDBOOK



Scan QR Code to Download Handbook

www.securityboat.net