Back-End Component Vulnerabilities

Cybersecurity

Web Vulnerabilities and Hardening, Day 2

Class Objectives

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

Differentiate between front-end and back-end component vulnerabilities.

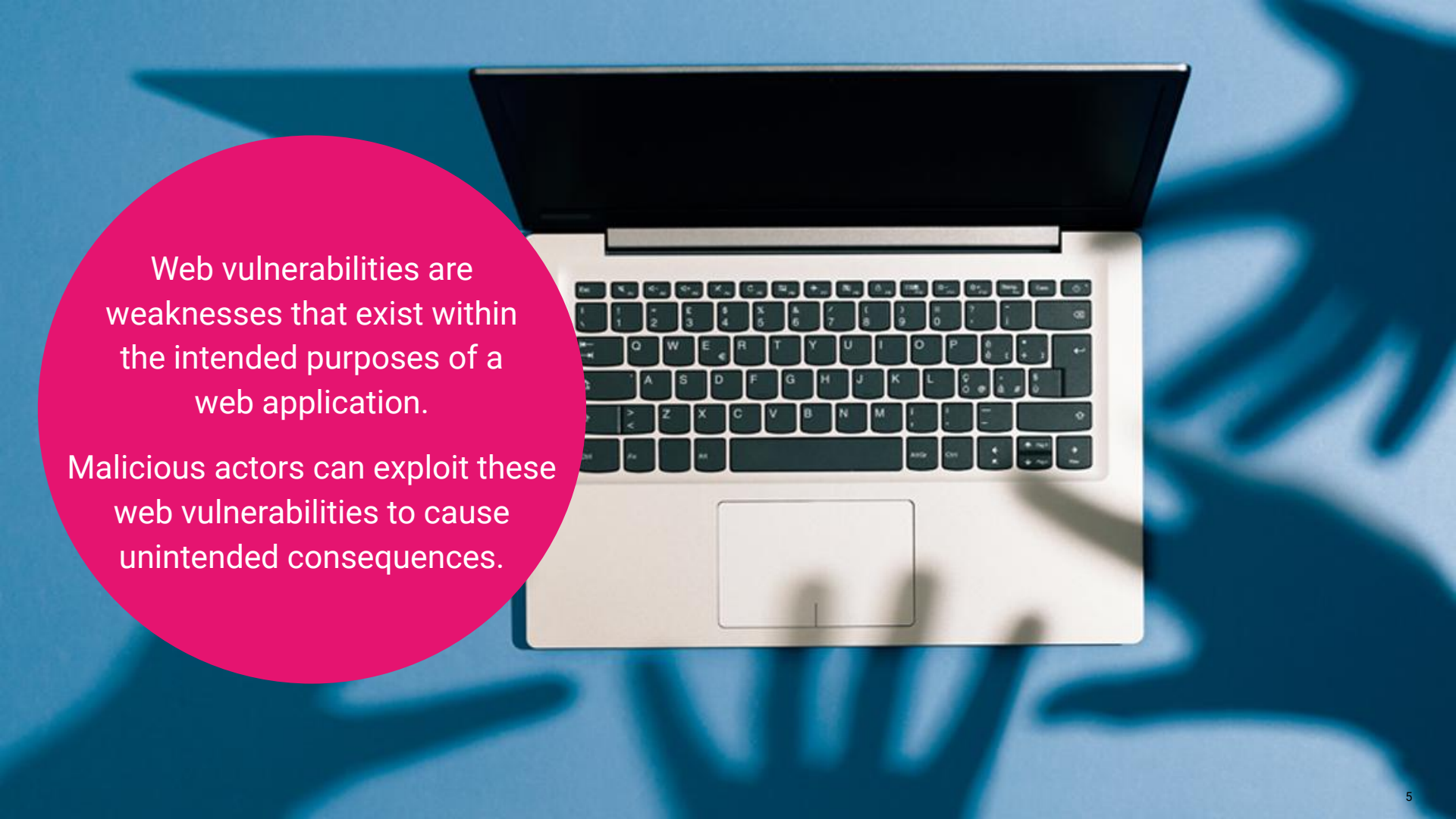
View confidential files with a directory traversal attack by using the dot-slash method.

Exploit a web application's file upload functionality to conduct a local file inclusion attack.

Modify a web application's URL to use a malicious remote script to conduct three different remote file inclusion attacks.

Intro to Back-End Component Vulnerabilities


Recap



Web vulnerabilities are weaknesses that exist within the intended purposes of a web application.

Malicious actors can exploit these web vulnerabilities to cause unintended consequences.

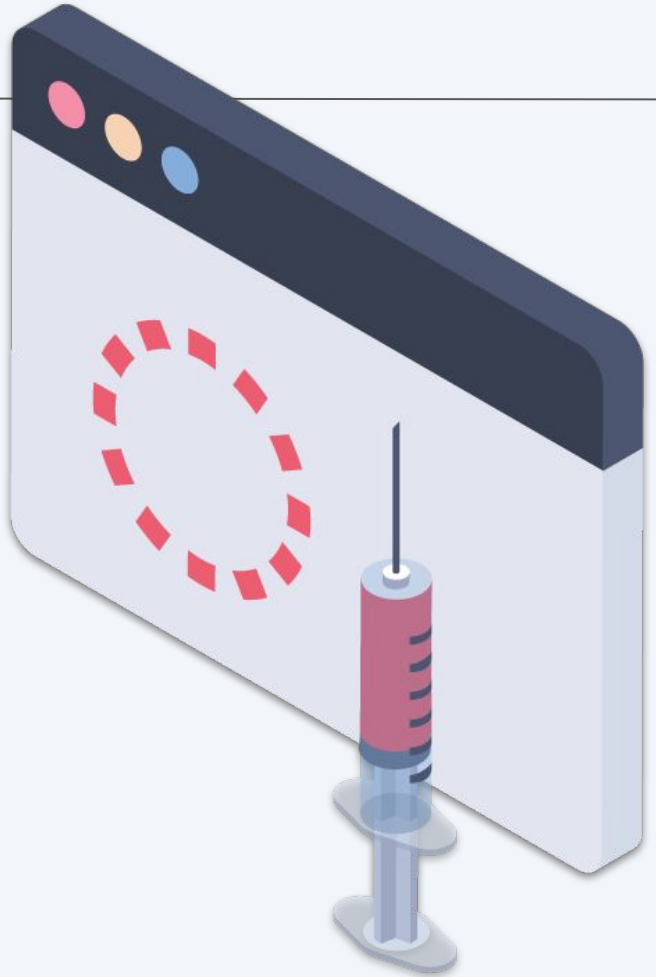
The **OWASP Top 10** is a list used by security professionals and web developers that represents a broad consensus of the most critical web application risks. The number three risk on the OWASP Top 10 list is **injection**.

A01	Broken Access Control
A02	Cryptographic Failures
A03	**Injection**
A04	Insecure Design
A05	Security Misconfiguration
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

Recap: Injections

Injection is a specific type of attack that occurs when an attacker supplies untrusted input to an application.

The **payload** is processed as part of a query or command that alters the intended functionality of a program.

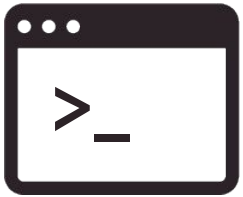


Review: Injection Types

There are various types of injection, depending on the architecture and functionality of the application.

Cross-Site Scripting (XSS)

- Conducted by an attacker inserting malicious JavaScript into a web application.
- Depends on the application modifying the client-side code with a user's input.



SQL Injection

- Conducted by an attacker inserting malicious SQL statements into a web application.
- Depends on the application running queries against a SQL database.

Review: Cross-Site Scripting (XSS)

The two types of XSS include:

01 Reflected

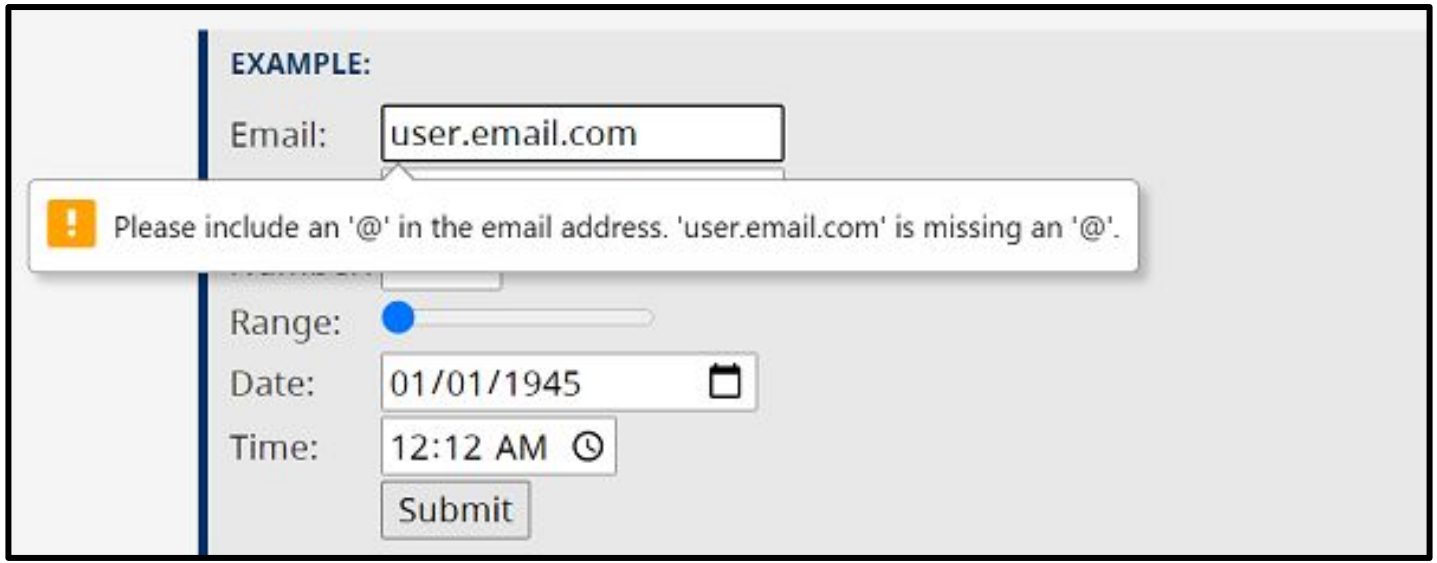
Depends on the user input being immediately returned to the user and not stored on the application's server.

02 Stored

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

Review: Input Validation

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



Back-End Component Vulnerabilities

Last Class

In the last class, we covered **JavaScript** and **HTML** cross-site scripting vulnerabilities.

These languages are considered **front-end components**, as they form the part of the web application that the user interacts with, typically with a browser.

Another front-end component is **Cascading Style Sheets (CSS)**, which formats the layout of the HTML.

Today's Class

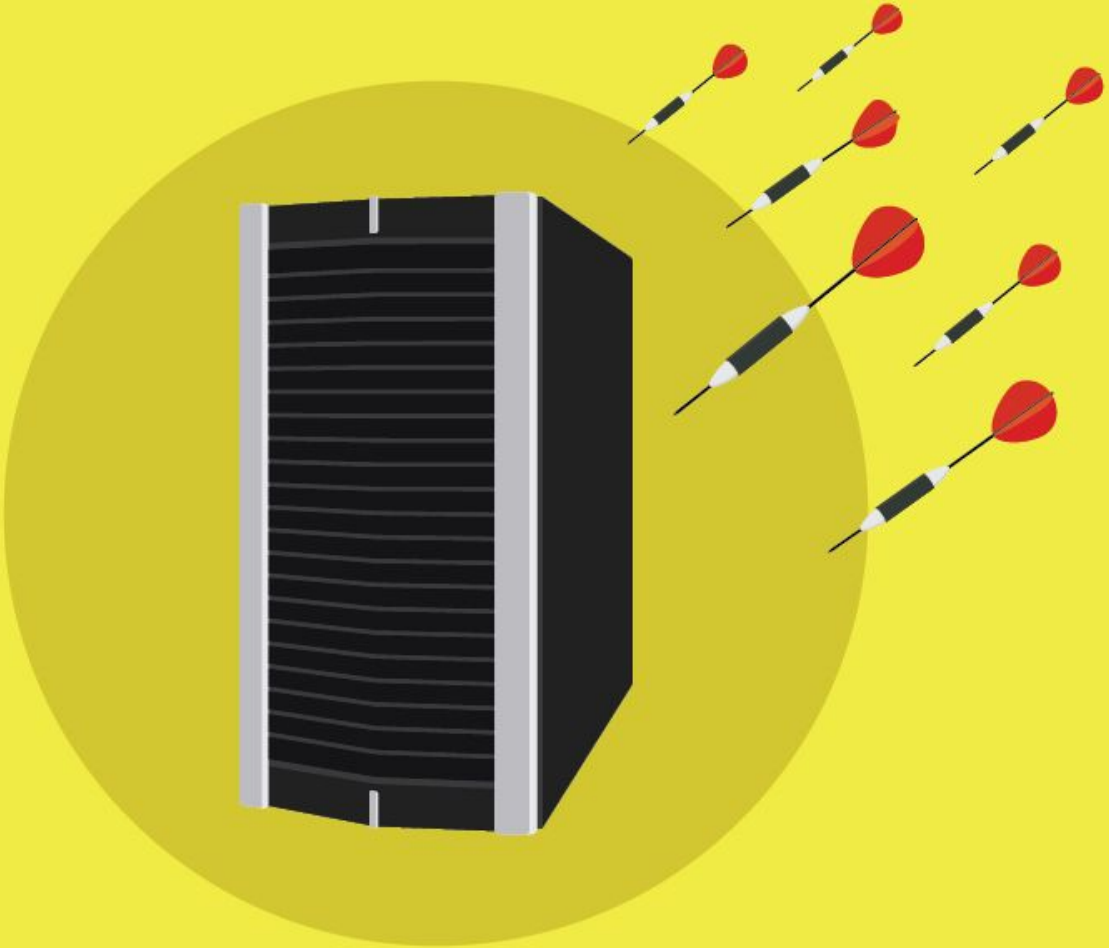
We will focus on vulnerabilities that can exist within **back-end components**.

These components apply the business logic of how the application works, including:

- File structure
- Content management
- Access control

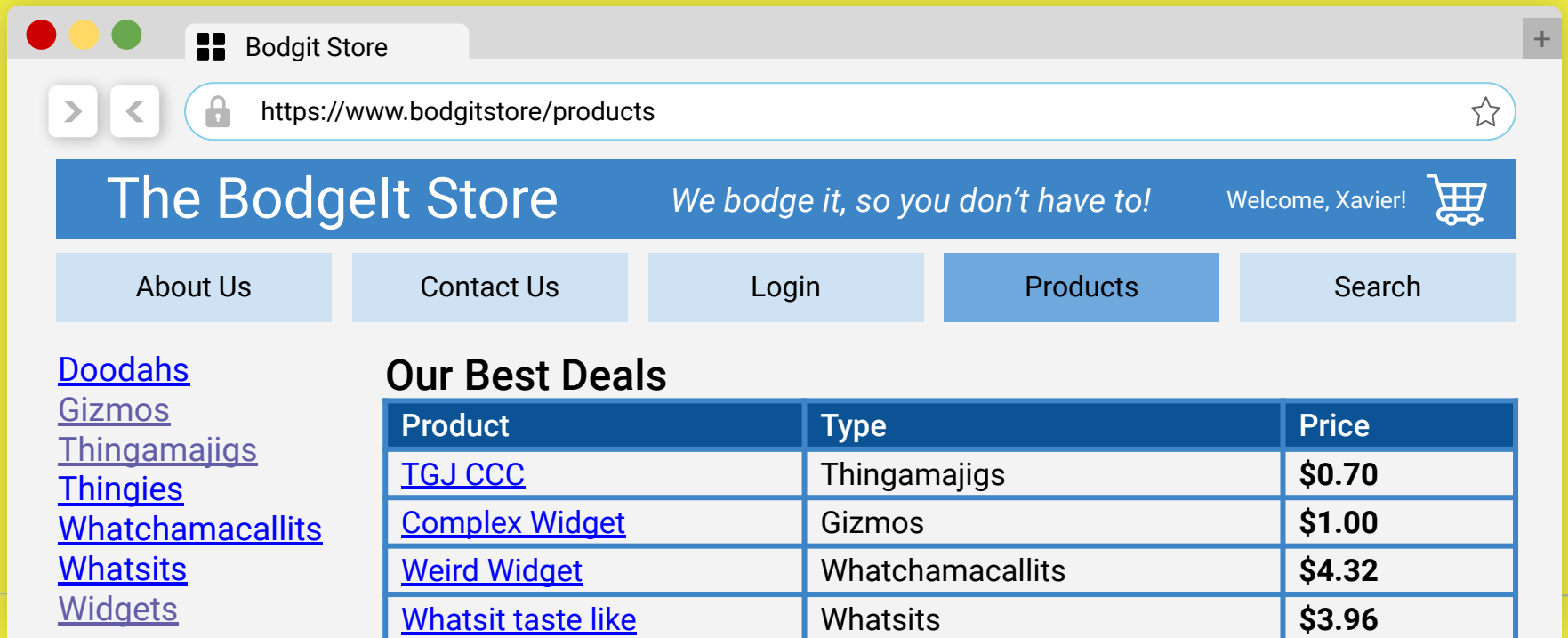
Back-end languages include **PHP** and **Java**.

Vulnerabilities in **back-end components** could potentially allow a malicious actor to access or display confidential files or documents that exist on an organization's private server.



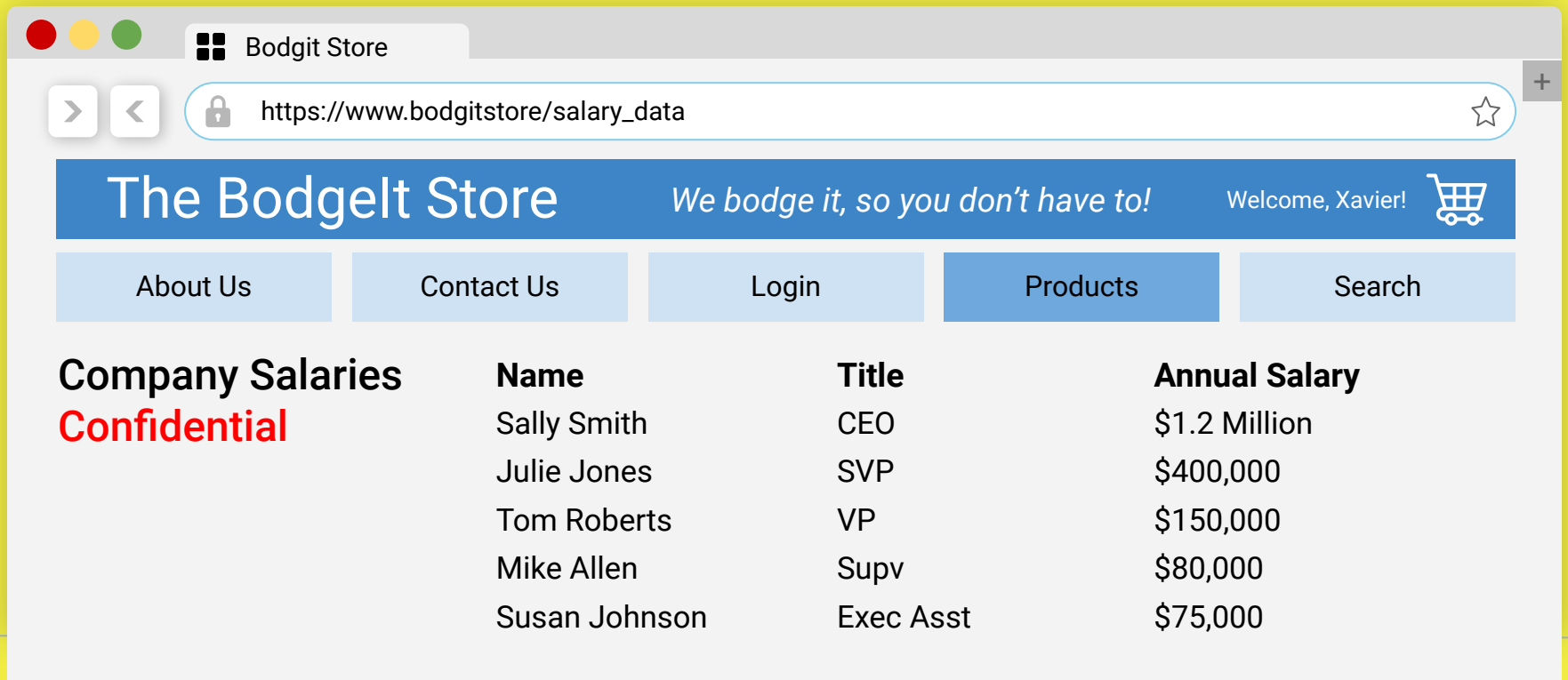
Vulnerabilities: Intended Purpose

An **intended purpose** of an application involves a user accessing a webpage and displaying prices of several products at an online store. The query that retrieves the product prices is the applied business logic.



Vulnerabilities: Unintended Consequence

A malicious actor has changed the URL to access confidential files, such as employee salaries, which are **NOT intended** to be displayed to users.



Today's Class

We will cover three attacks that exist within back-end component vulnerabilities:

First

We will begin by demonstrating how an attacker can use a **directory traversal attack** to access an organization's confidential data.

Next

We will learn how an attacker can use a common web application file upload functionality to conduct a **local file inclusion attack**.

Finally

We will demonstrate another file inclusion attack, called a **remote file inclusion attack**, in which an attacker can reference remote malicious scripts through the URL.

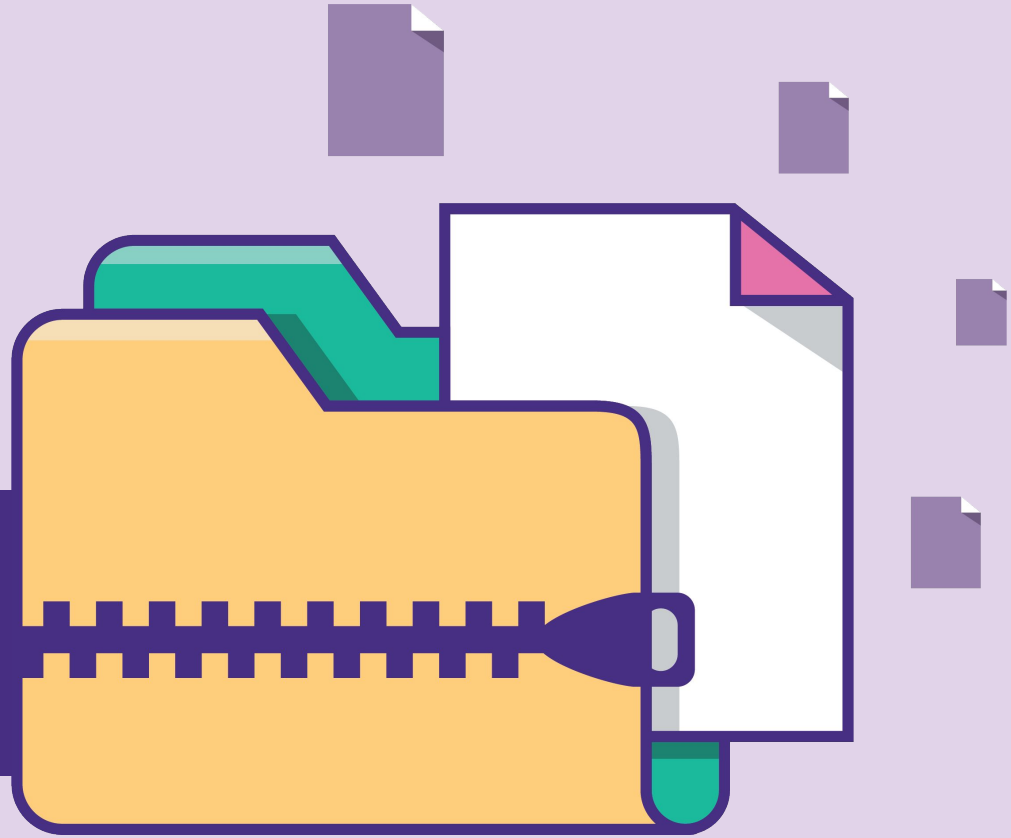
⚠️ Important ⚠️
The techniques that we will learn throughout this unit can be used to cause serious damage to an organization’s systems.
This is ILLEGAL when done without permission. All of the labs that we provide are safe locations to test the methods and tools taught during the week.
NEVER apply any of these methods to any web applications that you do not own or do not have clear, written permission to interact with.

Directory Traversal

Directory traversal, also known as path traversal, occurs when an attacker accesses files and directories from a web application outside a user's authorized permissions.

Directory Traversal

Using this attack, a malicious actor could access confidential business documents, source code, or critical system files that exist within an organization's private server—directly from a web application.



Directory traversal primarily falls under the OWASP Top 10 risk category **broken access control.**

The OWASP Top 10: No.1 Broken Access Control

A01	**Broken Access Control**
A02	Cryptographic Failures
A03	Injection
A04	Insecure Design
A05	Security Misconfiguration
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

Per OWASP, broken access control is explained as follows:

Restrictions on what authenticated users are allowed to do are often not properly enforced.

Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

Understanding Directory Traversal

We first need to understand what happens behind the scenes as an application interacts with the file structure of a web application.

We'll cover the following:

01 The intended purpose of the web application.

02 How the application accesses files and directories from a web server.

03 The modification of the URL to access unintended files.

04 The traversal of directories to access additional unintended files with the **dot-slash** method.

Instructor Demonstration

Directory Traversal

Real-World Challenges, Impact, and Mitigations

Access to Filesystem Structure

The demo illustrated how modifying a URL with the **dot-slash method** can cause unintended results.

In our demonstration...

We were able to preview back-end systems to display the file structures.

But, in the real world...

Application security analysts would likely not have access to the filesystems.

For example:

If `../../../etc/passwd` doesn't display the contents, we can try to keep adding the dot-slashes until the file is displayed:

```
../../../etc/passwd
../../../../etc/passwd
../../../../../etc/passwd
```

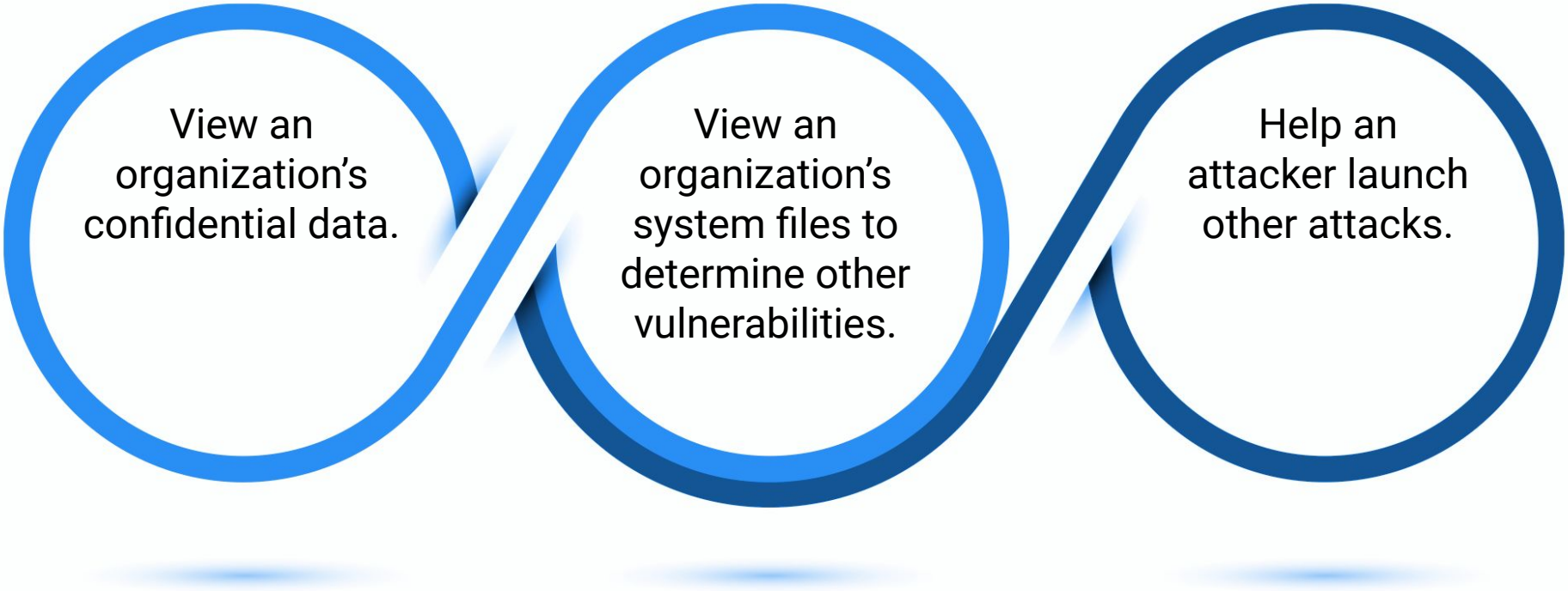
Variety of Confidential Files and Data

While the demonstration involved one file, `/etc/passwd`, many other system files or documents might exist that are not intended to be viewed by an attacker, such as:

`/etc/group`	To view information on groups that exist on the server.
`/etc/hosts`	To view information on how IPs are mapped to hostnames.

Impact

The impact of directory traversal attacks include allowing malicious actors to do the following:



Mitigation Methods

Because a directory traversal attack is conducted by using the dot-slash method to modify user input like the URL, methods to mitigate this attack include:

Limiting user input when calling for files from the web application.

Using **input validation** to limit the user's ability to modify the file being accessed, if the application does require user input when calling for files.

Running web servers under a special service user account that only has access to that web folder. Apache typically uses the `www-data` account.

Directory Traversal Summary

Intended purpose of the original function:	A web application interacts with its web server to access intended files.
Vulnerability and method of exploit:	With the directory traversal attack, a user can modify the user input using a dot-slash method to access unintended files in other directories.
Unintended consequence of the exploit:	Confidential documents or system files can be accessed directly from the web application by a malicious user.
Mitigation of the vulnerability:	Mitigation includes applying input validation code logic or limiting user input when calling for system files.
Potential impact of the exploit:	The impact could include unauthorized parties accessing an organization's confidential data within its servers. This unauthorized data could be used to launch other attacks.

Questions?

Activity: Directory Traversal

In this activity, you'll continue to inspect the Replicants web application for web vulnerabilities, specifically directory traversal.

Suggested Time:

10 Minutes

Time's Up! Let's Review.

Questions?


Break

Web Application Back-End Code

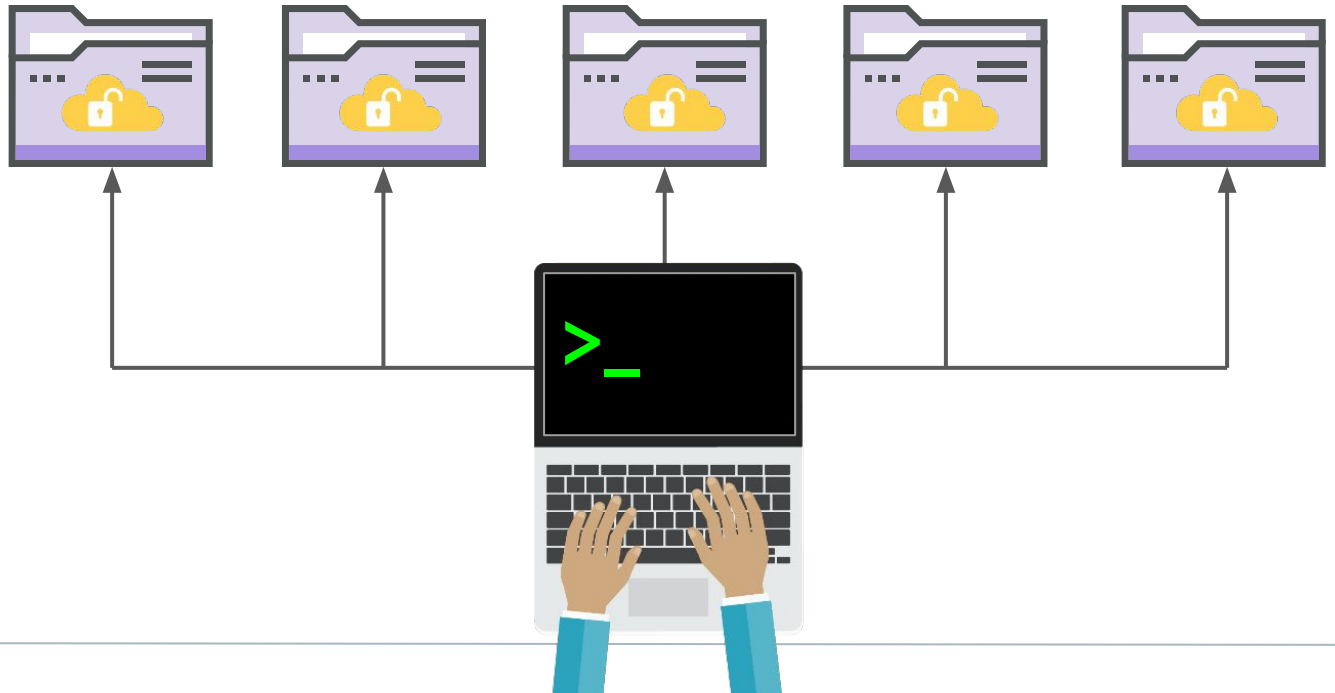
Web Application Back-End Code

Web applications use **back-end components** to apply business logic to the application. Back-end components can provide file structure, content management, and access control and include back-end languages such as **PHP** and **Java**.

Web application vulnerabilities can exist within **back-end components**.

Directory Traversal Has Limited Impact

Directory traversal is a back-end component vulnerability in which an attacker accesses files and directories from a web application outside a user's authorized permissions.



NOTE

While directory traversal is a back-end component vulnerability that can cause significant impact, the direct impact is limited to an attacker viewing unauthorized data.

Another type of back-end component vulnerability is called **local file inclusion (LFI)**. LFI could not only allow the viewing of unauthorized data but also impact the modification and deletion of data, as well as cause system outages.

Local file inclusion is a web application vulnerability in which an attacker tricks the application into running unintended back-end code or scripts that are local to the application's filesystem.

(In other words, the scripts reside on the web application's server.)

Local File Inclusion

Local file inclusion is typically conducted by using a file upload feature to upload malicious scripts into the application's local filesystem. Common examples of file upload web applications include:

Uploading your resume for an online job listing.

Uploading and viewing images on a social media website.

Depending on how the attack is conducted, local file inclusion could fall under the following OWASP Top 10 risks:

A01	**Broken Access Control**
A02	**Cryptographic Failures**
A03	Injection
A04	Insecure Design
A05	**Security Misconfiguration**
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

Before learning how to conduct an LFI exploit, we need to review how a web application runs back-end code like PHP.

PHP

PHP stands for **Hypertext Preprocessor**.

PHP is a server-side language used to develop web applications.

PHP files have a `.php` file extension.

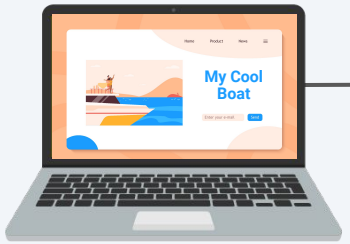
PHP is often used to do the following:

- Connect to back-end databases such as MySQL.
- Create and work with website cookies.

PHP Example

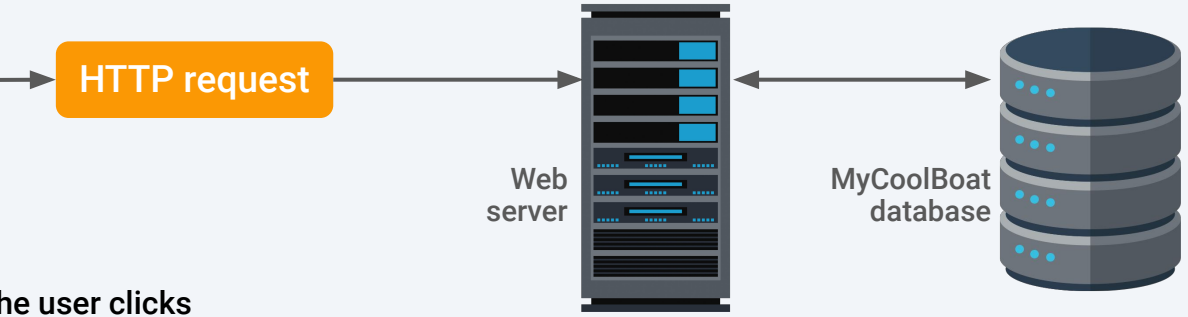
Suppose a user wants to display different types of boats at mycoolboat.com.
The different types of boats are stored in a database on MyCoolBoat's servers.

01 A user visits their favorite boating website: mycoolboat.com.



02 While on mycoolboat.com, the user clicks a Shopping link to redirect to a .php webpage (mycoolboat.com/shopping.php), where boats for sale are displayed.

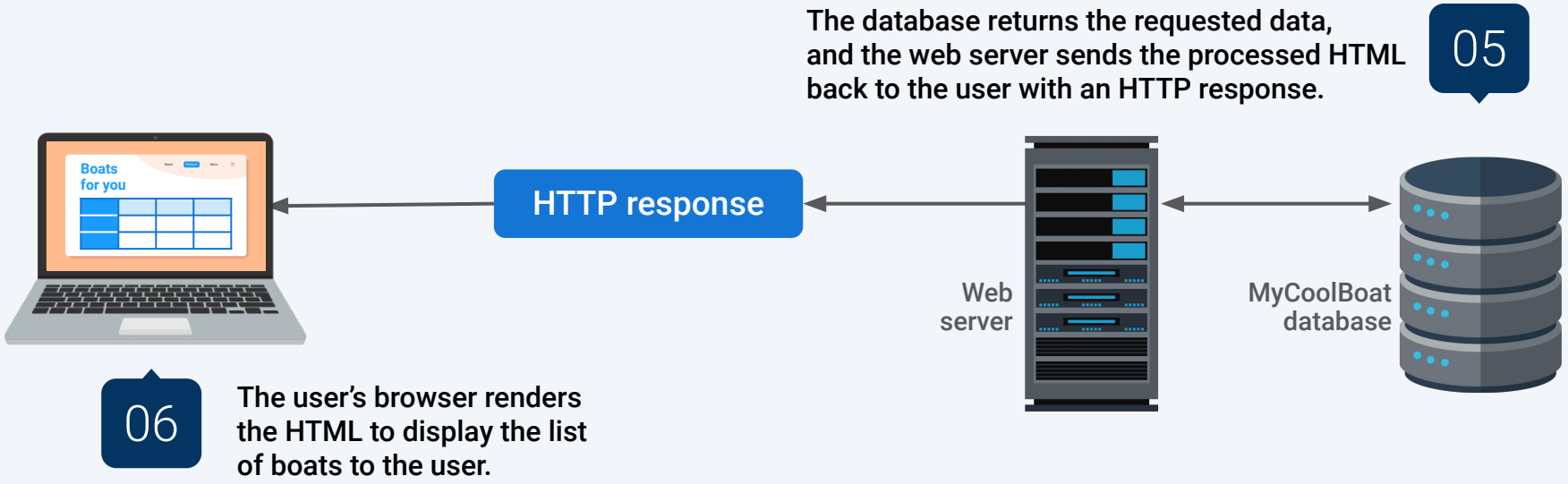
03 A HTTP request is made to the web server.



04 The web server for mycoolboat.com **pre-processes** the PHP scripts to request the list of boats from MyCoolBoat's database.

PHP Example

Suppose a user wants to display different types of boats at mycoolboat.com.
The different types of boats are stored in a database on MyCoolBoat's servers.



Takeaways

In the previous example:

The browser initiated a request to run the PHP script simply by accessing the page.

The PHP script exists and runs on the back-end server.

Now, we know that PHP is a language that provides back-end functionality to a webpage, so let’s explore how local file inclusion can use this functionality to cause unintended results.

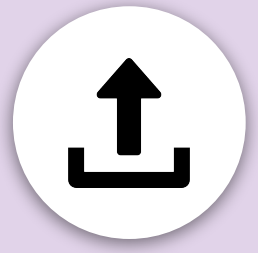
Local File Inclusion (LFI)

LFI Demonstration

In this demonstration, we will:

Illustrate

how a web application can use PHP and HTML together.



Show

the intended purpose of a web application's file upload functionality.

Upload

a malicious PHP script.

Run

command-line commands with the malicious PHP script.

Instructor Demonstration

Local File Inclusion

LFI Summary

The ability to successfully upload and run this script indicates that this web application is vulnerable to **local file inclusion**.

We were able to **include** (upload) **a file** (script) **locally** on the target filesystem, then have that unintended script run.

The ability to arbitrarily execute command-line code with the script means that this web application is also vulnerable to **remote code execution**.

Remote code execution (**RCE)**, is a type of attack in which a malicious actor can execute arbitrary commands on a target machine.

Real-World Challenges, Impact, and Mitigations

Real-World Challenges

Variety of attacks:

Our demonstration

Illustrated using one type of malicious script and two command-line commands.

Thousands of malicious scripts can be uploaded, and many malicious commands can be run.

Our attack method

Used the PHP language, but this attack can also be accomplished with other languages, such as ASP.

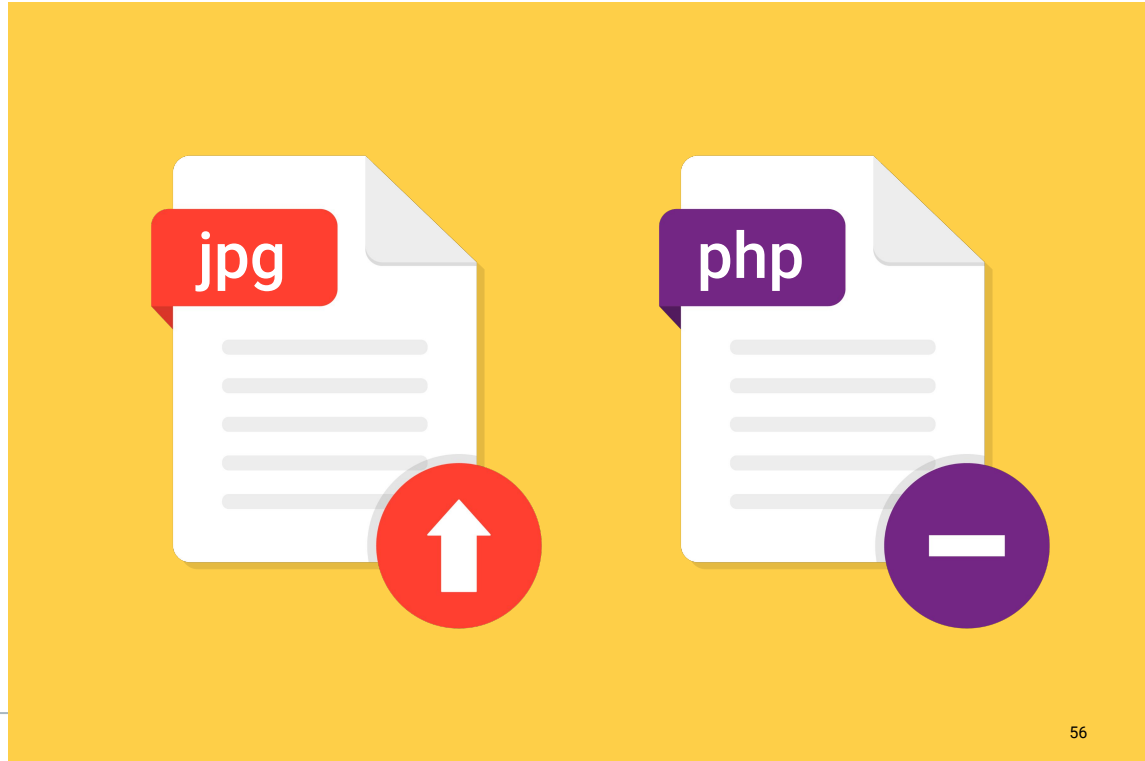
Impact

If a malicious actor can upload a malicious script, they can potentially:

Mitigation Methods

The best way to protect from LFI is to restrict users from being able to upload files into the local filesystem.

If the application requires file upload functionality, then the application should use **allow listing** to ensure that only a specific file type (like `.jpg`) can be uploaded, and not script file types such as `.php`.



Local File Inclusion Summary

Intended purpose of the original function:	Web applications use file upload functionalities to enable a user to upload files into their local filesystem.
Vulnerability and method of exploit:	Local file inclusion is a vulnerability in which a malicious user can use the file upload functionality to upload unintended scripts into the web server's local filesystem.
Unintended consequence of the exploit:	After the malicious script has been uploaded, a malicious user can run the script to cause unintended consequences.
Mitigation of the vulnerability:	Limit file upload capabilities or use allow listing to restrict unintended file types.
Potential impact of the exploit:	A malicious actor can access or modify confidential data or system files or potentially cause system outages.

Questions?

Activity: Local File Inclusion

In this activity, you'll inspect the Replicants main production website for local file inclusion vulnerabilities.

Suggested Time:

20 Minutes

Time's Up! Let's Review.

Questions?

Remote File Inclusion

Remote File Inclusion

So far, we've covered:

Directory traversal

A back-end component vulnerability in which an attacker accesses files and directories from a web application outside a user's authorized permissions.

Local file inclusion

Another back-end component vulnerability in which an attacker tricks the application into running unintended back-end code or scripts that are LOCAL to the application's filesystem.

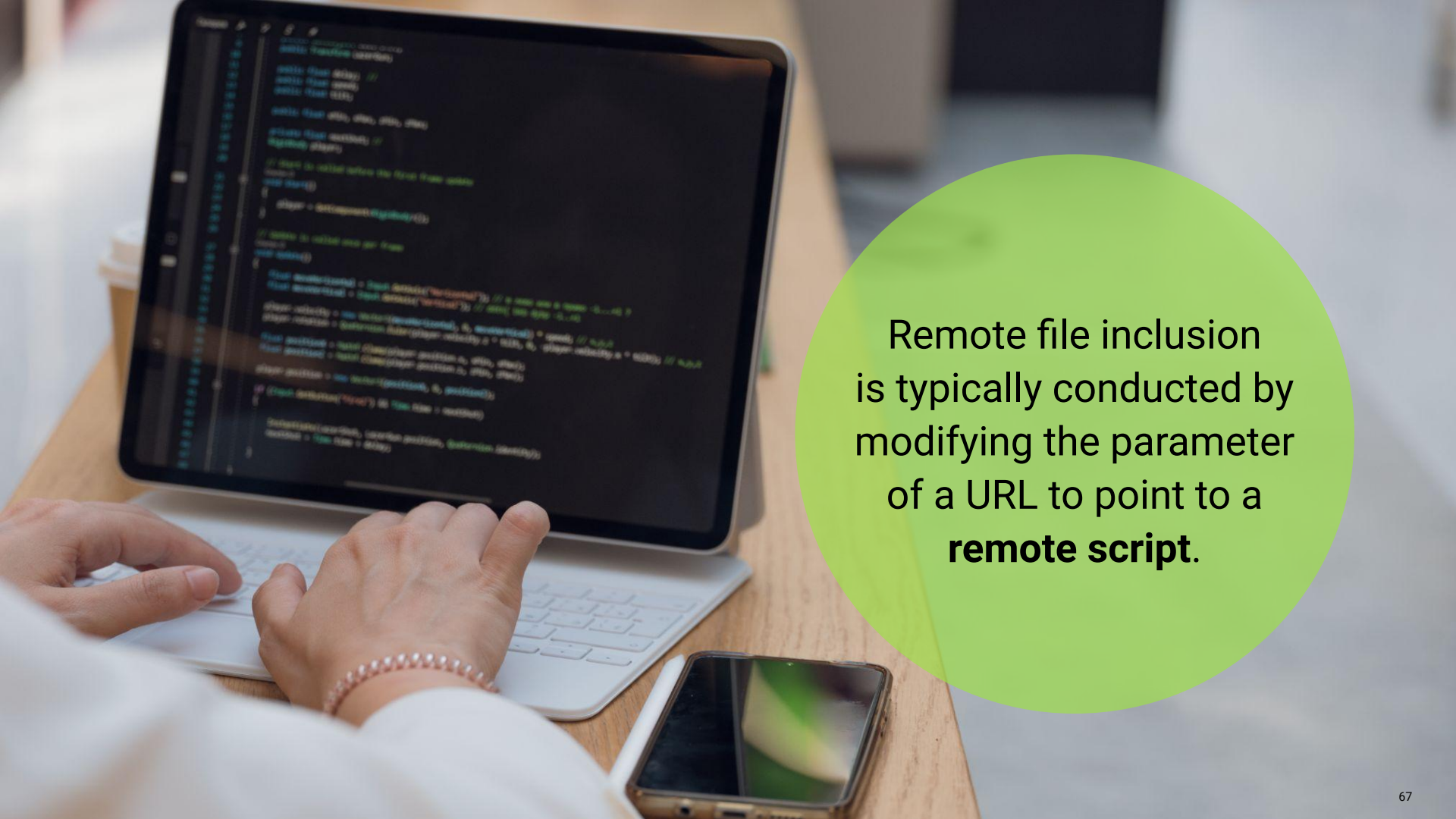
Local file inclusion depends on allowing a user to use unintended scripts that are LOCAL to the web server.

An application preventing malicious users from adding files to their local system **does not guarantee** that it is protected from an attacker running unintended back-end code or scripts.


Another type of back-end component vulnerability called **remote file inclusion (RFI)** involves using remote files or scripts to conduct a similar attack.

Remote file inclusion (RFI) is a web application vulnerability in which an attacker tricks the application into running unintended back-end code or scripts that are remote to the application's filesystem.

(In other words, the scripts DO NOT reside on the web application's server.)



Remote file inclusion is typically conducted by modifying the parameter of a URL to point to a **remote script**.

Depending on how the attack is conducted, remote file inclusion could fall under the following OWASP Top 10 risks:

A01	**Broken Access Control**
A02	**Cryptographic Failures**
A03	Injection
A04	Insecure Design
A05	**Security Misconfiguration**
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

RFI Demonstration

In this demonstration, we will conduct the following three steps:

Revisit how a web application uses parameters as intended.

Modify the parameters to point to a remote site to cause unintended consequences.

Modify the URL to run this remote malicious PHP script.

Instructor Demonstration

Remote File Inclusion

Demo Summary

The ability to successfully change the parameter of the URL to point to a remote script and then subsequently running this script indicates that this web application is vulnerable to **remote file inclusion**.

The ability to arbitrarily execute command-line code with the remote script means this web application is also vulnerable to **remote code execution.**

```
../../hackable/uploads/image.jpg successfully uploaded!
```

Real-World Challenges, Impact, and Mitigations

Real-World Challenges

Limitations of the RFI attack

- To conduct this attack successfully, the web server not only needs to run PHP but also needs to enable a configuration setting to allow external URLs.
- Many web servers that run PHP do not have this configuration enabled.

Variety of attacks

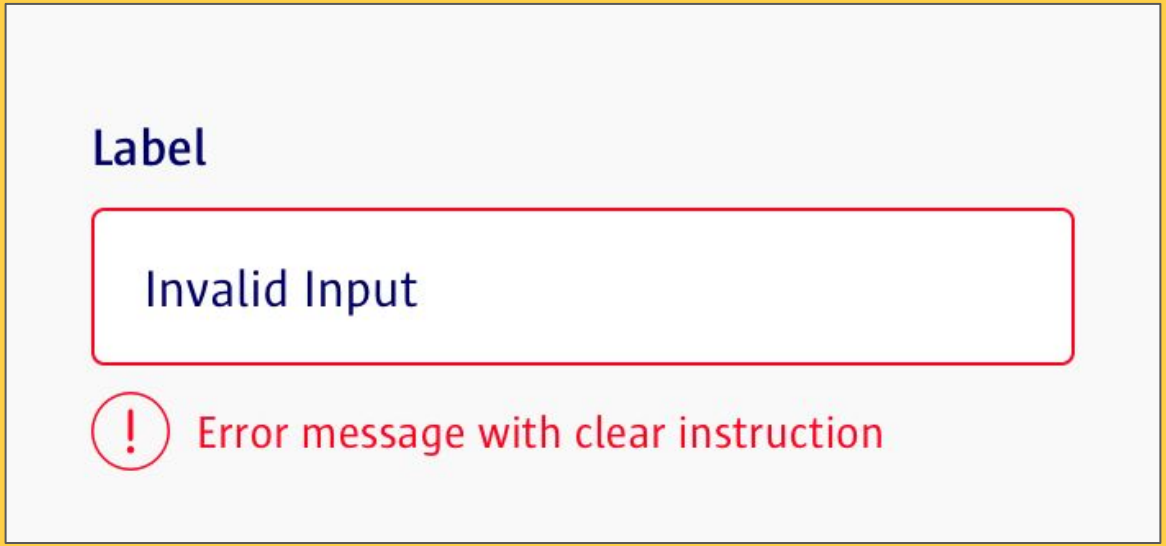
- The demonstration illustrated using one type of malicious script and two command-line commands.
- Thousands of malicious scripts can be uploaded, and many malicious commands can be run.
- The attack method used the PHP language, but this attack can also be accomplished with other languages, such as ASP.

Impact

If a malicious actor can remotely access a malicious script, they can:

Mitigation Methods

The best way to protect from RFI is to avoid passing user-submitted input to your web server. This can be accomplished by using input validation.

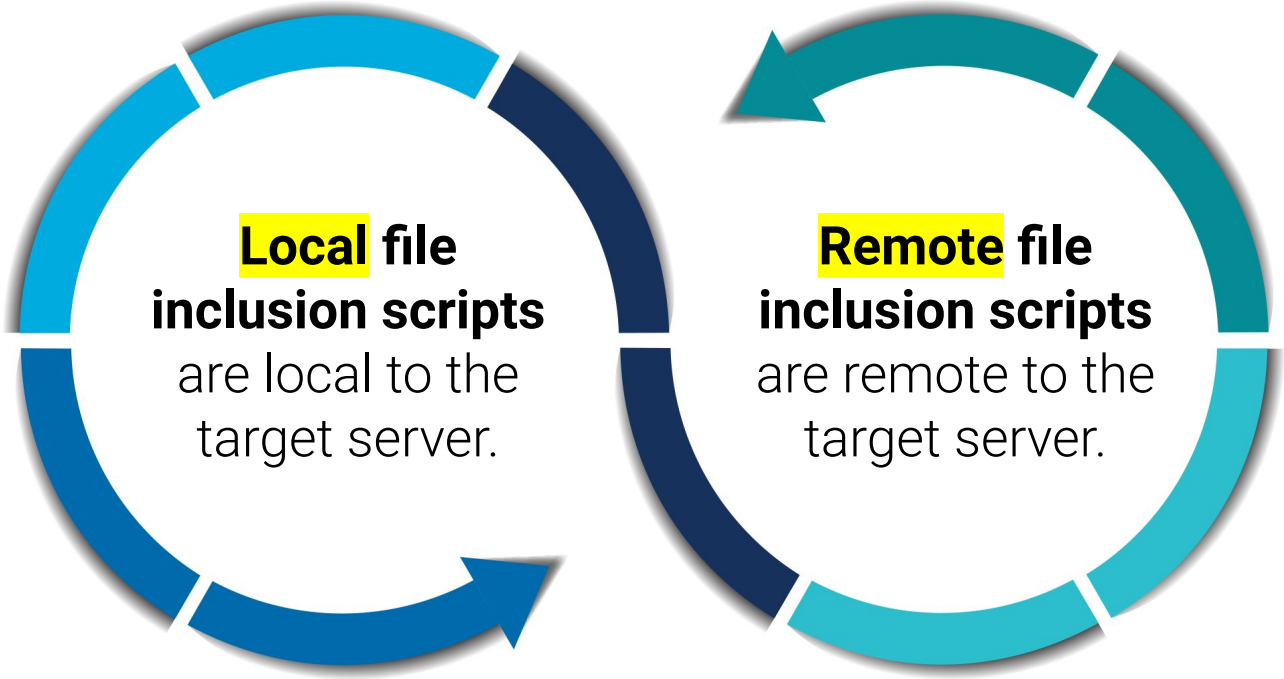


Remote File Inclusion Summary

Intended purpose of the original function:	A web application can use parameters in its URL to indicate to the back-end server to access a certain resource.
Vulnerability and method of exploit:	Remote file inclusion is a vulnerability in which a malicious user can modify this parameter so that the back-end server accesses a malicious remote resource, such as a script.
Unintended consequence of the exploit:	After the malicious remote script has been referenced, a malicious user can run the script to cause unintended consequences.
Mitigation of the vulnerability:	Use input validation to avoid malicious submitted input, such as a modified parameter referencing a remote resource.
Potential impact of the exploit:	A malicious actor can view or modify confidential data or system files or potentially cause system outages.

Recap Back-End Components

Both of these attacks use back-end components of a web server to run unintended scripts, but they have some differences:



Questions?

Activity: Remote File Inclusion

In this activity, you will test the Replicants main production website for remote file inclusion vulnerabilities.

Suggested Time:

20 Minutes

Time's Up! Let's Review.

Questions?


The End