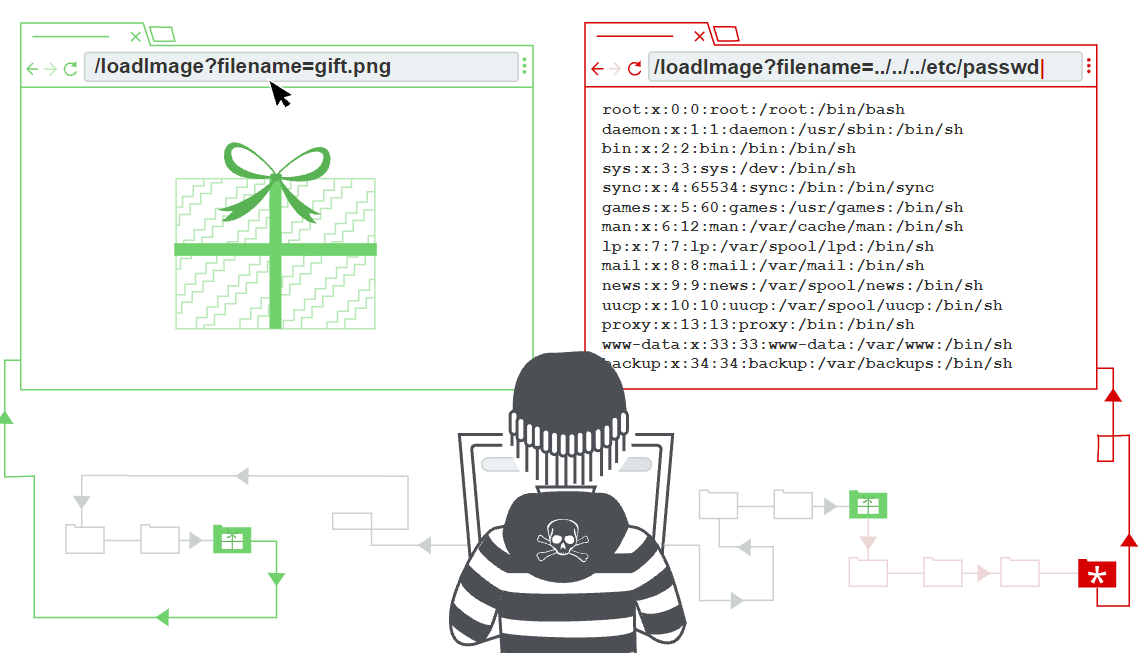
Directory traversal

In this section, we'll explain what directory traversal is, describe how to carry out path traversal attacks and circumvent common obstacles, and spell out how to prevent path traversal vulnerabilities.



 is a web security vulnerability that allows an attacker to **read arbitrary files on the server** that is running an application. This might include application **code and data, credentials** for back-end systems, and **sensitive operating system files**. In some cases, an attacker might be able to **write to arbitrary files on the server**, allowing them to modify application data or behavior, and ultimately take full control of the server.

## **Reading arbitrary files via directory traversal**

Consider a shopping application that displays images of items for sale. Images are loaded via some HTML like the following:

<img src="/loadImage?filename=218.png">

The loadImage URL takes a filename parameter and returns the contents of the specified file. The image files themselves are stored on disk in the location /var/www/images/. To return an image, the application appends the requested filename to this base directory and uses a filesystem API to read the contents of the file. In the above case, the application reads from the following file path:

/var/www/images/218.png

The application implements no defenses against directory traversal attacks, so an attacker can request the following URL to retrieve an arbitrary file from the server's filesystem:

https://insecure-website.com/loadImage?filename=../../../etc/passwd

This causes the application to read from the following file path:

/var/www/images/../../../etc/passwd

The sequence ../ is valid within a file path, and means to step up one level in the directory structure. The three consecutive ../ sequences step up from /var/www/images/ to the filesystem root, and so the file that is actually read is:

/etc/passwd

On Unix-based operating systems, this is a standard file containing details of the users that are registered on the server.

On Windows, both ../ and ..\ are valid directory traversal sequences, and an equivalent attack to retrieve a standard operating system file would be:

https://insecure-website.com/loadImage?filename=..\..\..\windows\win.ini

# Lab: File path traversal, simple case

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value ../../../etc/passwd.
3. Observe that the response contains the contents of the /etc/passwd file.

## **Common obstacles to exploiting file path traversal vulnerabilities**

Many applications that place user input into file paths implement some kind of defense against path traversal attacks, and these can often be circumvented.

If an application strips or blocks directory traversal sequences from the user-supplied filename, then it might be possible to bypass the defense using a variety of techniques.

You might be able to use an absolute path from the filesystem root, such as filename=/etc/passwd, to directly reference a file without using any traversal sequences.

# Lab: File path traversal, traversal sequences blocked with absolute path bypass

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value /etc/passwd.
3. Observe that the response contains the contents of the /etc/passwd file.

You might be able to use nested traversal sequences, such as ....// or ....\/, which will revert to simple traversal sequences when the inner sequence is stripped.

# Lab: File path traversal, traversal sequences stripped non-recursively

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value: ....//....//....//etc/passwd
3. Observe that the response contains the contents of the /etc/passwd file.

You might be able to use various non-standard encodings, such as ..%c0%af or ..%252f, to bypass the input filter.

# Lab: File path traversal, traversal sequences stripped with superfluous URL-decode

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value ..%252f..%252f..%252fetc/passwd.
3. Observe that the response contains the contents of the /etc/passwd file.

If an application requires that the user-supplied filename must start with the expected base folder, such as /var/www/images, then it might be possible to include the required base folder followed by suitable traversal sequences. For example:

filename=/var/www/images/../../../etc/passwd

# Lab: File path traversal, validation of start of path

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value /var/www/images/../../../etc/passwd.
3. Observe that the response contains the contents of the /etc/passwd file.

If an application requires that the user-supplied filename must end with an expected file extension, such as .png, then it might be possible to use a null byte to effectively terminate the file path before the required extension. For example:

filename=../../../etc/passwd%00.png

# Lab: File path traversal, validation of file extension with null byte bypass

#### Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value ../../../etc/passwd%00.png.
3. Observe that the response contains the contents of the /etc/passwd file.

## **How to prevent a directory traversal attack**

The most effective way to prevent file path traversal vulnerabilities is to avoid passing user-supplied input to filesystem APIs altogether. Many application functions that do this can be rewritten to deliver the same behavior in a safer way.

If it is considered unavoidable to pass user-supplied input to filesystem APIs, then two layers of defense should be used together to prevent attacks:

* The application should validate the user input before processing it. Ideally, the validation should compare against a whitelist of permitted values. If that isn't possible for the required functionality, then the validation should verify that the input contains only permitted content, such as purely alphanumeric characters.
* After validating the supplied input, the application should append the input to the base directory and use a platform filesystem API to canonicalize the path. It should verify that the canonicalized path starts with the expected base directory.

Below is an example of some simple Java code to validate the canonical path of a file based on user input:

File file = new File(BASE\_DIRECTORY, userInput);  
if (file.getCanonicalPath().startsWith(BASE\_DIRECTORY)) {  
    // process file