



# Back-End Component Vulnerabilities

Cybersecurity

15.2 Web Vulnerabilities and Hardening



# Class Objectives

---

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



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



Access 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.

# Last Class...

---

Remind the class that in the second half of the last class, we covered cross-site scripting vulnerabilities that exist with **JavaScript** and **HTML**.



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



Users typically use a browser to interact with these components.



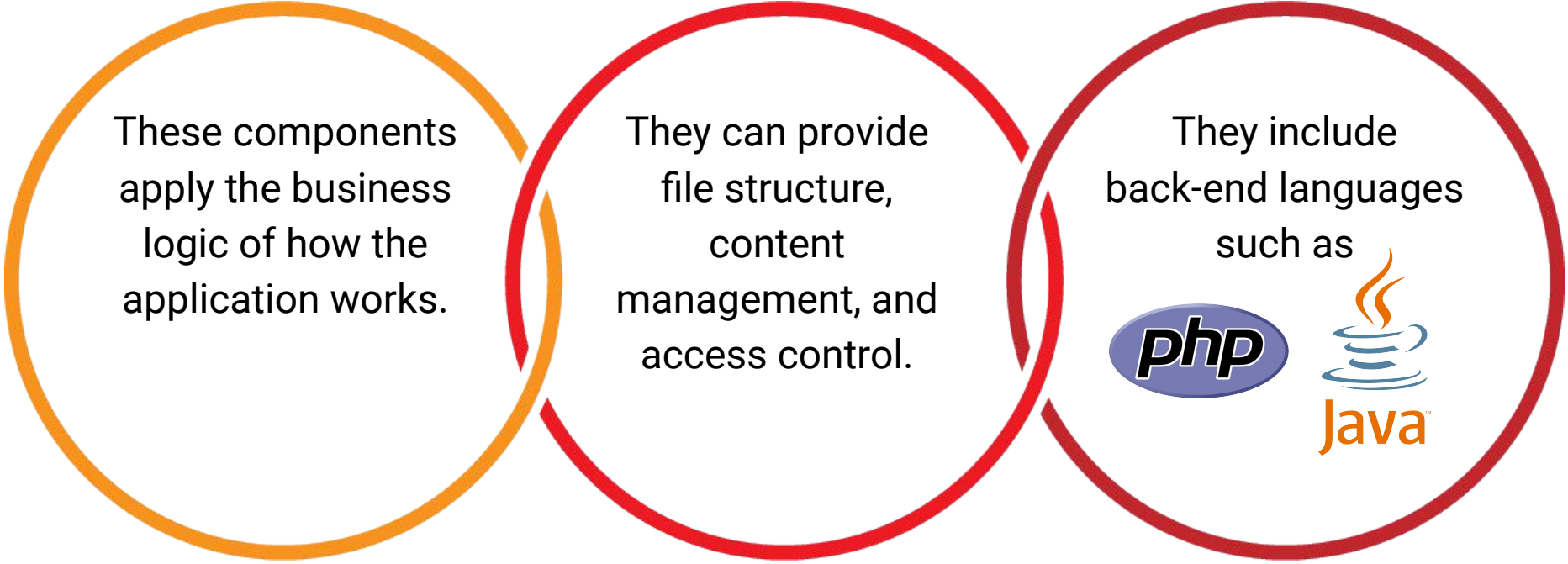
Another front-end component is **Cascading Style Sheets (CSS)**, which formats the layout of the HTML.

# Intro to Back-End Component Vulnerabilities

# 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.

They can provide  
file structure,  
content  
management, and  
access control.

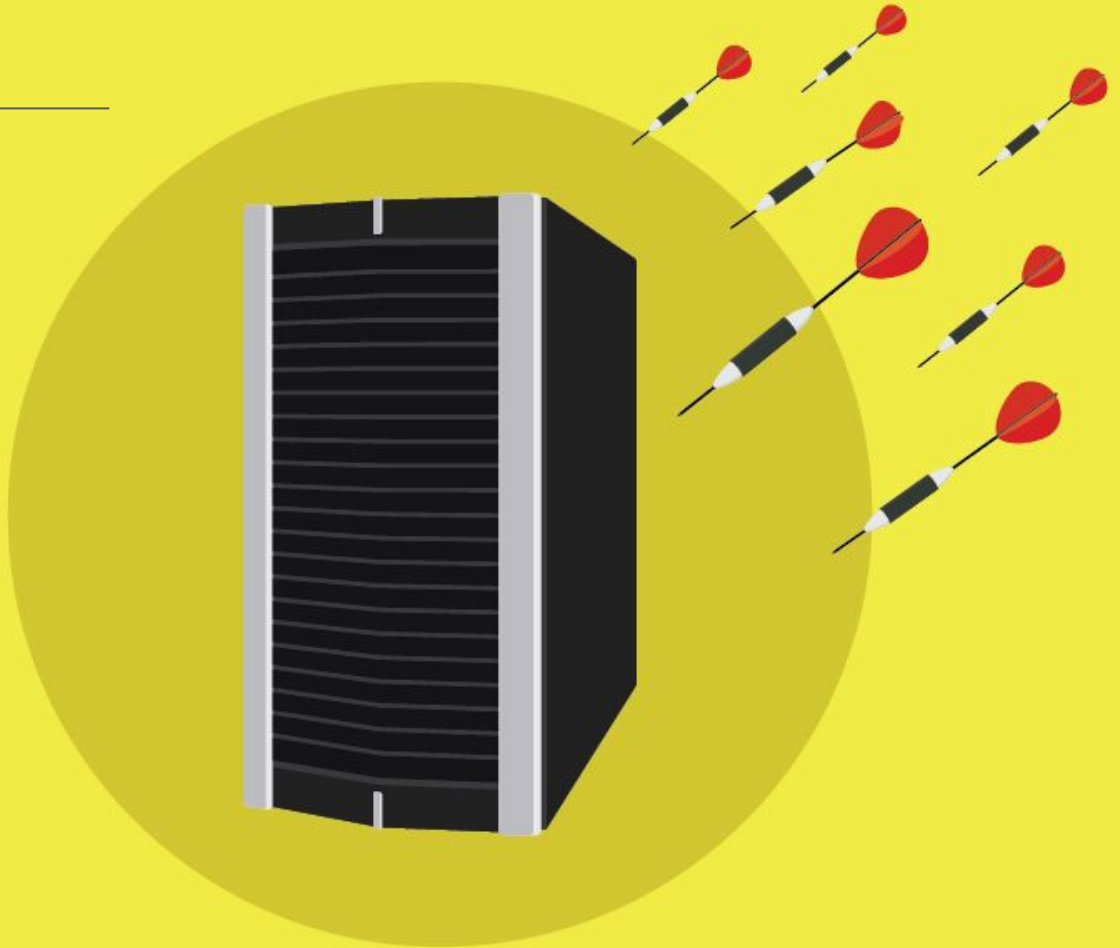
They include  
back-end languages  
such as



# Vulnerabilities

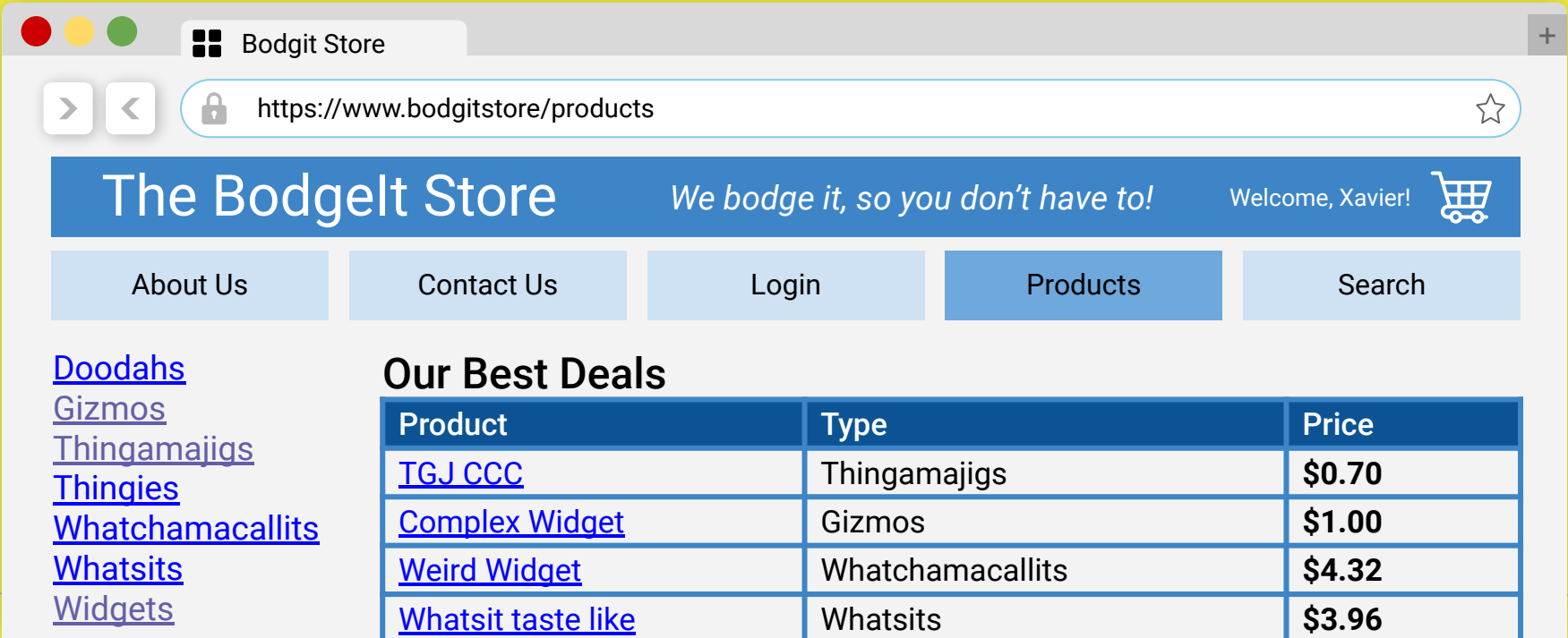
---

Vulnerabilities in back-end components could allow a malicious actor to potentially 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.

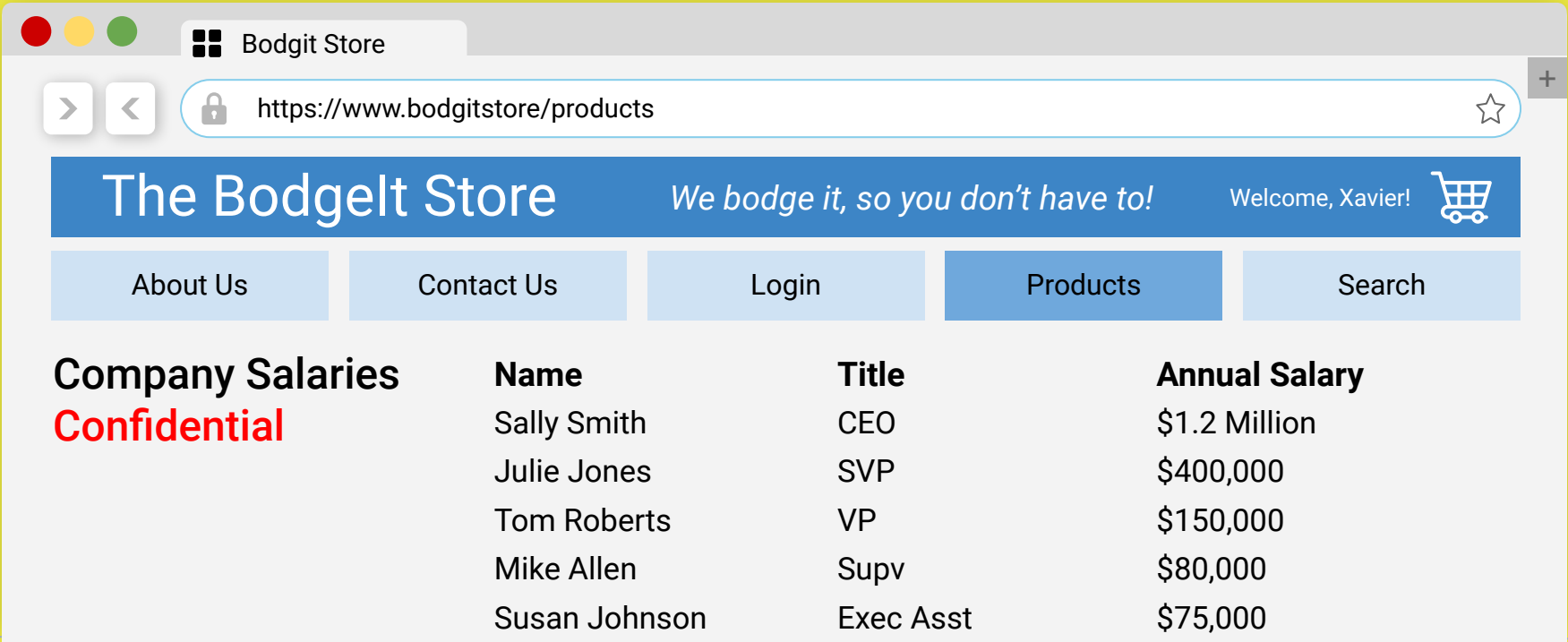


The screenshot shows a web browser window with the address bar displaying <https://www.bodgitstore/products>. The website header features the logo "The Bodgelt Store" with the tagline "We bodge it, so you don't have to!" and a welcome message "Welcome, Xavier!". A navigation bar includes links for "About Us", "Contact Us", "Login", "Products", and "Search". On the left side, there is a vertical list of product categories: [Doodahs](#), [Gizmos](#), [Thingamajigs](#), [Thingies](#), [Whatchamacallits](#), [Whatsits](#), and [Widgets](#). The main content area is titled "Our Best Deals" and contains a table with three columns: Product, Type, and Price.

Product	Type	Price
<a href="#">TGJ CCC</a>	Thingamajigs	\$0.70
<a href="#">Complex Widget</a>	Gizmos	\$1.00
<a href="#">Weird Widget</a>	Whatchamacallits	\$4.32
<a href="#">Whatsit taste like</a>	Whatsits	\$3.96

# 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.



The screenshot shows a web browser window with the address bar displaying `https://www.bodgitstore/products`. The website is "The Bodgelt Store" with the tagline "We bodge it, so you don't have to!". The user is logged in as "Xavier". The navigation menu includes "About Us", "Contact Us", "Login", "Products", and "Search". The main content area displays a table titled "Company Salaries" with a red "Confidential" label. The table lists the following employees and their salaries:

Name	Title	Annual Salary
Sally Smith	CEO	\$1.2 Million
Julie Jones	SVP	\$400,000
Tom Roberts	VP	\$150,000
Mike Allen	Supv	\$80,000
Susan Johnson	Exec Asst	\$75,000



# Today's Class..

---

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

## Directory Traversal

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

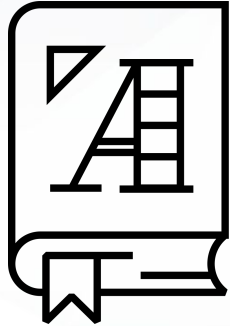
## Local File Inclusion

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

## Remote File Inclusion

Lastly, 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.

# 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**.





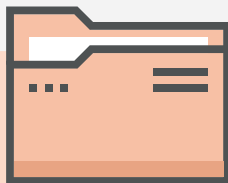
# Impact

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

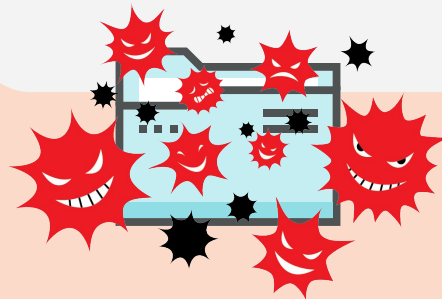
Access an organization's confidential data.



Access an organization's systems files to determine other vulnerabilities.



Help an attacker launch other attacks.



# The OWASP Top 10: No.5 Broken Access Control



**Broken access control is defined as:**

“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 will first need to understand what happens behind the scenes as an application interacts with the file structure of a web application. This will be demonstrated by covering the following:

01

The intended purpose of the web application.

02

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

03

Modification of the URL to access unintended files.

04

Traversing directories to access additional unintended files with the dot-slash method.



# Instructor Demonstration

## Directory Traversal





# **Real-World Challenges, Impact, and Mitigations**



# Real-World Challenges

While the purpose was to illustrate how modifying the URL with the **dot-slash method** can cause unintended results, in the real world, we need to consider the following:

## In our demonstration

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

## 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`



# Real-World Challenges

While the purpose of the lesson was to illustrate how modifying the URL with the **dot-slash method** can cause unintended results, in the real world, we need to consider the following:

## In our demonstration

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

**`/etc/groups`**

To display information on groups that exist on the server

**`/etc/hosts`**

To display information on how IPs are mapped to hostnames



# Mitigation Methods

---

Methods that can be used to mitigate this attack include:



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



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

# 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 their servers. This unauthorized data could be used to launch other attacks.



## **Activity:** Directory Traversal

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

**Suggested Time:**  
20 Minutes





**Time's Up!** Let's Review.

# Web Application Back-End Code



# Web Application Back-End Code

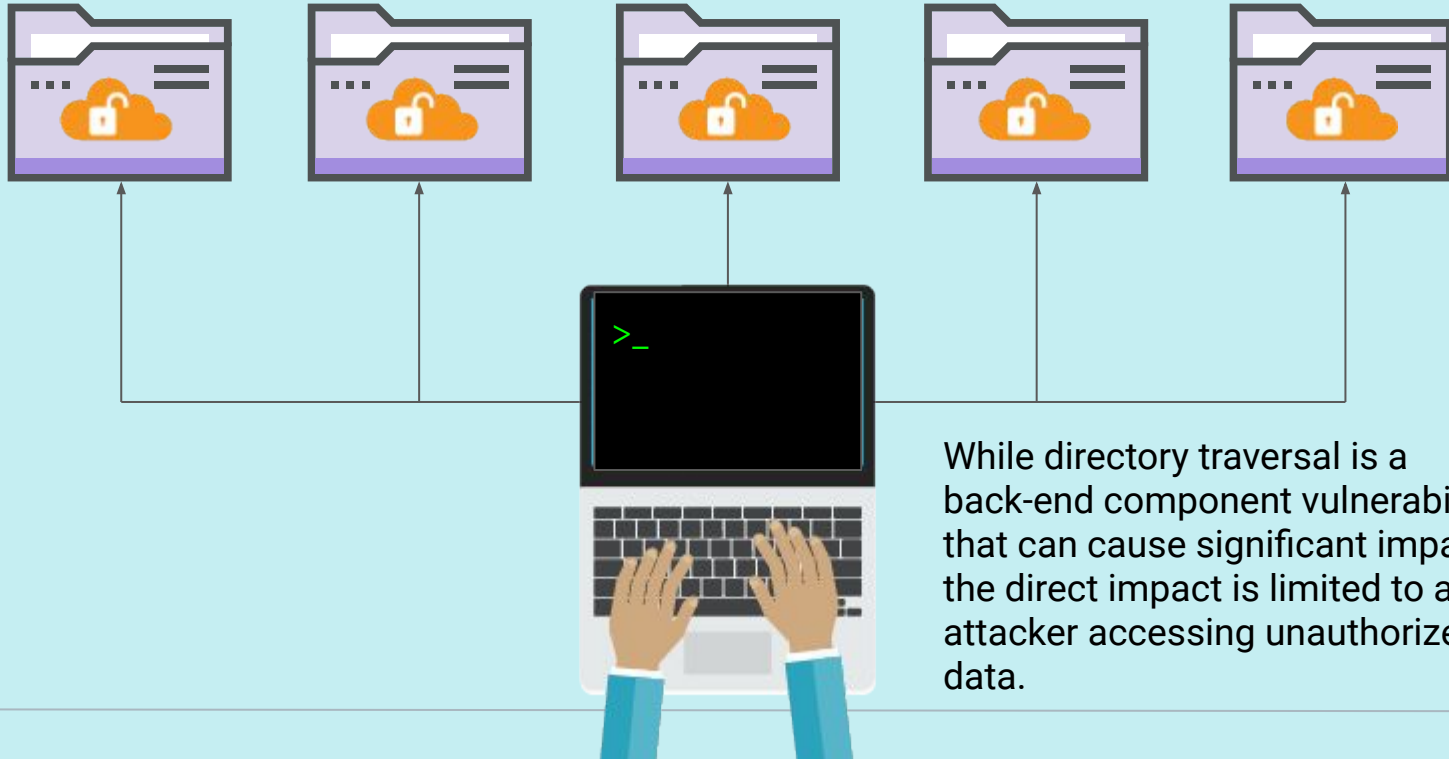
---

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.



# 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.



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

# Local File Inclusion

---

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

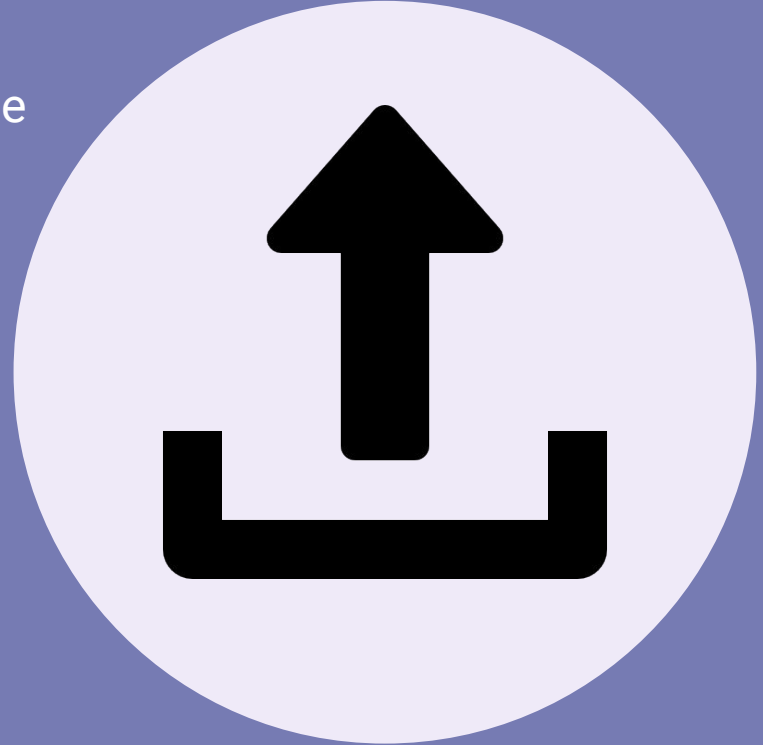
Some of the common examples of file upload web applications are:



Uploading your resume on a job listing



Uploading and displaying images on a social media website



# The OWASP Top 10: No. 3, 5, and 6



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

- Sensitive data exposure
- Broken access control
- Security misconfiguration



Before we learn the details of conducting an LFI exploit, we need to review how a web application runs back-end code like PHP.

# PHP

PHP stands for **H**ypertext **P**reprocessor.



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



PHP files have a **.php** file extension



PHP is often used to:

- 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. While on mycoolboat.com, the user clicks a Shopping link to redirect to a `.php` webpage (mycoolboat.com/shopping.php) where they can display boats for sale.



HTTP request

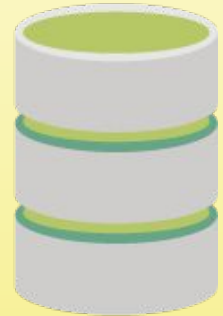
02

A HTTP request is made to the web server.

Webserver



MyCoolBoat database



03

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

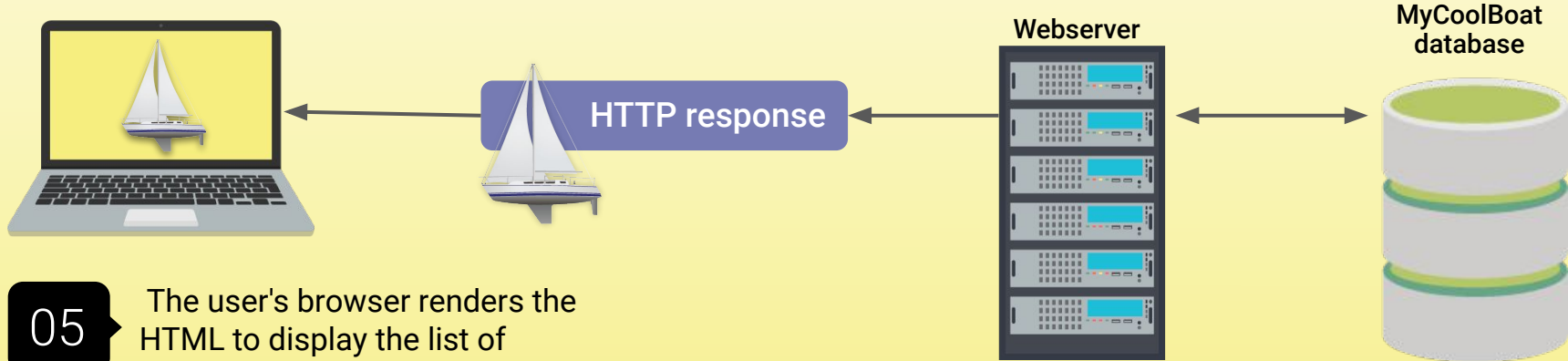
# PHP Example



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.

04

The database returns the requested data, and the web server sends the processed HTML back to the user with an HTTP response.



05

The user's browser renders the HTML to display the list of boats back to the user.



# LFI Demo

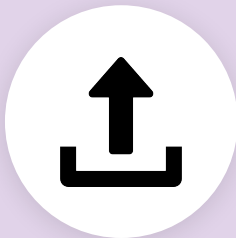
---

In this demonstration, we will do the following:



## Illustrate

How a web application can use PHP and HTML together.



## Show

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

```
<script></script>
```

## Upload

A malicious php script



## Run

Command line commands with the malicious PHP script



# Instructor Demonstration

## Local File Inclusion

# LFI Summary

---

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



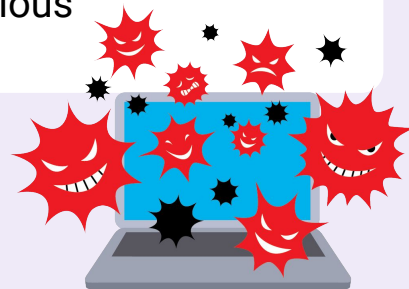
We were able to **include** (upload) a **file** (script) **locally** on the target filesystem, then have that unintended script run.



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

---

**Local file inclusion** is considered a harmful attack due to the impact it can cause.


If a malicious actor can upload a malicious script, they can potentially do the following:



Access confidential files or data.



Modify or delete confidential files or data.



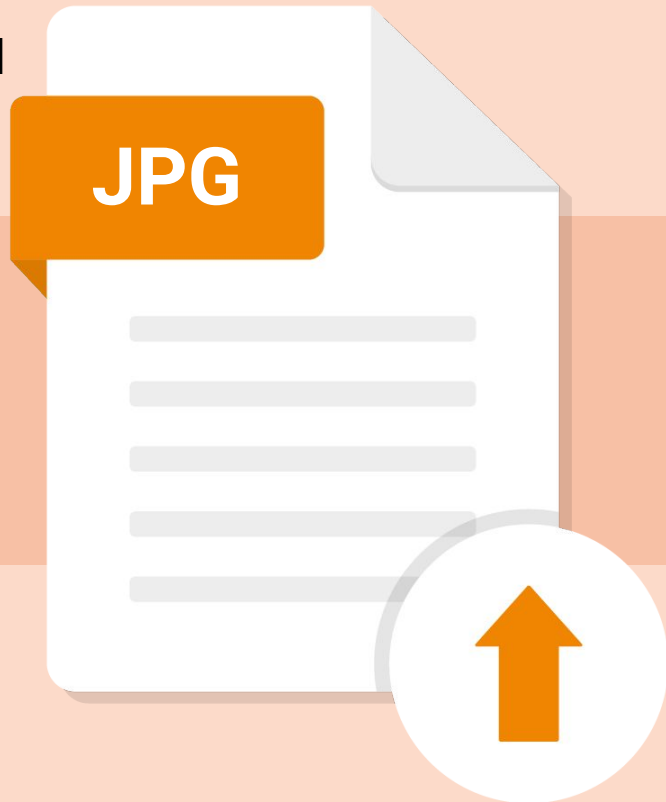
Modify or delete critical files to cause system disruptions.



# 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 this upload functionality, then the application should use **allow listing** to ensure that only a specific file type like `.jpg` is uploaded, and not an arbitrary script file type, 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.





## **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.

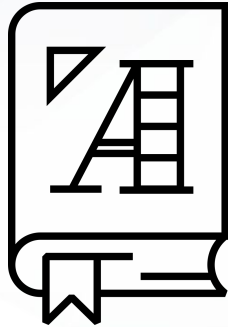
A close-up, high-angle shot of a computer keyboard. The central focus is a large, white, rectangular key with rounded corners. On this key, there is a dark blue icon of a coffee cup with three wavy lines above it representing steam. Below the icon, the word "Break" is printed in a dark blue, serif font. The key is set against a light-colored, textured keyboard surface. Surrounding the main key are other keys: to the left, a key with double quotation marks; above, a key with a right square bracket; and to the right, a key with a left square bracket. The lighting is soft and even, highlighting the texture of the keys and the surface.

Break

# Remote File Inclusion



Just because an application doesn't allow a malicious user to add files into their local system, it doesn't guarantee that they are protected from an attacker being able to run unintended back-end code or scripts.



**Remote file inclusion (RFI)** involves using remote files or scripts to conduct a similar attack.

# Remote File Inclusion (RFI)

---

An attacker tricks the application to run unintended back-end code or scripts that are remote to the application's filesystem.

Remote scripts do NOT reside on the web application's server.



# The OWASP Top 10: No. 3, 5, and 6



Similar to local file inclusion, depending on how the attack is conducted, remote file inclusion can fall under the following OWASP Top 10 risks:

- Broken access control
- Sensitive data exposure
- Security misconfiguration



# RFI Demonstration

---

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

01

Revisit how a web application uses parameters as intended.

02

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

03

Modify the URL to run this remote malicious PHP script.



# Instructor Demonstration

## Remote File Inclusion

# Demo Summary

---

Being able 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 do the following:



Access confidential files or data.



Modify or delete confidential files or data.



Modify or delete critical files to cause system disruptions.



# 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.

Label

Invalid Input



Error message with clear instruction

# Remote File Inclusion Summary

---

Intended purpose of the original function	Web applications can use parameters in their URL to indicate 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 an remote resource.
Potential impact of the exploit	A malicious actor can access or modify confidential data or system files or potentially cause system outages.



# Recap Back-End Components

---

## Local File Inclusion

Unintended scripts are **local** to the target server.

## Remote File Inclusion

Unintended scripts are **remote** to the target server.



## **Activity:** Remote File Inclusion

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

**Suggested Time:**  
10 Minutes





**Time's Up!** Let's Review.

# Class Objectives

---

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



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



Access 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.

*The  
End*