# Null Byte Injection

**Null Byte Injection** is a vulnerability where an attacker inserts a **null byte (%00 or \0)** into user input to prematurely **terminate strings** in vulnerable programming languages (especially C, C++, PHP, etc.).

This causes the application to **interpret only part of the input**, allowing the attacker to **bypass validations**, **change logic**, or **access unintended files**.

# Why It Happens

Many **older functions or libraries** treat \0 (null byte) as the **end of a string**. When user input includes %00, the **rest of the string is ignored** during processing — but might still be included in validations.

# Common Attack Scenarios

## File Extension Bypass (File Upload)

Frontend Validation:

if (preg\_match('/\.jpg$/', $filename)) {

move\_uploaded\_file($filename, "uploads/" . $filename);

}

Attacker Input: shell.php%00.jpg  
Server-side check passes (sees .jpg),  
but move\_uploaded\_file() stores shell.php if the server interprets %00 as null.

## LFI Bypass

| Goal | Trick server into including a .php file while the app thinks it’s .txt |

include($\_GET['file'] . '.txt');

input  
file=../../../../etc/passwd%00  
Server may include /etc/passwd ignoring .txt

## Path Traversal Bypass

Some sanitization routines remove ../, but fail to account for:

%00../  
May bypass sanitizers and reach forbidden paths.

## Null Byte in Shell Commands or SQL

If a shell or SQL command uses null-byte vulnerable parsing, the attacker may manipulate input to truncate the string, causing logic to break or alter.

# Impact

| Risk | Description |
| --- | --- |
| Arbitrary File Access | Bypass extension checks → read/write restricted files |
| Upload Shell Execution | Upload .php disguised as .jpg%00 |
| Security Logic Bypass | Input truncation breaks validation or enforcement |
| LFI/RFI Enhancement | Used to bypass .php filters or auto-extension appending |
| Legacy RCE | Chained with LFI to execute logs, or in C-based parsing |

# Detection Techniques

| Method | Description |
| --- | --- |
| Fuzz Parameters with %00 | Try appending %00 before dangerous file extensions or path |
| Upload Test | Try uploading file.php%00.jpg and accessing it |
| Combine with LFI | Use %00 to truncate .php.txt logic and access real .php |
| Observe Response | Watch for difference in behavior vs. normal input |
| Source Review | Check if app uses legacy functions or unsafe file/path validation logic |

# Test Payloads

| Payload | Use |
| --- | --- |
| ../../../../etc/passwd%00 | LFI bypass |
| avatar.php%00.jpg | File upload bypass |
| id=admin%00 | SQL/Shell injection truncation |
| %00../ | Bypass path traversal sanitizers |

# Affected Languages

| Language | Vulnerable? |
| --- | --- |
| C/C++ | ✅ Yes (native null-terminated strings) |
| PHP (<=5.3) | ✅ Vulnerable (especially in file APIs) |
| Java | ❌ Safe (handles strings as objects) |
| Python | ✅ Older libs vulnerable; mostly safe now |
| Ruby | ✅ Some legacy file APIs are vulnerable |
| Node.js | ❌ Safe if using modern file handling |

# Mitigation

## 1. Use Safe Language Features

* Avoid legacy C-style string functions (e.g., strcpy, fopen)
* Use **strict typing** and **full string validation** (not just suffix check)

## 2. Input Sanitization

* Block or strip null byte characters:

%00, \0, \x00

* Use regex like:

if (preg\_match('/[\x00]/', $input)) exit("Bad input!");

## 3. Secure File Handling

* Use **strict allowlist** of file extensions:

$ext = pathinfo($filename, PATHINFO\_EXTENSION);

if (!in\_array($ext, ['jpg', 'png'])) die('Invalid extension');

Avoid using raw user input in file paths.

## 4. Upgrade Language Versions

* PHP ≥ 5.3.4 and modern frameworks handle null bytes properly
* Ensure your runtime disables legacy null byte behaviors

## 5. Logging & Alerting

* Log all requests with null byte indicators (%00)
* Alert on any unusual upload filename patterns

# Real-World Examples

| System | Vulnerability |
| --- | --- |
| Joomla, WordPress plugins | Null byte allowed .php file upload via %00.jpg |
| Old PHP forums & CMS | LFI to RCE using %00 truncation |
| CTFs and Bug Bounties | Frequent use of %00 in LFI, file upload, and path tricks |

# Points

Null Byte Injection exploits legacy systems that treat %00 as **end-of-string**, allowing **validation bypass and logic confusion**.”

“Modern apps are more resistant, but still vulnerable if using **insecure file operations or legacy libraries**.”

“It’s often **chained with LFI or file upload flaws** to escalate to RCE.