Injection Vulnerabilities

Cybersecurity

15.1 Web Vulnerabilities and Hardening

Class Objectives

By the end of today's class, we'll be able to:

- Articulate the intended and unintended functionalities of a web application.
- Identify and differentiate between SQL and XSS injection vulnerabilities.
- Design malicious SQL queries using DB Fiddle.
- Create payloads from the malicious SQL queries to test for SQL injection against a web application.
- Design malicious payloads to test for stored and reflected cross-site scripting vulnerabilities.

Last Week...

We learned about the intended use and functionalities of web application:

- How HTTP requests and responses work to make a web application function.
- How to maintain a user's session using cookies.
- The various stacks of a common web architecture and how the components within these stacks work together.
- How the database component works within a web application.
- How to write and run basic SQL queries against a database.

This Week...

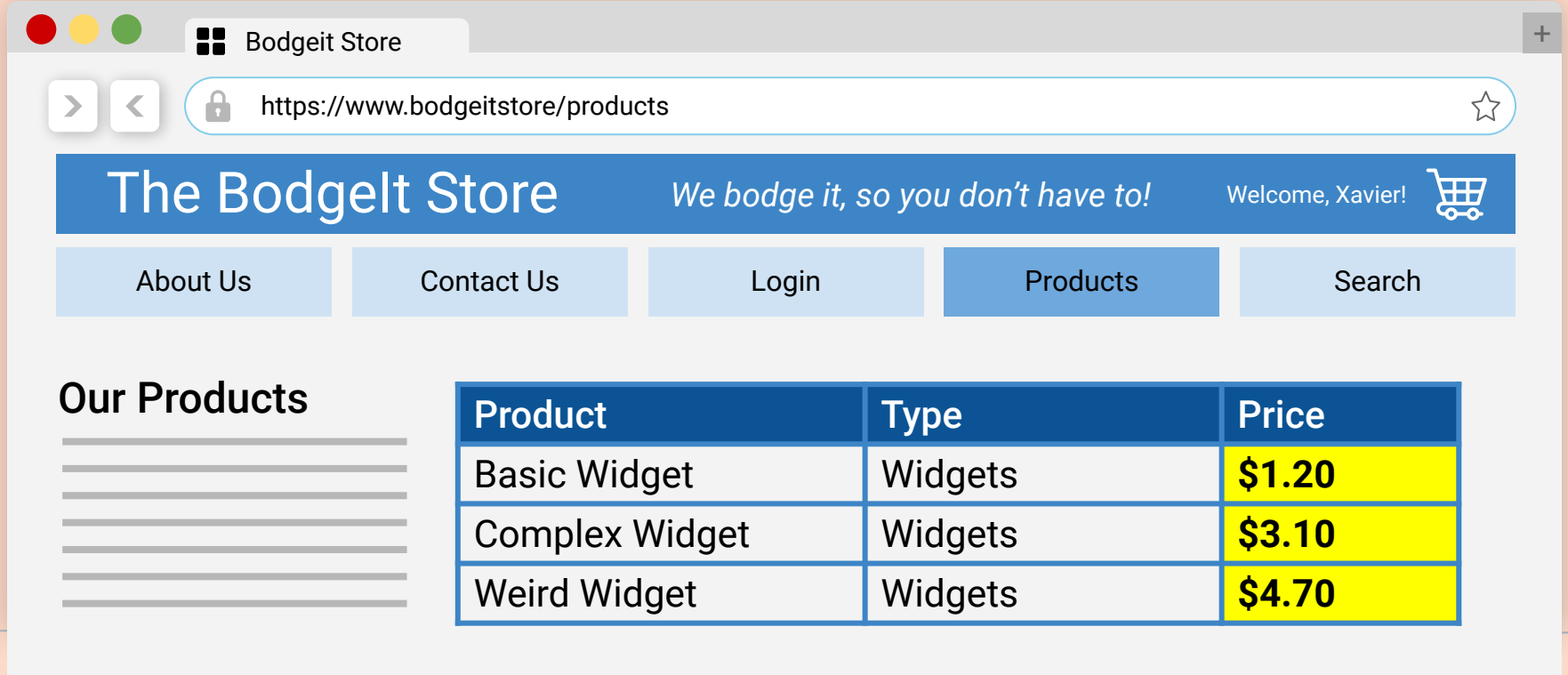
We will learn how to exploit those web applications for unintended uses and weaknesses.

Malicious actors will exploit weaknesses that might exist in these functions to cause unintended consequences.

These weaknesses are called **web vulnerabilities**.

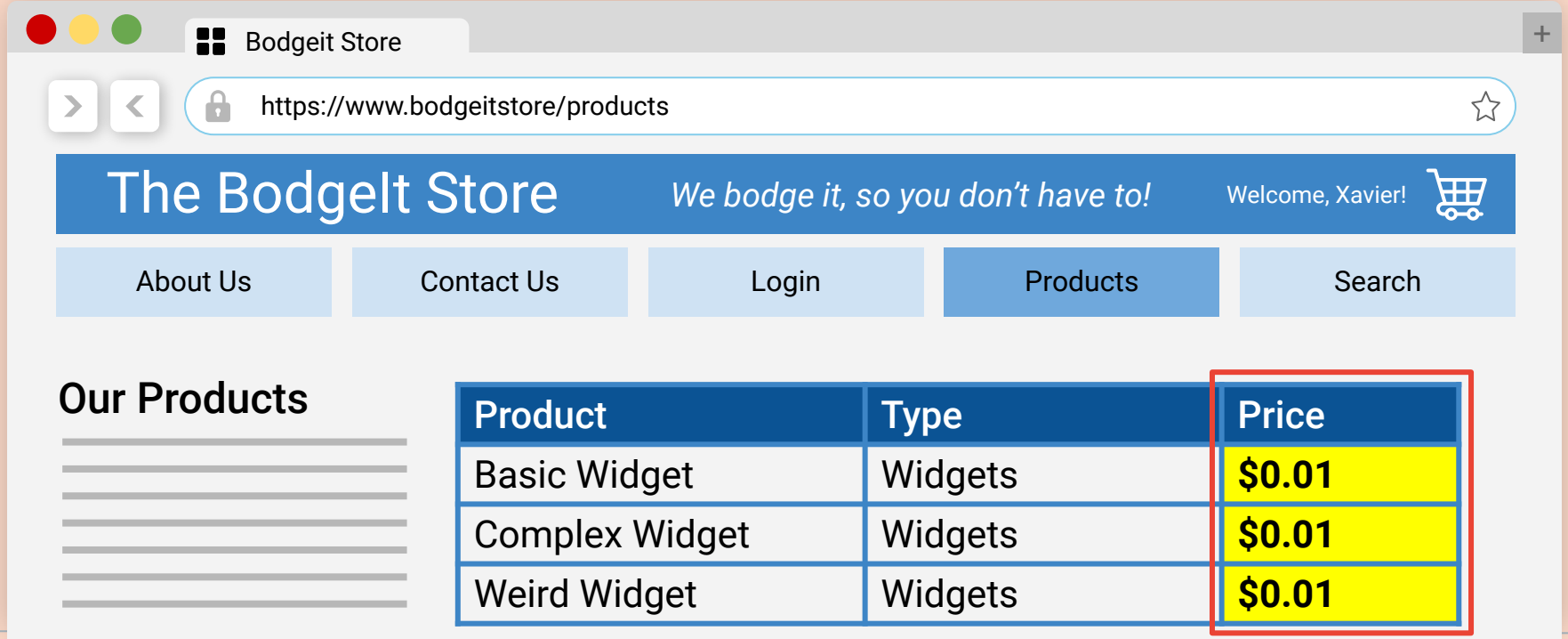
Intended

The intended functionality of the sample webpage allows users to purchase different widgets at various prices.



Unintended

A malicious user could exploit a web application vulnerability and create an unintended result by changing the price of all the widgets to $0.01.



Impacts

Financial impact

The malicious actor can exploit the web application and purchase widgets for a significantly reduced cost

Legal impact

If a web application has a vulnerability that exposes confidential user data, a business could be subject to penalties.

Reputational impact

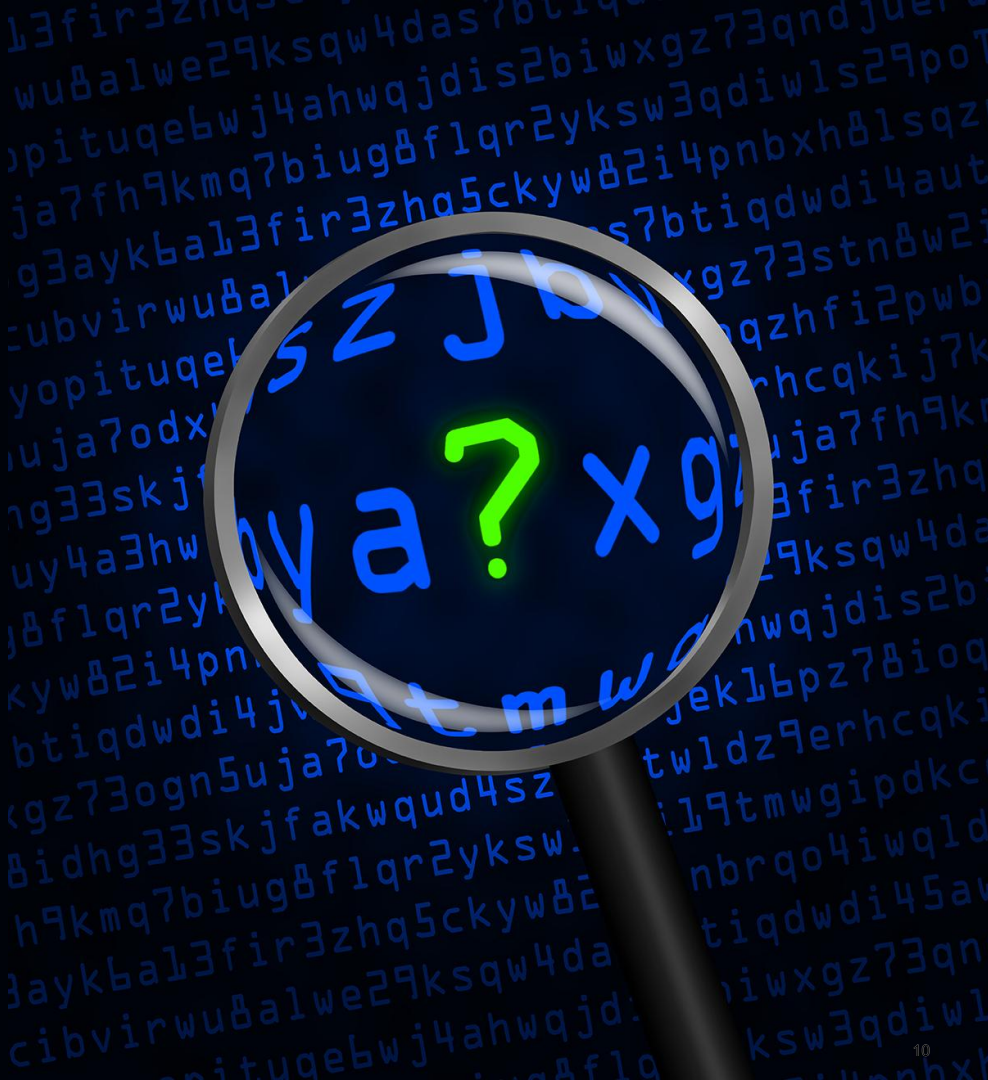
If a web application has a reputation for being down, or dangerous due to web vulnerabilities, their customers will likely search for another business.

OWASP Top 10

The expansiveness of the web and its various technologies, coding languages, and architectures leads to new capabilities. With these capabilities come many potential vulnerabilities.

The Challenge Is...

With so many potential vulnerabilities, how do security professionals identify and manage the top issues that could impact web applications?



Introducing the OWASP Top 10

OWASP

Open Web Application Security Project

A nonprofit foundation that works to improve the security of software through community-led open-source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences.

Source: https://owasp.org/

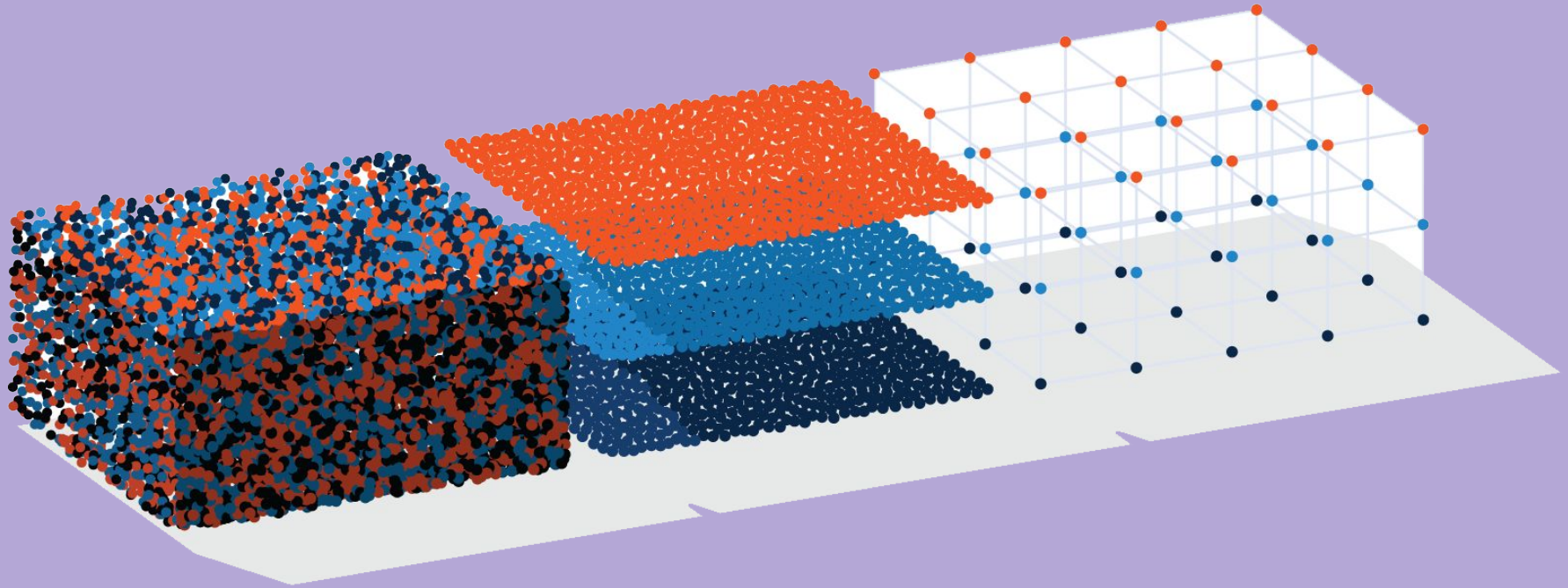
The OWASP Top 10 represents a consensus from the OWASP community of the most common and critical security risks to web applications.

OWASP Top 10

Created to educate software developers, designers, architects, managers, and organizations about the consequences of web application security weaknesses.

OWASP Top 10

Developed through industry surveys completed by over 500 individuals from hundreds of organizations and over 100,000 real-world applications and APIs.



This Week's Career Context

We will cover many web application vulnerabilities and demonstrate how to exploit them. Familiarity with these vulnerabilities and the methods used to exploit them are important skills for several cybersecurity careers:

Web Application Developers	Application Security Engineers	Penetration Testers
While web application developers might not work directly in cybersecurity, understanding vulnerabilities and their impact can help them create secure code	Application security engineers work alongside developers to ensure that they develop secure code.	Penetration testers who test an organization's web applications need to understand these vulnerabilities, attack methods, and impacts to conduct a thorough penetration test.

This Week

The week will proceed as follows:

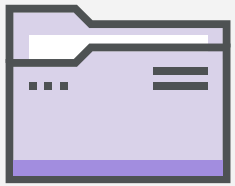
Day 1

We will begin today with the number one risk from the OWASP list—injection.

We will cover injection vulnerabilities such as SQL injection and cross-site scripting.

Day 2

We will cover web application vulnerabilities that exist within back-end components, such as directory traversal and file inclusion.



Day 3

We will cover broken authentication vulnerabilities and learn how to exploit these vulnerabilities with advanced tools such as Burp Suite.

BURPSUITE

How We Will Approach Each Vulnerability

As we introduce each web application vulnerability, we will cover the following:

- The intended purpose of the original function
- The vulnerability and method of exploit
- Unintended consequences of the exploit
- Mitigation of the vulnerability
- The potential impact of the exploit

Warning

The techniques we will learn throughout this unit can be used to cause serious damage to an organization's systems. This is ILLEGAL when done without permission.

Warning

All of the labs we provide are safe locations to test the methods and tools taught during the week.

Warning

NEVER apply any of these methods to any web applications you do not own or do not have clear, documented permission to be interacting with.

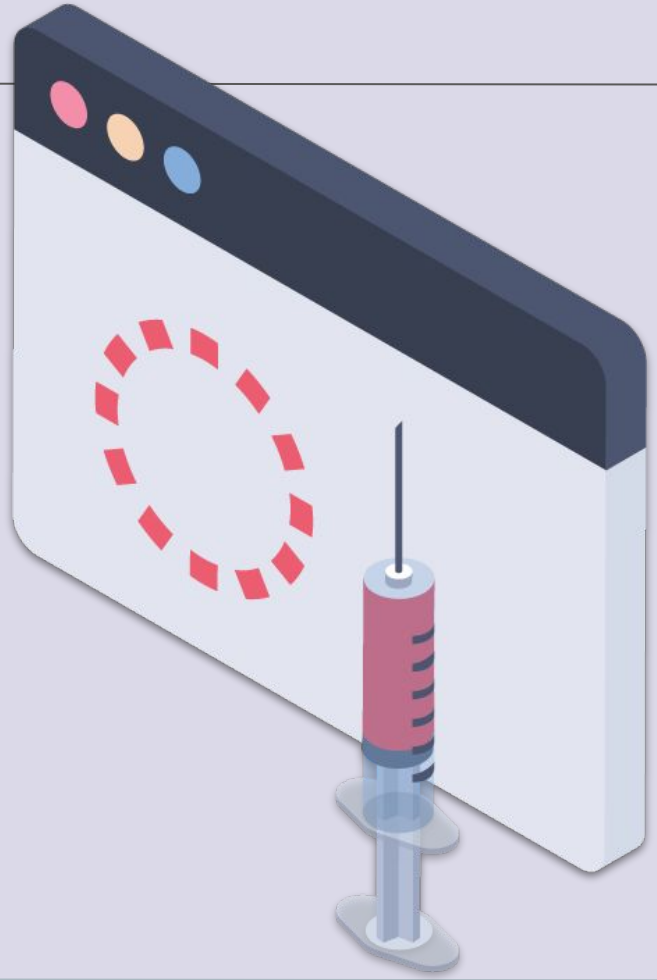
Injections

The OWASP Top 10: No.1 Injections

Injections

Injection attacks occur when an attacker supplies untrusted input to an application.

That malicious input, also known as a **payload**, contains malicious data or code that is then processed as part of a query or command that alters the way a program is intended to function.



Injections commonly occur in fields and forms on web applications, where malicious code can be injected.

Intended Purpose of Application

Search Hotel

What are you searching for? SEARCH

Las Vegas

Intended Results

Search Hotel

Las Vegas SEARCH

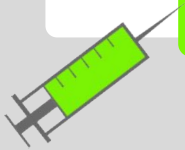
Las Vegas Convention Hotel:
2 Nights - 2 Adults $199 per night BOOK NOW

Applying injection to cause unintended results

Search Hotel

... SEARCH

Malicious Script



Unintended Results

Search Hotel

What are you searching for? SEARCH

Success: Command Executed -
Hotel Database has been deleted BOOK NOW

Types of Injection

Injection involve submitting an untrusted input. How applications work with input depends on their design, architecture, and functionality. We'll focus on:

Cross-Site Scripting

This injection is a submitted user input that can run malicious scripts against the website. It depends on the application modifying the client-side code with a user's input.

SQL Injection

In this injection type, a submitted user input can run SQL commands against a database. It depends on the application running queries against a SQL database.

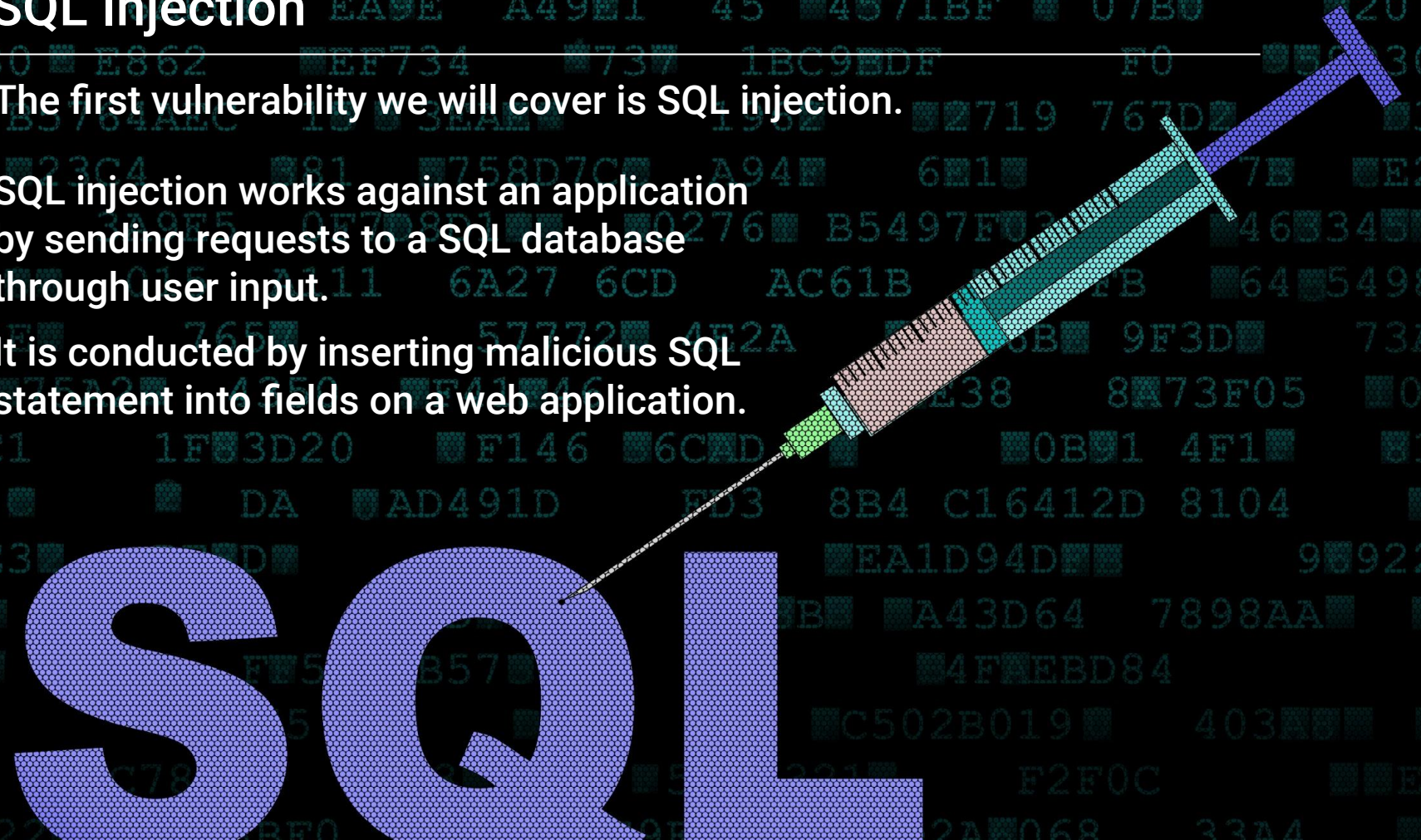
SQL Refresher and Unintended SQL Queries

SQL Injection

The first vulnerability we will cover is SQL injection.

SQL injection works against an application by sending requests to a SQL database through user input.

It is conducted by inserting malicious SQL statement into fields on a web application.



SQL Injections

Before we learn how to conduct a SQL injection attack, we need to understand what happens behind the scenes as an application interacts with a database. We will do this as follows:

01 Cover how a web application connects to a database.

02 Review the structure of a SQL database and a SQL query.

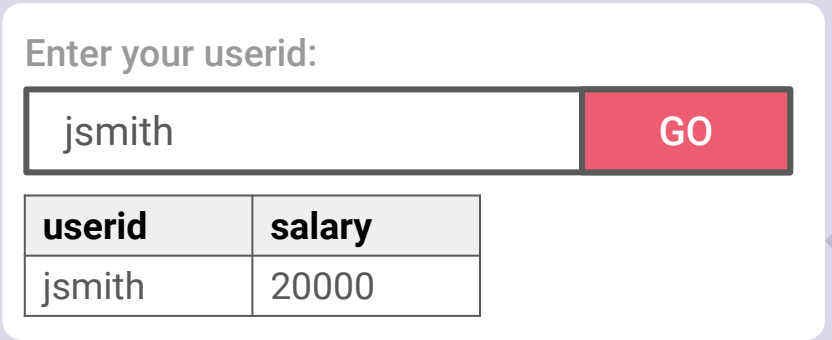
03 Demonstrate an intended query and the intended results.

04 Cover modification of the intended query to create unintended results.

How a Web Application Connects to a Database

The web application contains a field where a user can input their **user id**.

Julie enters her userid, "jsmith", and selects Go to submit.



The application returns Julie's salary of $20,000.

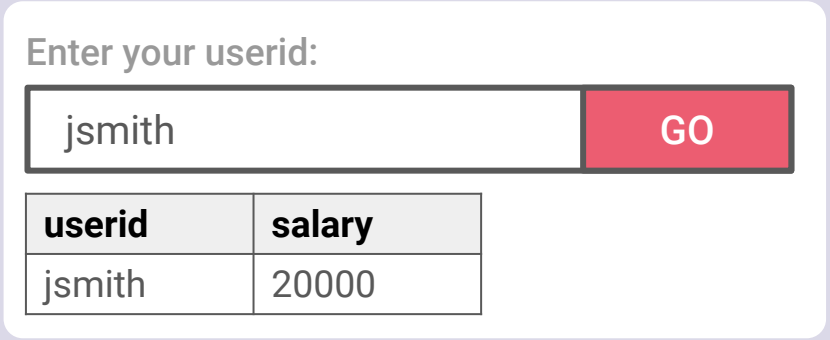
How a Web Application Connects to a Database

The web application took Julie's input ("jsmith") and sent it back to the web server.

The web server took the value of "jsmith" and submitted it to a database in a **pre-built** query to find that record.

The database contained a salary table with a list of users and salaries.

The query found the record for "jsmith" and returned the salary "20000" to Julie.



userid	salary
jsmith	20000

userid	salary
lsmith	45000
wgoat	1000000
rjones	777777
manderson	65000
jsmith	**20000**

Structure of a SQL Database and Query

SQL databases organize data like a spreadsheet.

Each **row** or **record** is an item in the database.

userid	salary
lsmith	45000
wgoat	1000000
rjones	777777
manderson	65000
jsmith	20000

A whole spreadsheet of rows and columns is called a **table**.

A collection of tables is a **database**.

Each **column** is a piece of data in the row.

Structure of a SQL Database and Query

`Userid, salary` indicates the two fields requested to be displayed.

`where userid = 'jsmith'` instructs the program to only return records where the userid is “jsmith”.

```
select userid, salary from salaries where userid = 'jsmith'
```

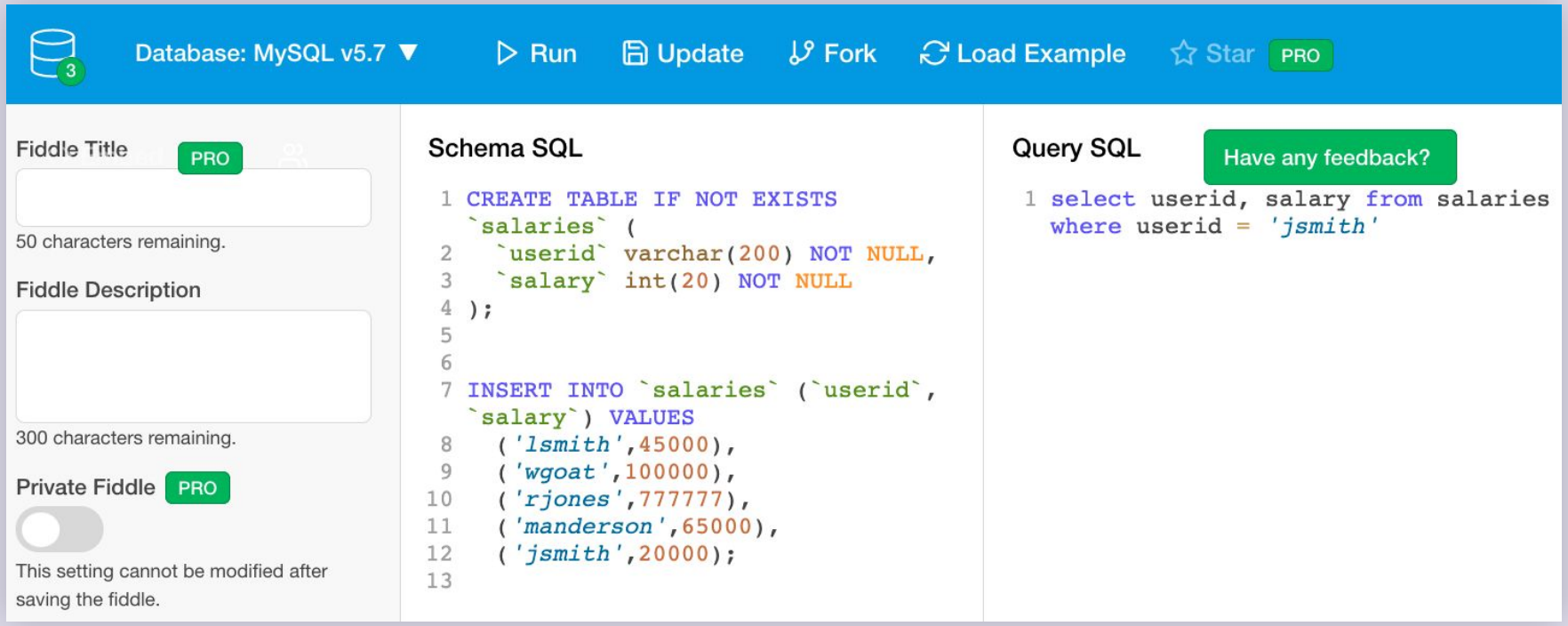
`select` is used to read or select data from a table.

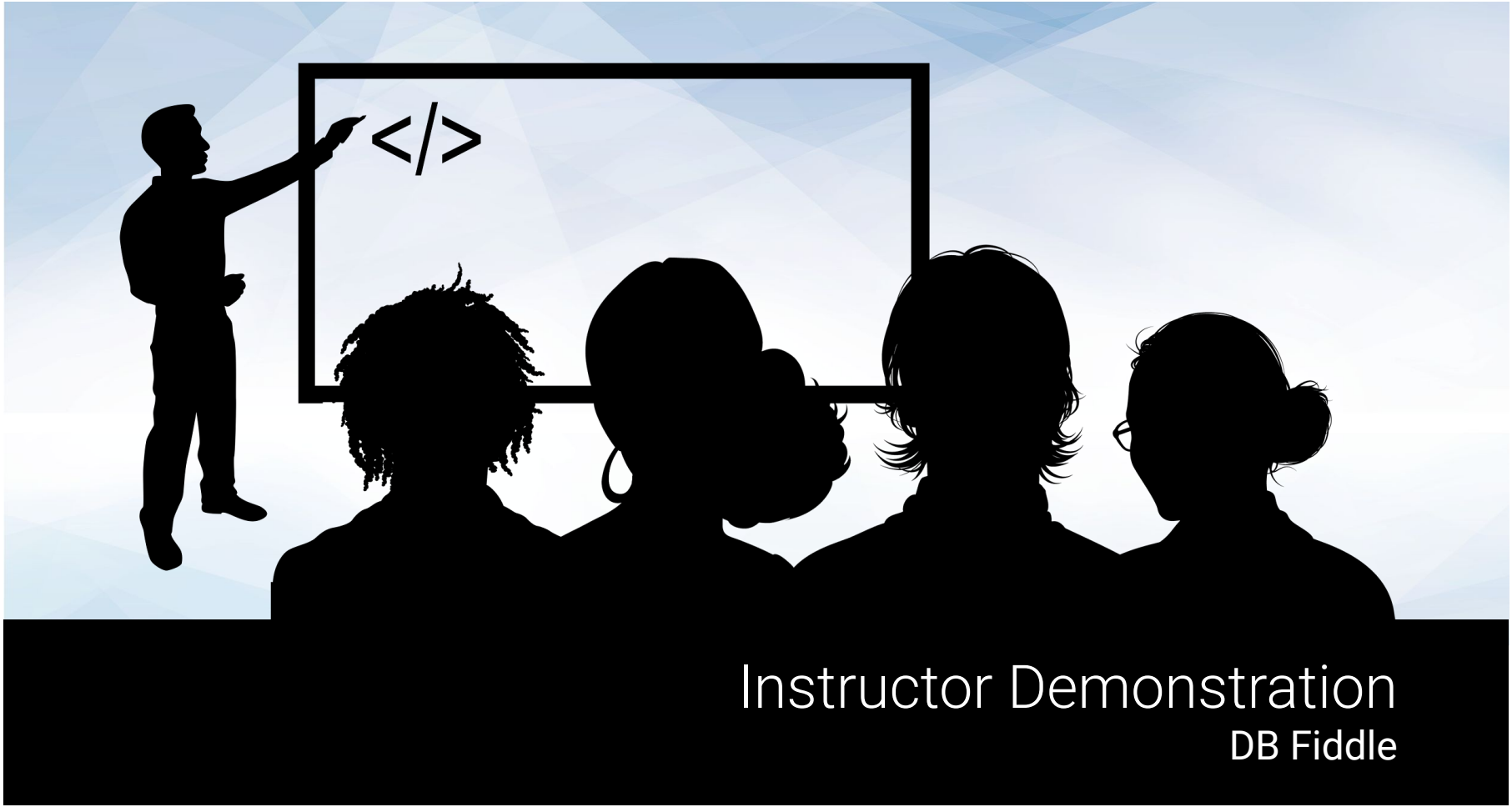
`from salaries` instructs the program to select the data from the salaries table.

userid	salary
lsmith	45000
wgoat	1000000
rjones	777777
manderson	65000
jsmith	20000

Intended Query and Results Demonstration

In the next demonstration, we will review and run an intended query against a database to visualize how they return intended results.





Instructor Demonstration

DB Fiddle

Modification of Intended Query to Create Unintended Results

We know the intended pre-built query for the web application:

```
select userid, salary from salaries where userid = ' '
```

After Julie entered her user id of "jsmith", the intended query to pull the salary was updated:

```
select userid, salary from salaries where userid = 'jsmith'
```

Modification of Intended Query to Create Unintended Results

SQL query uses the conditional `where` clause, like `where userid = 'jsmith'`, to specify the desired results of a query.

	userid	salary	T/F
`Does userid = jsmith?`	lsmith	45000	**False**
`Does userid = jsmith?`	wgoat	1000000	**False**
`Does userid = jsmith?`	rjones	777777	**False**
`Does userid = jsmith?`	manderson	65000	**False**
`Does userid = jsmith?`	jsmith	20000	**True**

`userid = jsmith`, so return salary.

Modification of Intended Query to Create Unintended Results

We can add a second condition, called an **always true statement**, to trigger true statements for the other columns.

	userid	salary	T/F
`Does 1 = 1?`	lsmith	45000	True
`Does 1 = 1?`	wgoat	1000000	True
`Does 1 = 1?`	rjones	777777	True
`Does 1 = 1?`	manderson	65000	True
`Does 1 = 1?`	jsmith	20000	True

Whereas `where userid = jsmith`, only applies to the `jsmith` row, an always true statement like `1=1` applies to all rows.

Therefore, the query returns every user id.

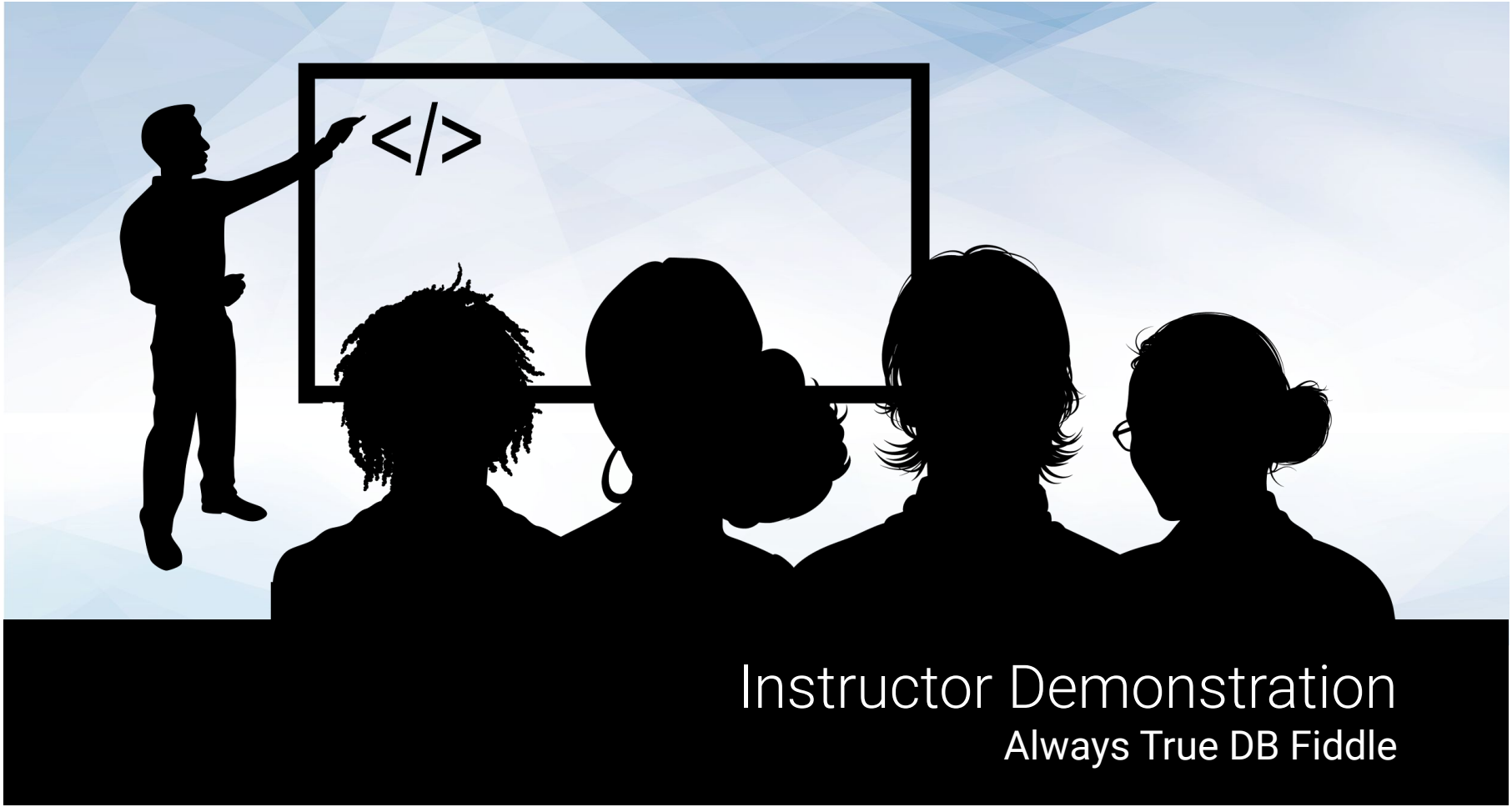
Modification of Intended Query to Create Unintended Results

We can add a second condition, called an **always true statement**, to trigger true statements for the other columns.

```
select userid, salary from salaries where userid = 'jsmith' OR '1' = '1'
```

Malicious payload

Let’s take examine always true statements in DB Fiddle.



Instructor Demonstration

Always True DB Fiddle

Review

The concepts covered so far...

01 Web vulnerabilities are weaknesses that exist within the intended functions of web applications.

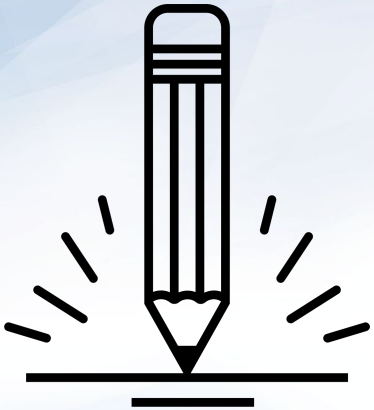
02 The OWASP Top 10 is a published list of the top 10 most common and critical security risks to web applications.

03 **Injections** supply untrusted input, or the **payload**, to an application that is processed as part of a query or command.

04 **SQL injection** depends on web applications that apply user input to a database.

05 A web server can take the user input and apply it to a database using SQL queries.

06 A method to modify an intended SQL query is to add an **always true statement**.



Activity: SQL Refresher and Unintended SQL Queries

In this activity, you will design several SQL queries to test directly against a database, which is represented by DB Fiddle.

Suggested Time:
20 Mins

Time’s Up! Let’s Review.

Testing SQL Injection on Web Applications

Testing SQL Injection

Now we will focus on the web application and conduct the following steps:

01

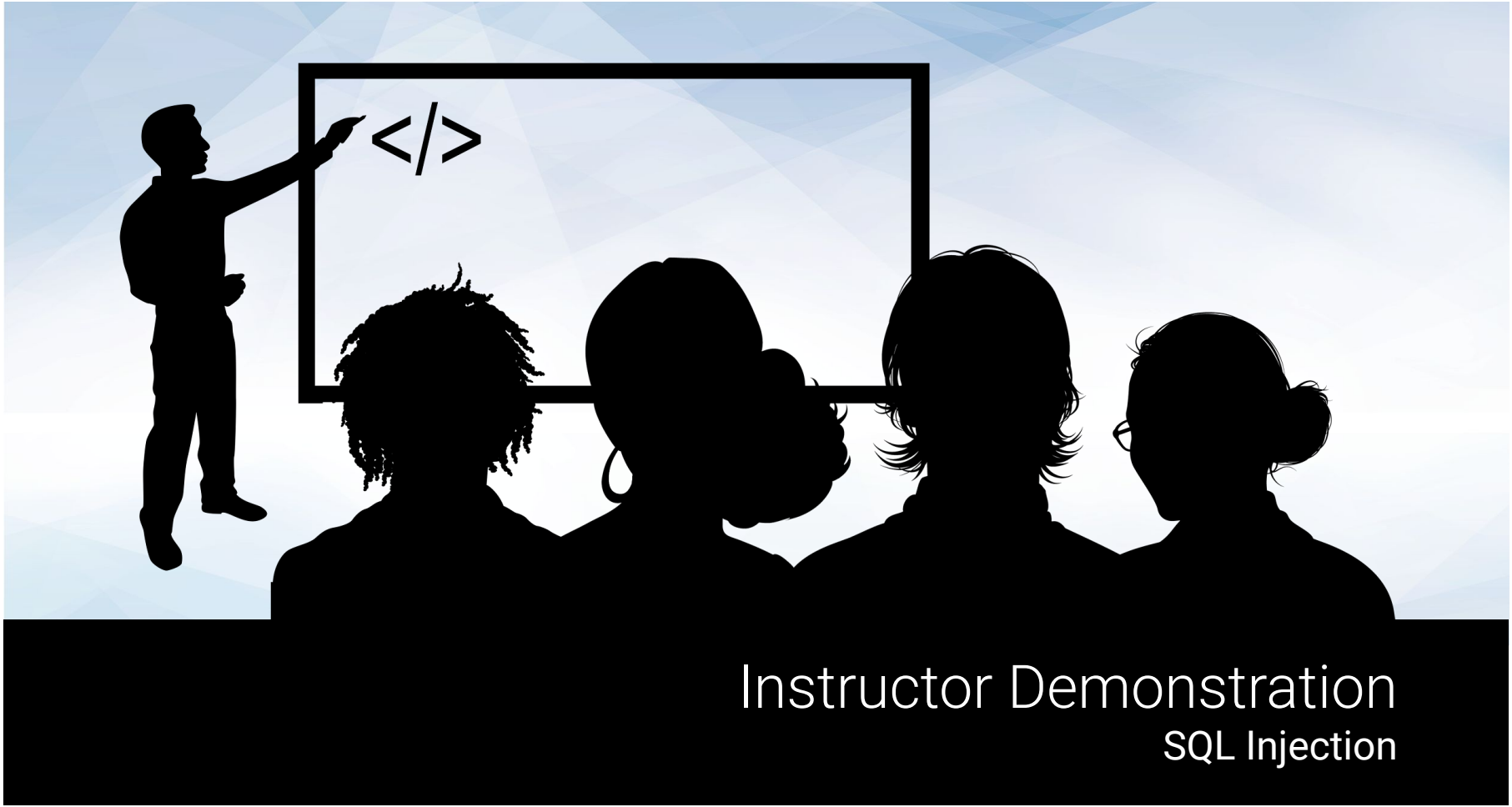
Test the intended purpose of the application.

02

Design malicious payloads by using the SQL queries built in the previous activity.

03

Test SQL injection on the application by using those payloads.



Instructor Demonstration

SQL Injection

Real-World Challenges

While the purpose of the lesson was to illustrate how a SQL injection payload can create unintended results, in the real world we need to be aware of several issues not covered in this lesson.

Access to Database Structure

In the real world, application security analysts would not have access to the database tables or structure.

Variety of Databases

In the real world there are multiple database types, in which SQL queries will not work the same way.

Variety of Attacks

Other SQL injection attacks can delete or alter data, rather than just display it as we did.

Impact

The reason that SQL injection is considered one of the most harmful attacks is due to the potential impact.

If a malicious actor successfully launches a SQL injection attack against an organization's database, they might be able to display, change, or delete the organization's confidential data.


COMPUTERWORLD UNITED STATES INSIDER

SONY CYBERATTACK

Sony Pictures falls victim to major data breach

Hacking group LulzSec claims it has accessed personal data on more than 1 million people

By Jaikumar Vijayan

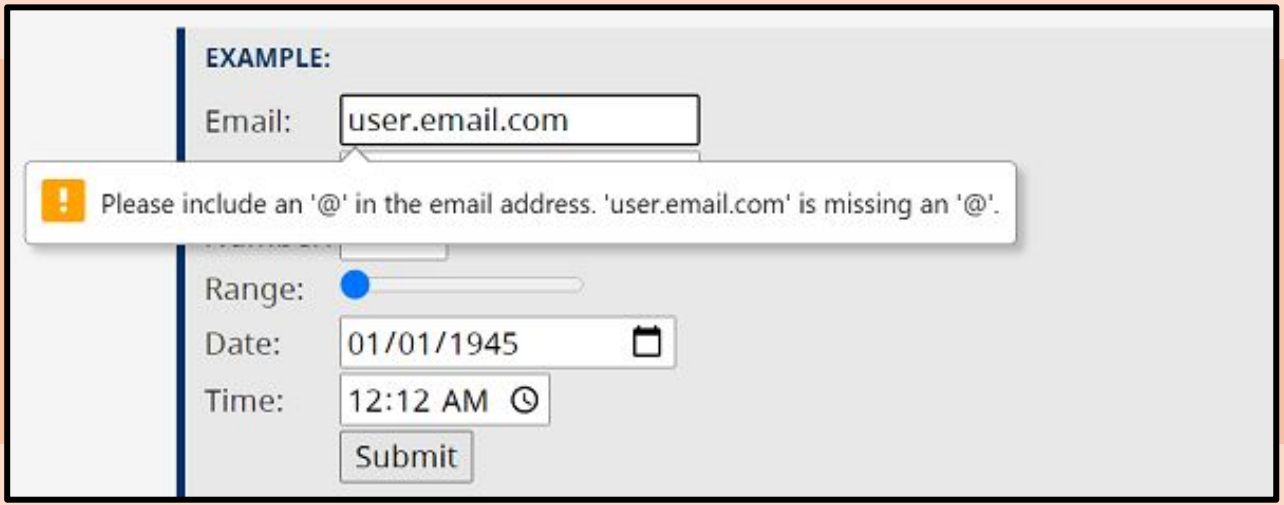
Computerworld | JUN 2, 2011 7:20 PM PST

https://www.computerworld.com

Mitigation Methods

Input validation is a common method used to mitigate this attack.

Input validation is a method to validate the data input with a predefined logic, ensuring that the input is what the application is expecting.



Mitigation Methods

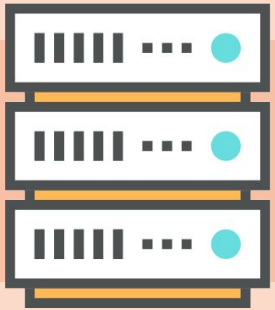
Input validation can be applied on the client side or the server side

Client-side

Client-side input validation involves coding the predefined logic into the webpage.

For example

An input can only be chosen from a predefined drop-down menu.

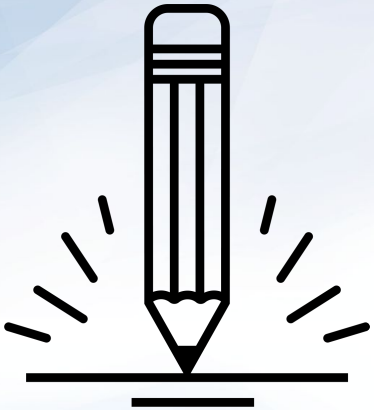


Server-side

Server-side input validation involves adding the predefined logic into the code on the web server.

For example

If a user enters a malicious SQL code and selects submit, then the web server will check and remove it after receiving this malicious input.



Activity: Testing SQL Injection on Web Applications

In this activity you will use the SQL queries you crafted last activity to create payloads.

Suggested Time:
20 Mins

Time’s Up! Let’s Review.

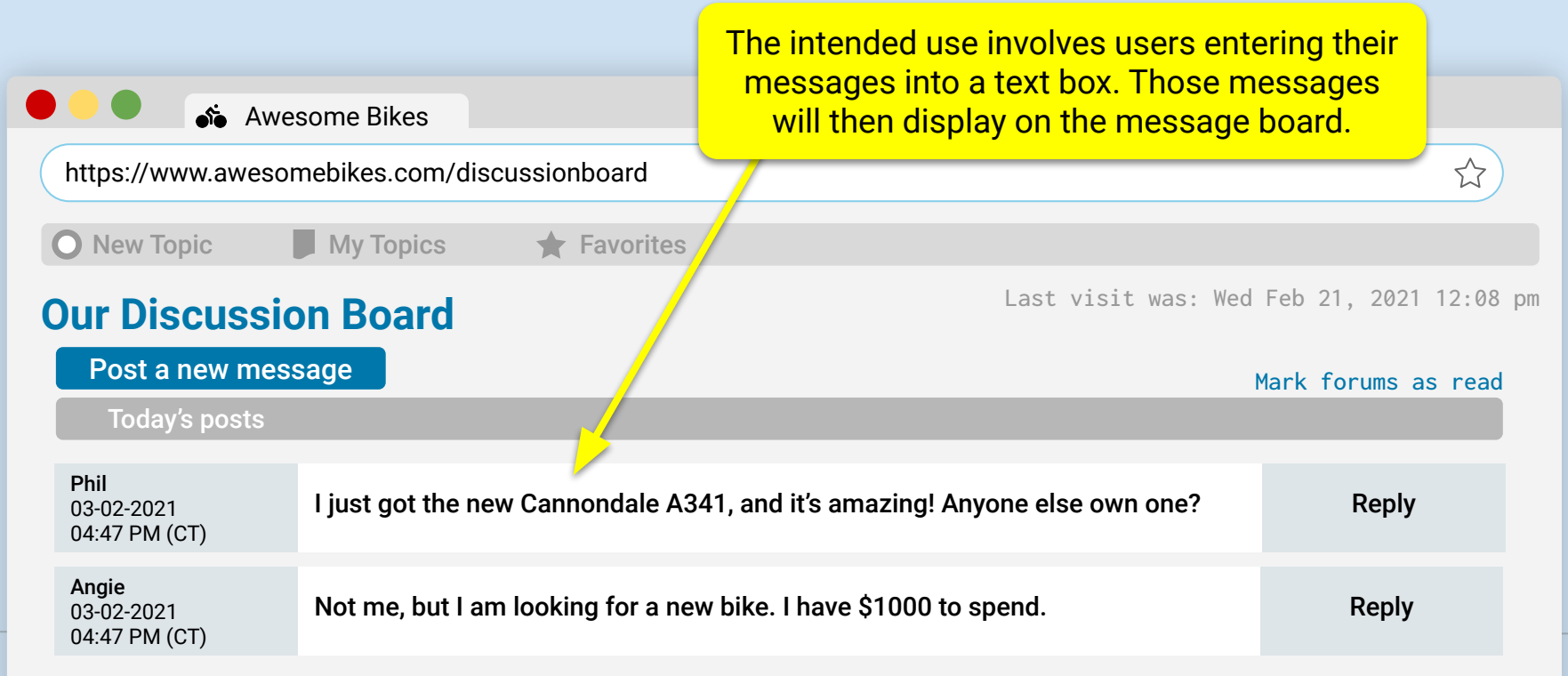

Break

HTML & JavaScript

Cross-site scripting (XSS) is another injection type that occurs when an application takes in malicious user input that modifies the source code of the application.

XSS Scenario

AwesomeBikes.com is an online message board intended for users to recommend bike models, ask questions, and chat about everything related to cycling.



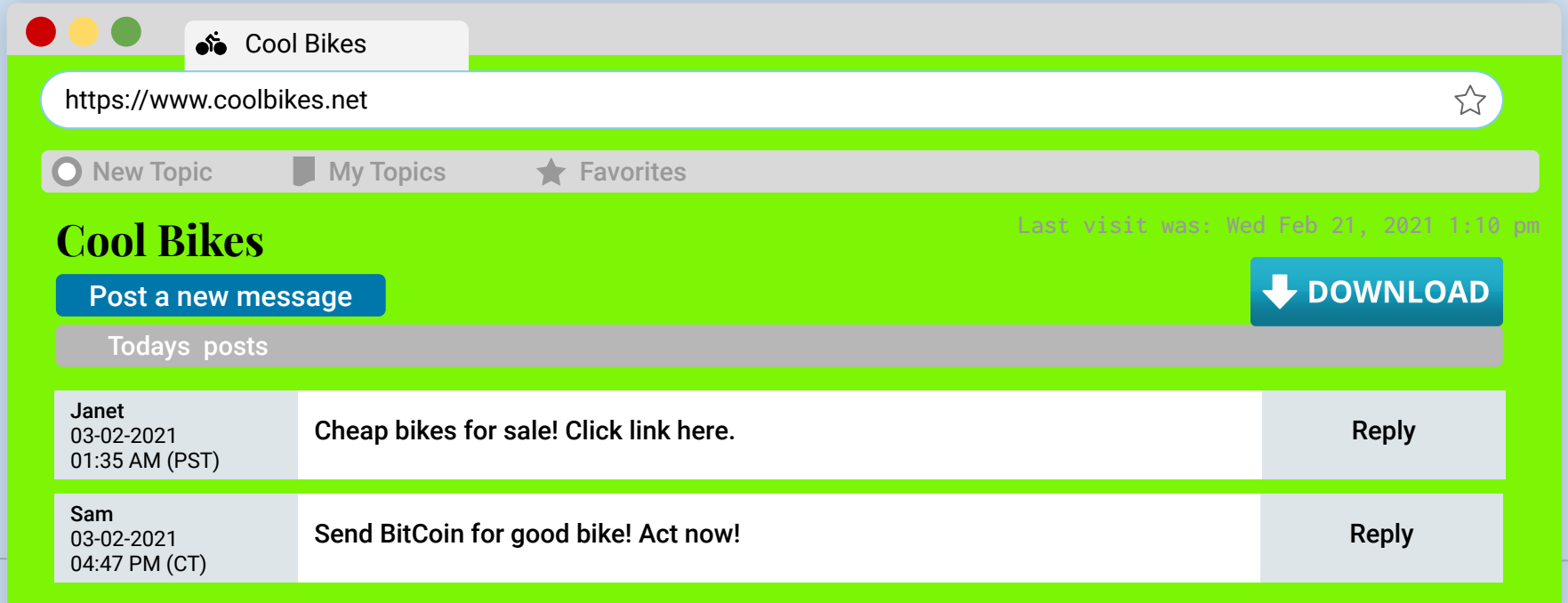
XSS Scenario

However, because this site is vulnerable to injection attacks, a malicious user can input a script in the text box.

Rather than post the text of the script to the message board, the webpage will interpret the script as code. This code will infect any user who subsequently visits AwesomeBikes.com.

XSS Scenario

Once the attacker posts the redirect script, subsequent visitors to AwersomeBikes.com will be redirected to coolbikes.net, a malicious spoof site that tries to scam customers.



XSS Impacts

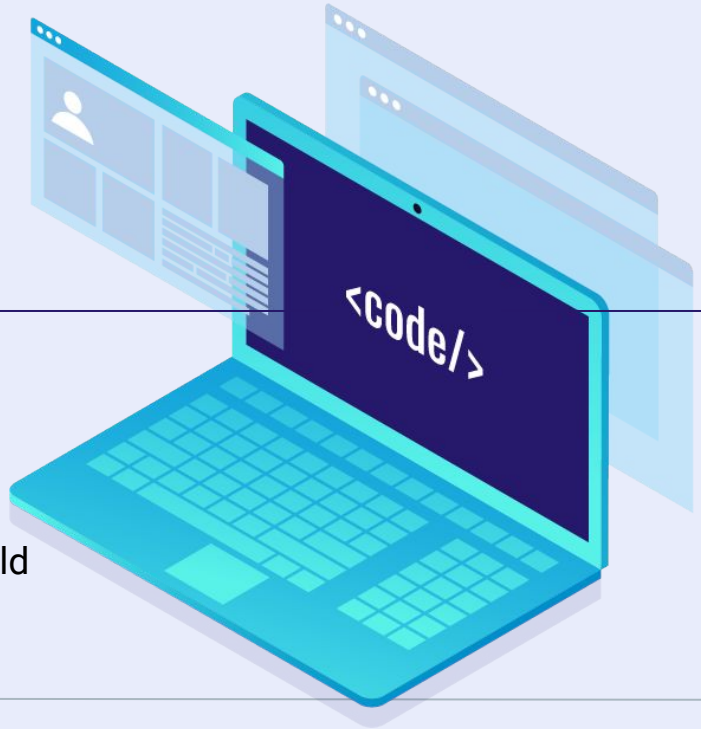
Depending on the specific script that the user inputs, the impact on subsequent visitors can include a number of potential actions, such as the following:

- Redirecting to a spoof page where the malicious user can then try to sell fake products and capture credentials.
- Stealing the user's cookies.
- Adding a keylogger onto the user's machine.
- Downloading malware to the user's machine.

HTML and JavaScript

To learn how XSS exploits affect webpages, we first need to go behind the scenes of a webpage and learn how the source code creates the displays that we interact with when visiting.

To better understand how the webpage source code works, we will cover the following:



HTML:
A language language used to build the structure of a webpage.

Javascript:
A language used to make webpages interactive and dynamic.

HTML

When a user enters a URL in a browser and the browser displays a webpage, several high level-steps occur behind the scenes.

For example, if you want to access Google.com

STEP 1

You enter "google.com" in the search bar on the top of your browser and press Enter.

STEP 2

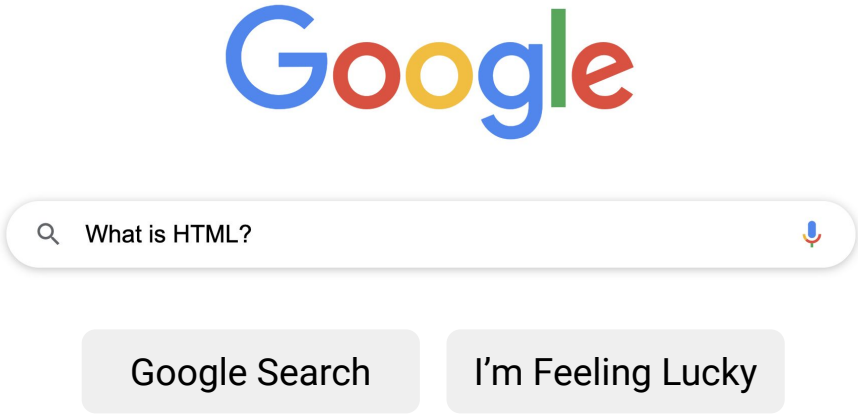
Your browser sends a request to Google's web server.

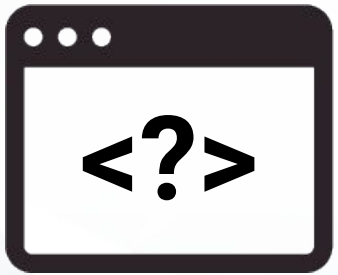
STEP 3

Google's web server returns an HTML file to your browser.

STEP 4

Your browser renders the HTML file to display the items on the webpage.





What Do You Know about HTML?

HTML

HTML, or Hypertext Markup Language, is a language used to display the content on a webpage.

It is a considered a **client-side language**, as it is designed to run on the user's client, the browser.

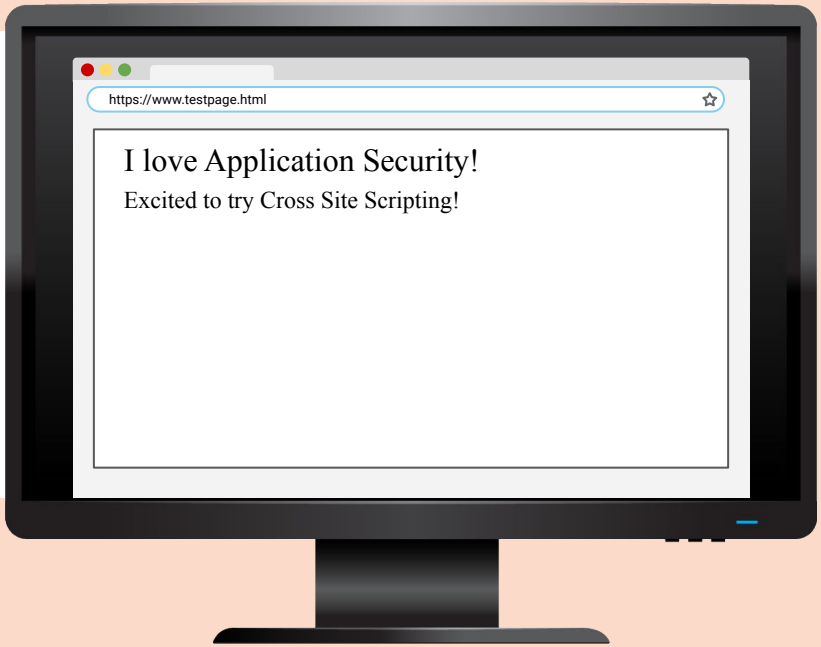
HTML contains **elements**, which can define the following:

- The boundaries of a paragraph on a webpage, like where a paragraph starts and ends.
- The size and boldness of headings on a webpage.
- The placement of embedded images, video, or audio.

HTML elements use **tags** and **angle brackets** `< >` to delineate the HTML structure of a webpage. For example, the `<html>`, `<body>`, and `<H1>` tags introduce content into a webpage.

Sample HTML

```
<html>
<body>
<h1>I love Application Security!</h1>
<h2>Excited to try Cross Site Scripting!</h2>
</body>
</html>
```



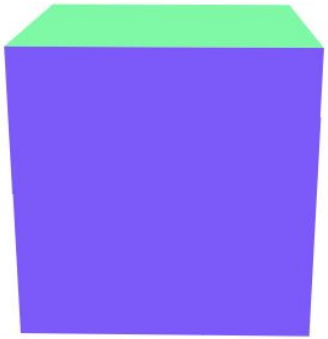
JavaScript

While HTML provides options that can improve the aesthetic and design of a web page, it still has limitations to static improvements and is unable to create more dynamic features on a web page.

JavaScript is a programming language that allows web developers to add complex web features that dynamically update web content and add life to a web page.

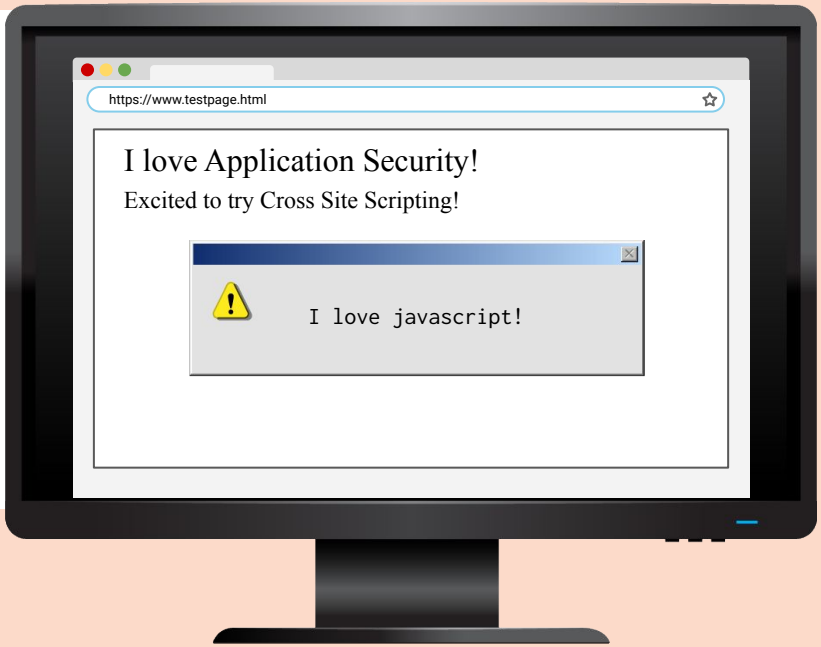
These features include:

- Animations
- Audio and Video playback
- Interactive games and maps
- Online Chats

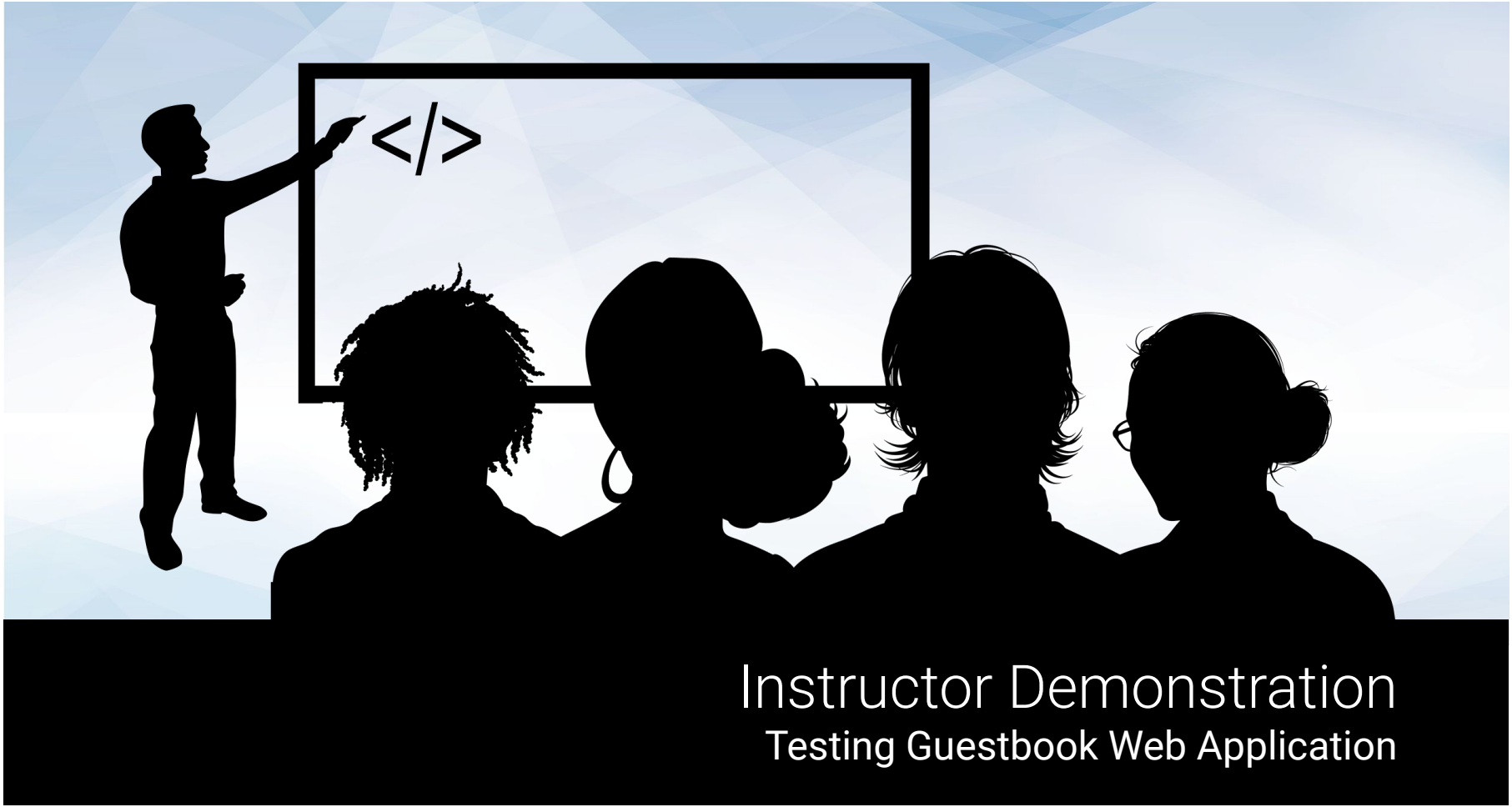


JavaScript

```
<html>
<body>
<h1>I love Application Security!</h1>
<h2>Excited to try Cross Site Scripting!</h2>
<script>alert("I love javascript!")</script>
</body>
</html>
```



- `<script>` indicates the start of the JavaScript.
- `alert("I love javascript")` is a JavaScript alert script used to create a pop-up that states "I love javascript".
- `</script>` closes the script JavaScript.



Instructor Demonstration

Testing Guestbook Web Application

Testing XSS on Web Applications

The OWASP Top 10: No.7 Cross-Site Scripting

While cross-site scripting is technically an injection risk, it is prevalent enough to have its own risk category on the OWASP Top 10 list.

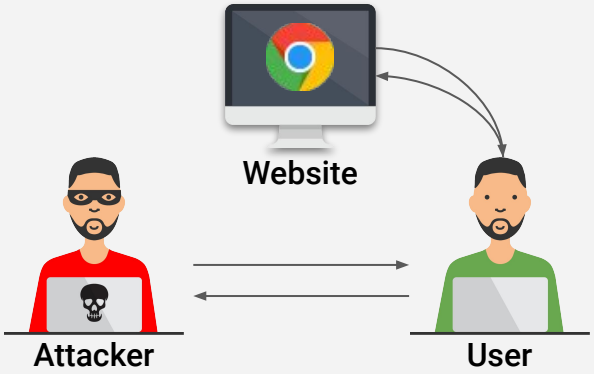
Cross-Site Scripting (XSS) is a web application attack that injects malicious scripts into vulnerable web applications, impacting the users of the web application.

Testing XSS on Web Applications

There are multiple types of cross-site scripting, depending on how the application is designed:

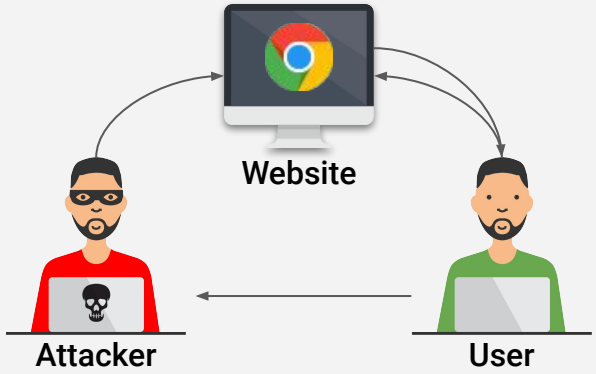
Reflected (Non-Persistent) XSS

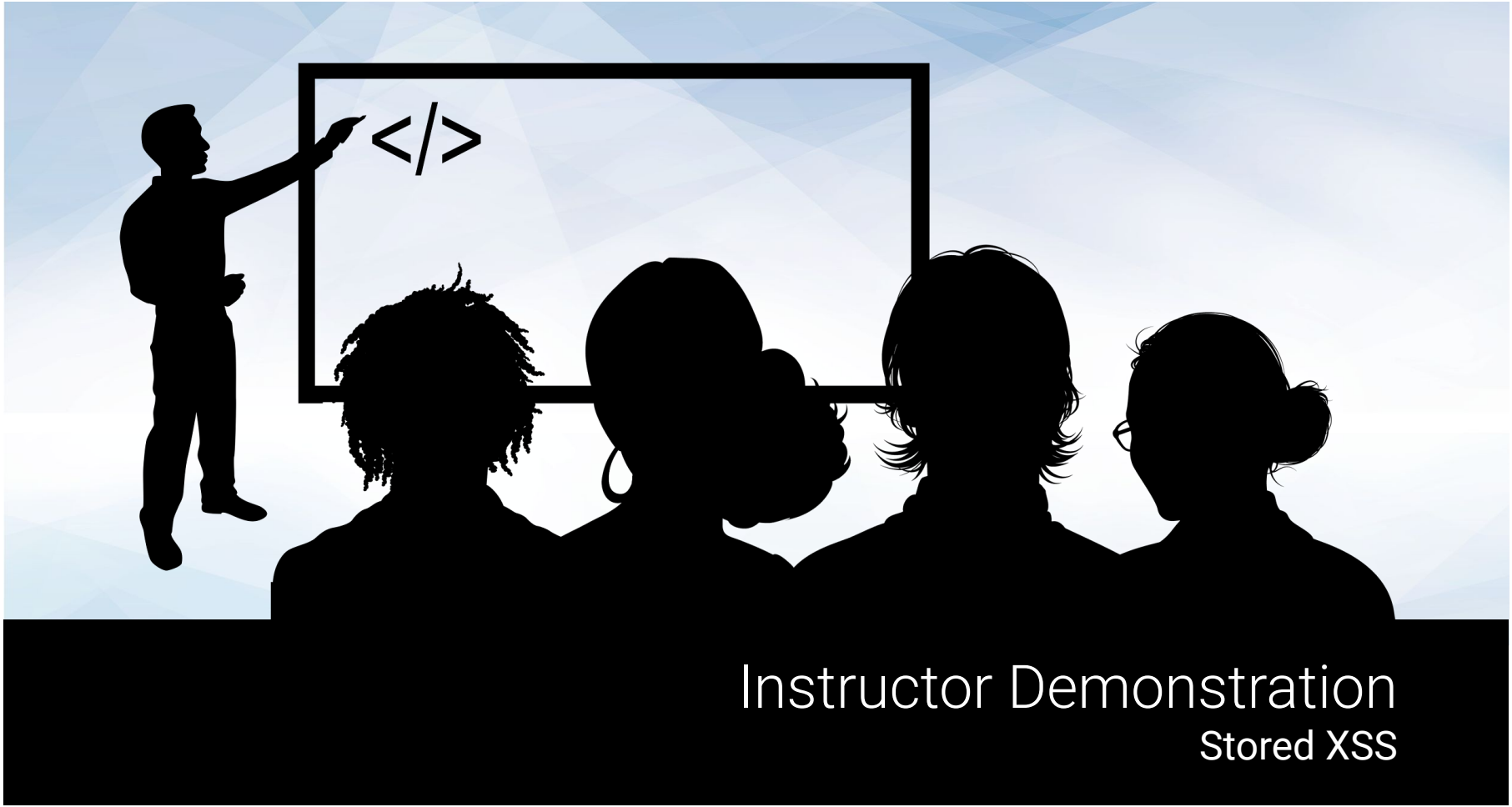
depends on the user input being immediately returned to the user and not stored on the application's server.



Stored (Persistent) XSS

depends on the user input being stored on the application's server and later retrieved by a victim accessing the web application.





Instructor Demonstration

Stored XSS

Stored XSS Recap

Stored XSS depends on the application storing the user input on the web application's server.

An attacker submits a malicious input (or payload) into the web application.

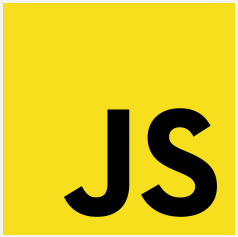
The malicious input is returned to ANY subsequent user that visits the infected page on the web application.

Stored XSS Recap

Stored XSS depends on the application storing the user input on the web application's server.

01

In the example, we added a JavaScript tag to the HTML.



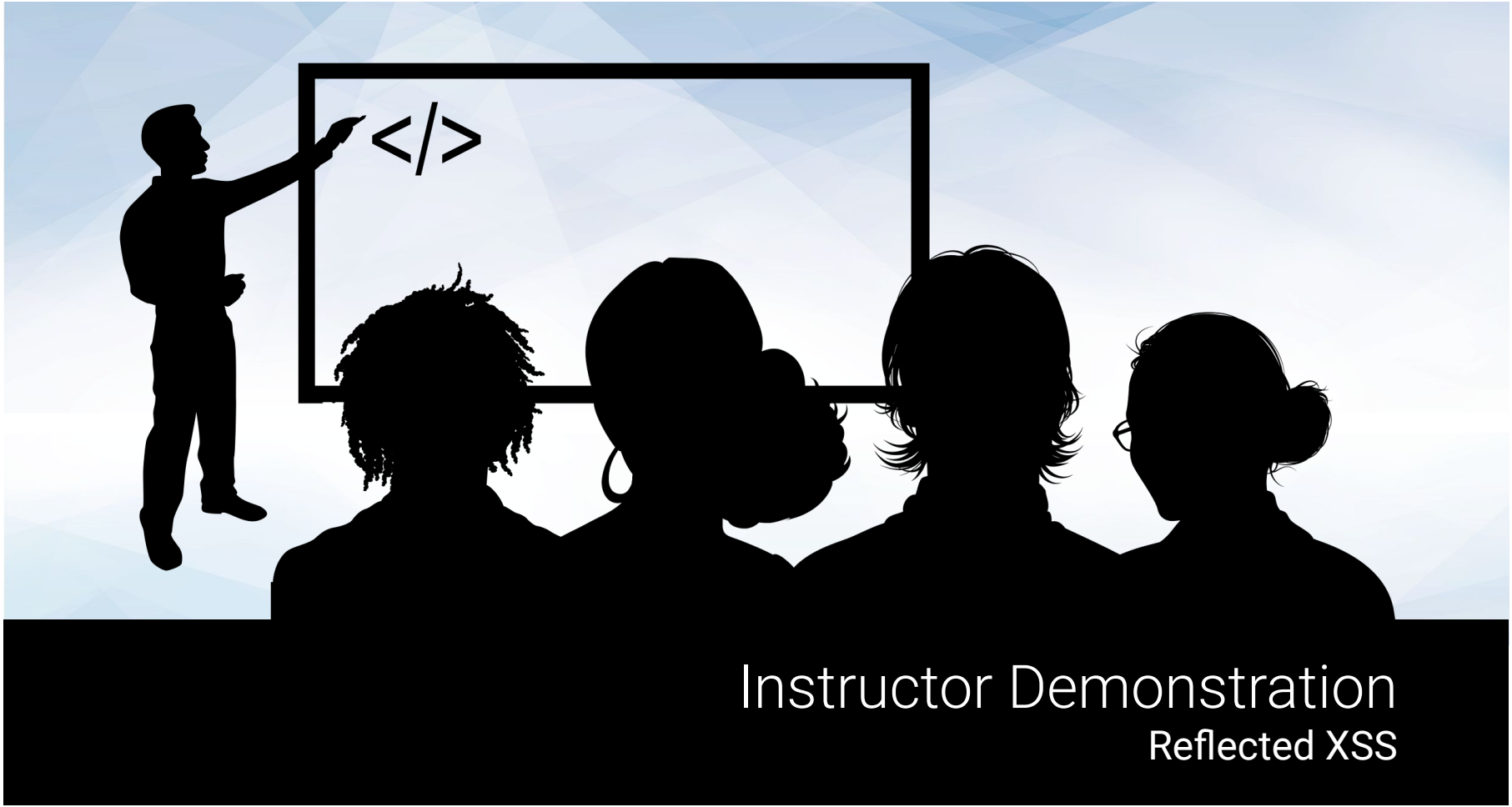
02

Anytime a user visits the page, a pop-up will be displayed.

03

An attacker could apply much more malicious scripts, such as the following:

- Redirecting the user to a spoof webpage.
- Installing a keylogger to capture what the user enters on other webpages.
- Capturing the user's cookies and sending them to the attacker.



Instructor Demonstration

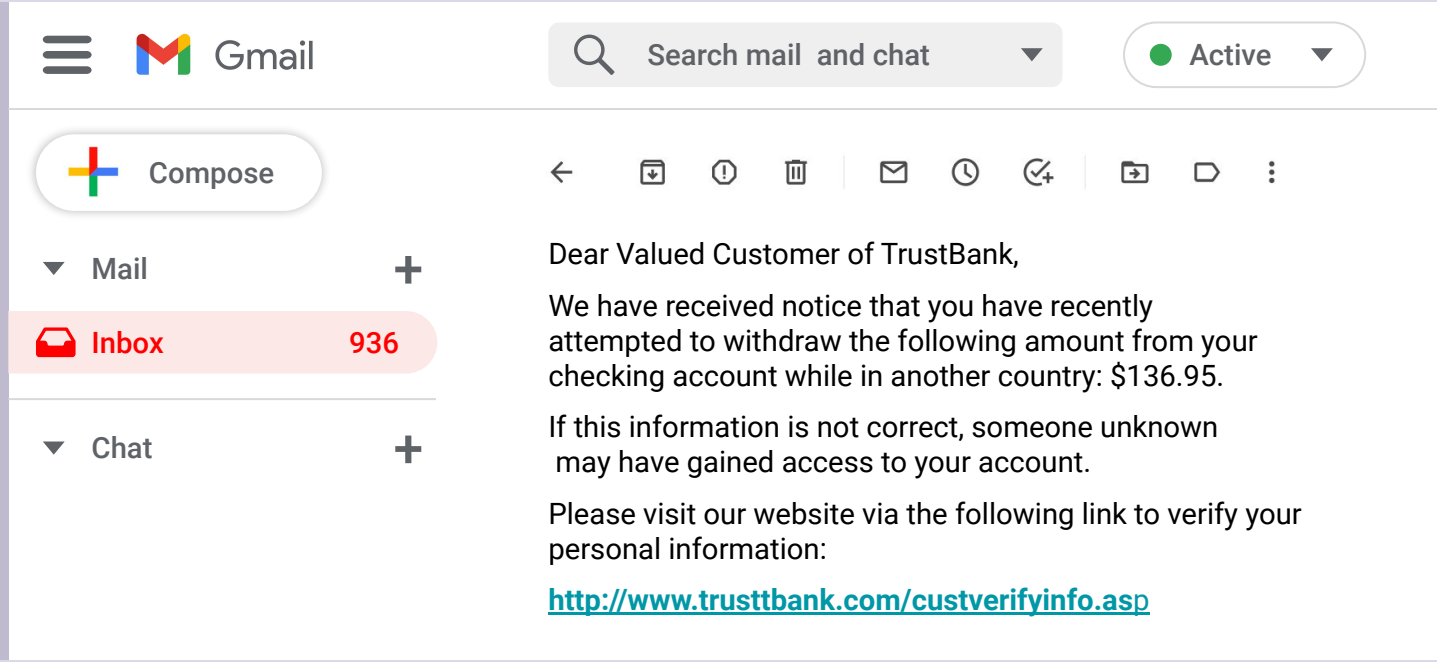
Reflected XSS

Delivering a Reflected XSS Attack

Delivering a stored XSS attack simply requires inputting malicious scripts into the application. Then anyone who visits that application will have that script run.

Reflected XSS scripts need to be delivered differently, because the script is not stored on the web application. The URL must be sent to the victim, and the victim must be deceived into clicking the link.

Phishing attacks are a common delivery method.



Impact

Cross-site scripting is considered a harmful attack due to the serious potential impact.

01 Stored XSS

If a malicious actor applies **stored XSS** to a web application, every subsequent user that visits the infected web application will have the malicious script run, with the following impact:

- The victim's cookies stolen and session hijacked.
- The victim's computer infected with malware.

02 Reflected XSS

If a malicious actor applies **reflected XSS** to a web application by sending a phishing email to a victims, the attacker can also do the following:

- Steal the victim's cookies and hijack their session.
- Infect the victim's computer with malware.

Mitigation Methods

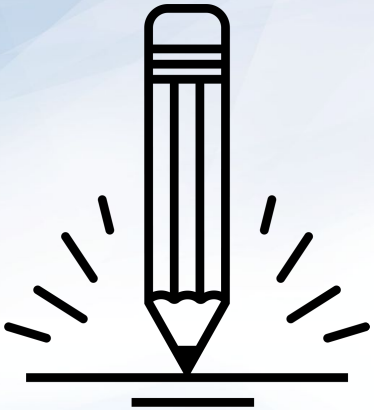
Input validation is a common method used to mitigate cross-site scripting.

Input validation is a method used to validate the data input with a pre-defined logic to ensure that the input is what the application is expecting.

For stored XSS, we can use **server-side input validation** to prevent scripts from being stored on the application's web server.

For example, it would not accept, at any time, submitted input that might includes `<script>` or `</script>`.

For reflected XSS, the input is not stored on the web server, so it would be best to use a **client-side input validation**, in which the code might not allow a malicious user to input scripts.



Activity: Testing XSS on Web Applications

In this activity you will test the web application for cross-site scripting vulnerabilities.

Suggested Time:
20 Mins

Time’s Up! Let’s Review.

Class Objectives

- Articulate the intended and unintended functionalities of a web application.
- Identify and differentiate between SQL and XSS injection vulnerabilities.
- Design malicious SQL queries using DB Fiddle.
- Create payloads from the malicious SQL queries to test for SQL injection against a web application.
- Design malicious payloads to test for stored and reflected cross-site scripting vulnerabilities.

79