Path Traversal

Path traversal is also known as directory traversal. These vulnerabilities enable 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.
- Sensitive operating system files.

Reading arbitrary files via path traversal

Imagine a shopping application that displays images of items for sale. This might load an image using the following HTML

```vbscript-html
<img src="/loadImage?filename=218.png">
```

The `loadImage` URL takes a `filename` parameter and returns the contents of the specified file. The image files are stored on disk in the location `/var/www/images/`

```vbscript-html
/var/www/images/218.png
```

This application implements no defenses against path traversal attacks. As a result, an attacker can request the following URL to retrieve the `/etc/passwd` file from the server's filesystem:

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

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

```vbscript-html
/var/www/images/../../../etc/passwd
```

On Windows, both `../` and `..\` are valid directory traversal sequences.On Linux `../`

Common obstacles to exploiting path traversal vulnerabilities

Many applications that place user input into file paths implement defenses against path traversal attacks. These can often be bypassed.

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

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

```vbscript-html
GET /image?filename=....//....//....//etc/passwd
```

web servers may strip any directory traversal sequences before passing your input to the application. You can sometimes bypass this kind of sanitization by URL encoding, or even double URL encoding, the `../` characters. This results in `%2e%2e%2f` and `%252e%252e%252f` respectively. Various non-standard encodings, such as `..%c0%af` or `..%ef%bc%8f`, may also work.

```vbscript-html
GET /image?filename=..%252F..%252F..%252Fetc/passwd
```

An application may require the user-supplied filename to start with the expected base folder, such as `/var/www/images`. In this case, it might be possible to include the required base folder followed by suitable traversal sequences

```vbscript-html
filename=/var/www/images/../../../etc/passwd
```

An application may require the user-supplied filename to end with an expected file extension, such as `.png`. In this case, it might be possible to use a null byte to effectively terminate the file path before the required extension.

```vbscript-html
filename=../../../etc/passwd%00.png
```

How to prevent a path traversal attack

The most effective way to prevent 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.

- Validate the user input before processing it. Ideally, compare the user input with a whitelist of permitted values. If that isn't possible, verify that the input contains only permitted content, such as alphanumeric characters only.
- After validating the supplied input, append the input to the base directory and use a platform filesystem API to canonicalize the path. Verify that the canonicalized path starts with the expected base directory.

```vbscript-html
File file = new File(BASE_DIRECTORY, userInput);
if (file.getCanonicalPath().startsWith(BASE_DIRECTORY)) {
    // process file
}
```