Unrestricted File Upload

What is Unrestricted File Upload Testing and how to test for **Unrestricted File Upload Vulnerabilities** including filter bypass techniques for Windows, Linux, Apache and IIS.

What is Unrestricted File Upload?

Unrestricted File Upload Testing

> Vulnerable upload functions allow attackers to bypass input controls, upload payloads and potentially perform command execution

Table of Contents [show]

Web application file upload functions that do not have the correct controls in place to ensure user uploaded files are validated or sanitised are potentially vulnerable to unrestricted file upload. This document outlines the testing process for file upload functions while performing a penetration test. This document contains various techniques to bypass File Upload Black List filtering and concludes with a helpful check list.

Identify File Upload Functions

File upload functions are both easy to identify and easy to exploit. Typical places are profile image avatars, document upload functions and file import functions. Burp passive scanner will identify file upload entry points when you're at the discovery and application mapping phase.

Looking for a manual web app penetration test? See our web application pen testing page for more details

Using Burp Intruder to Test for Unrestricted File Upload

Web applications often use black listing for file input validation or sanitisation which is normally insufficient. If a file extension is missed from the blacklist an attacker can bypassed filtering. The preferred mechanism for input validation is input white listing, which uses a denyall approach and only allows input that is required.

Burp Testing File Upload Extensions

1. Manually upload a file that will likely fail the upload validation or sanitisation test, find a response that can be used to identify the web application is rejecting the file extension.
2. Send the upload request to Burp intruder
3. Clear the default insertion points
4. Select the file extension point as the insert location
5. Select a payload containing various extensions php.jpg, asp.jpg etc
6. In options configure grep within response to contain the failed response string identified at step 1
7. Start intruder, any responses unticked for the grep string are likely vulnerable are require further inspection and Confirm any findings

Key Points:

1. Test all extensions using Burp Intruder and use the Grep feature to sort results
2. Use uncommon file extensions that may bypass the black list such as: `.php3`, `.php5`, `.phtml`

Test for File Upload Content-Type change with Burp

Identify accepted file upload Content-Type's accepted by the target.

1. Establish a baseline – use a known accepted Content-Type and monitor the applications response, repeat with a content type that is likely not accepted, use the failed response at step 6
2. Send the upload request to Burp intruder
3. Clear the default insertion points
4. Select the "Content-Type:" header as the insert location
5. Select a payload list containing Content-Types
6. In options configure grep within response to contain the failed response string identified at step 1
7. Start intruder, any responses unticked for the grep string are likely vulnerable are require further inspection
   Confirm any findings

Key Points:

1. Test all Content-Types using Burp Intruder and use the Grep feature to sort results
2. Try changing the Content-Type to one that is supported, with a extension that the web server / web app will process
3. Try uncommon Content-Types that may bypass the black list

File Name and Extension Fuzzing

The file name and extension should be tested for input validation, what happens if the file name is an XSS, SQLi, LDAP or a Command Injection payload?

1. Manually upload a file that will likely fail the upload sanitisation or validation test, find a response that can be used to identify the web application is rejecting the file extension
2. Send the upload request to Burp intruder
3. Clear the default insertion points
4. Select the file extension or file name point as the insert location
5. Select a payload containing various injection [js, XSS, CMD, LDAP, Xpath, SQL etc [payloads
6. In options configure grep within response to contain the failed response string identified at step 1
7. Start intruder, any responses unticked for the grep string are likely vulnerable
8. Confirm findings

File Upload Black List Bypass Techniques

Windows IIS Server Black List File Upload Bypass:

1. Upload a file with the semi colon after the black listed extension, such as: `shell.asp;.jpg`
2. Upload a directory with the .asp extension, then name the script within the directory with a permitted file extension, example: `folder.asp\file.txt`
3. When serving PHP via IIS `< > and .` get converted back to `? * .`
4. Use characters that can replace files, example `>>` can replace `web.config`
5. Try using spaces or dots after characters, example: `foo.asp..... .. . . .`
6. `file.asax:.jpg`
7. Attempt to disclose information in an error message by uploading a file with forbidden characters within the filename such as: `| %< * ? "`

Apache Windows Black List Bypass:

1. Windows 8.3 feature allows short names to replace existing files, example: web.config could be replaced by web~config.con or .htaccess could be replaced by HTACCE~1
2. Attempt to upload a . file, if the upload function root is `/www/uploads/` it will create a file called uploads in the directory above.

Looking for a manual consultant lead web application security test? See our [web application penetration testing] services page for more details

General Black List Bypass:

1. Identify what characters are being filtered – use burp intruder to assess the insert points with a meta character list
2. Ensure your list contains uncommon file extension types such as `.php5`,`.php3`,`.phtml`
3. Test for flaws in the protection mechanism, if it's stripping file names can this be abused? Example: `shell.p.phpp` if the app strips .php it could rename the extension back to .php
4. Try a null byte `%00` at various places within the file name, example: `shell.php%00.jpg`, `shell.php%0delete0.jpg` – observe how the application responds
5. Double extensions: if the application is stripping or renaming the extension – What happens if you give it two extensions? Example: `shell.php.php` or 4 `shell.txt.jpg.png.asp`
6. Try long file names, example `supermassivelongfileeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeename.php` apply other filter bypass techniques used in conjunction with long file names
7. Try `test.asp\`, `test.asp.\`
8. Can you upload the flash XSS payload, that is named as a .jpg
9. Try the previous technique but use PDF or Silverlight instead
10. Same again but attempt to abuse crossdomain.xml or clientaccesspolicy.xml files
11. Try using encoding to bypass blacklist filters, try URL, HTML, Unicode and double encoding
12. Combine all of the above bypass techniques
13. Try using an alternative HTTP Verb, try using POST instead of PUT or GET (or vice versa), you can enumerate the options using Burp Intruder and the HTTP Verbs payload list
14. Additionally, ensure all input points are fuzzed for various input validation failures such as, XSS, Command Injection, XPath, SQLi, LDAPi, SSJI

Bypassing File Upload White Listing

1. Enumerate whitelist and test for any permitted file extensions that can be abused
2. Test files contents are being validated
3. Test that extension renaming techniques such as `shell.php.png` are not possible

---

Bypass File Size Upload Mechanisms

Malicious File Contents

Assess any file upload contents is correctly sanitised by the application. For example, can you inject XSS into an Excel, CSV or txt files that will later be rendered by the application? Use Burp repeater and intruder to attempt injecting various payloads within file import and upload functionality, assess the applications response.

EXIF Image Data

Can a reverse shell be injected within image EXIF data ?

Install on Kali:

```
apt-get install exiftool
```

After injecting the code within the image file, simply upload the file and verify the file is the same using a checksum (details below).

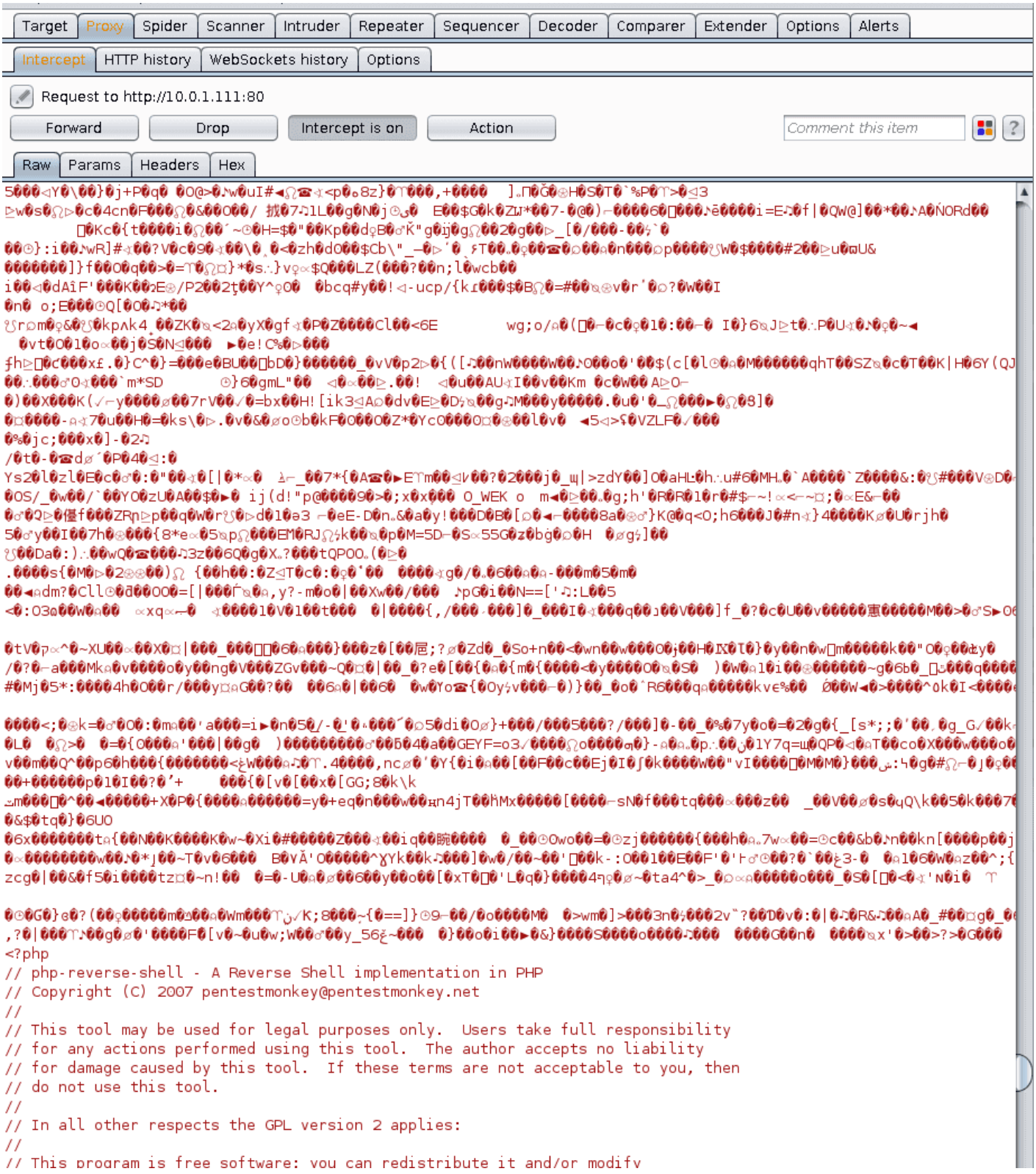
Looking for a manual consultant lead mobile application security test? See our mobile application penetration testing services page for more details

---

Injecting into the request with Burp

This is a similar technique to the EXIF method above, however you paste the code directly into the burp request.

1. Upload a legitimate image using burp and verify upload is successful
2. Send the previous request to burp repeater
3. After the legitimate image data in the request, attempt to inject payloads (injection payloads or a reverse shell)
4. Submit the request
5. Download the uploaded file from the target server, verify it has the contained payload within, can you leverage this to execute a payload on the target server?



Verify the file is Uploaded

Regardless of the method you used in the previous step (EXIF or Modifying the Burp Request) create a checksum of the file locally and upload to the target, download the file from the target and verify the checksum matches.

Key Points:

1. Checksum a file, upload it to the app, download it from the app, verify it's the same file

Techniques for Server Side Command Execution

If it's possible to successfully upload a shell to the target web application you can attempt some of the following techniques to execute the uploaded shell.

Has your organisation performed an External Pen Test recently? See our Network Penetration Testing services page for more details

1. Apache MIME Types: Attempt to upload a renamed file e.g. `shell.php.jpg` or `shell.asp;.jpg` and assess if the web server process the file by exploiting weak Apache MIME types
2. Null Byte: Try a null byte `%00` at the end of the file name or within such as: `shell.php%0delete0.jpg`– observe how the application responds
3. Can you upload dot files, if so can you upload a .htaccess file an abuse AddType: `AddType application/x-httpd-php .foo`
4. Be mindful of any processing to upload files – Example: Could command injection be used within a file name that will later be processed by a backend backup script?
5. Be mindful of any server side processing to upload files – If compressed files are permitted, does the application extract them or vice versa?
6. Does the server Anti Virus process uploaded files? – Try uploading a compressed file type such as .zip, .rar etc if the server side AV is vulnerable, it's possible to exploit and gain command execution.

Unauthenticated File Upload Testing

Unauthenticated file upload, allows an attacker to DoS a target by fill disk space on target machine.

Testing for Arbitrary File Upload using Burp:

- Identify file upload function
- Perform a normal file upload using an authenticated user (if possible)
- Send the request to burp comparer
- Remove the cookie or session identifier from the request
- View the response to assess if file upload is possible without authentication

---

Has your organisation performed a Vulnerability Assessment recently? See our Vulnerability Testing services page for more details

Testing for DoS Condition Disk Filling

If within testing scope, assess if DoS is possible via file upload or disk filling from a single session. Use a low number (~100) of jpg files and use Burp intruder Number payload option to increment the payload names, e.g. image1.jpg, image2.jg, image4.jpg etc.

Large File Upload

Upload a large file and assess if the application allows the upload.

If shell access is available on the test, it's easier to perform a server side assessment checking for `LimitRequestBody` within the Apache config and `MAX_FILE_SIZE` within php.ini

---

Test for Server Side Antivirus Scanning

Use an EICAR file, a benign test file for testing AV detection and verify it gets detected by AV scanners on VirusTotal.

- Checksum the EICAR file

- Perform a normal file upload
- Download the EICAR file
- Checksum the EICAR file and validate it's the same file

Repeat the same test 24 hours later and assess if any daily antivirus filtering is taking place.

CSV Macro Injection

If the application has an export function assess if it's possible to injection macros within the web application that could be executed client side by another user.

Example:

- Attacker injects malicious payload within the web application
- Administrator logs in and exports the web application data to CSV
- Attackers injected payload is then executed client side by the victims Excel, it's likely even if excel prompts or warns the victim will proceed as the exported data is from a site they trust.

Key Points:

1. Upload a malicious payload and access if it's possible to download the payload from a back end interface.
2. Verify the payload is the same file using a checksum

File Upload Testing Check List

Discovery:

1. ☐ Identify File Upload Points

Windows IIS Server Black List File Upload Bypass:

1. ☐ Upload a file with the semi colon after the black listed extension, such as: `shell.asp;.jpg`
2. ☐ Upload a directory with the .asp extension, then name the script within the directory with a permitted file extension, example: `folder.asp\file.txt`
3. ☐ When serving PHP via IIS `> < and .` get converted back to `? * .`
4. ☐ Use characters that can replace files, example `<<` can replace `web.config`
5. ☐ Try using spaces or dots after characters, example: `foo.asp..... .. . . .`
6. ☐ `file.asax:.jpg`
7. ☐ Attempt to disclose information in an error message by uploading a file with forbidden characters within the filename such as: `| > < * ?"`

Apache Windows Black List Bypass:

1. ☐ Windows 8.3 feature allows short names to replace existing files, example: web.config could be replaced by web~config.con or .htaccess could be replaced by HTACCE~1
2. ☐ Attempt to upload a . file, if the upload function root is `/www/uploads/` it will create a file called uploads in the directory above.

General Black List Bypass:

1. ☐ Identify what characters are being filtered – use burp intruder to assess the insert points with a meta character list
2. ☐ Ensure your list contains uncommon file extension types such as `.php5`,`.php3`,`.phtml`

3. ☐ Test for flaws in the protection mechanism, if it's stripping file names can this be abused? Example: `shell.p.phpp` if the app strips .php it could rename the extension back to .php
4. ☐ Try a null byte `%00` at various places within the file name, example: `shell.php%00.jpg`, `shell.php%0delete0.jpg` – observe how the application responds
5. ☐ Double extensions: if the application is stripping or renaming the extension – What happens if you give it two extensions? Example: `shell.php.php` or 4 `shell.txt.jpg.png.asp`
6. ☐ Try long file names, example, `supermassivelongfileeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeename.php` apply other filter bypass techniques used in conjunction with long file names
7. ☐ Try `test.asp\`, `test.asp.\`
8. ☐ Can you upload the flash XSS payload, that is named as a .jpg
9. ☐ Try the previous technique but use PDF or Silverlight instead
10. ☐ Same again but attempt to abuse crossdomain.xml or clientaccesspolicy.xml files
11. ☐ Try using encoding to bypass blacklist filters, try URL, HTML, Unicode and double encoding
12. ☐ Combine all of the above bypass techniques
13. ☐ Try using an alternative HTTP Verb, try using POST instead of PUT or GET (or vice versa), you can enumerate the options using Burp Intruder and the HTTP Verbs payload list
14. ☐ Additionally, ensure all input points are fuzzed for various input validation failures such as, XSS, Command Injection, XPath, SQLi, LDAPi, SSJI

Bypassing File Size Upload Checks:

1. ☐ Use EXIF image file technique
2. ☐ Inject shell directly after image data within Burp request
3. ☐ Verify uploaded file is the same using checksums

Techniques for Executing Uploaded Shells:

1. ☐ Apache MIME Types: Attempt to upload a renamed file e.g. `shell.php.jpg` or `shell.asp;.jpg` and assess if the web server process the file by exploiting weak Apache MIME types
2. ☐ Null Byte: Try a null byte `%00` at the end of the file name or within such as: `shell.php%0delete0.jpg`– observe how the application responds
3. ☐ Can you upload dot files, if so can you upload a .htaccess file an abuse AddType: `AddType application/x-httpd-php .foo`
4. ☐ Be mindful of any processing to upload files – Example: Could command injection be used within a file name that will later be processed by a backend backup script?
5. ☐ Be mindful of any processing to upload files – If compressed files are permitted, does the application extract them or vice versa?

File Upload DoS Testing:

1. ☐ Unauthenticated File Upload Testing
2. ☐ Testing for DoS condition Disk Filling
3. ☐ Large File Upload

Test for Server Side Anti-Virus:

1. ☐ Checksum the EICAR file
2. ☐ Upload & download the file, verify the checksum matches original
3. ☐ Repeat test in 24 hours

FAQ

CSV Macro Injection:
What is an Unrestricted File Upload Vulnerability?

1. ☐ Test for CSV macro Injection

A local file upload vulnerability where an application fails to verify the contents of an uploaded file, allows an attacker to upload a malicious file to the web server or application. Malicious files such as reverse shells, malware or viruses could potentially be executed or stored / served by the vulnerable application.

How to prevent Unrestricted File Upload Vulnerabilities

- **Do Not Store Uploaded Files Within The File System:** Avoid storing files directly within the webserver file system, to avoid execution consider storing uploaded files within a database or segregate the uploaded files on another server.
- **Sanitise File Names:** rename sanitise and encode uploaded file names before storing or reflecting within the web application.
- **Verify Whitelisted File Types:** Ensure uploaded fille types are whitelisted and the whitelist of approved file types is validated to avoid malicious attacks to spoof or evade filtering.
- **Server-side Anti-Virus:** Ensure server-side anti-virus is in place and configured to scan uploaded files.

What are the Risks from Unrestricted File Upload

The potential risks of an **unrestricted file upload vulnerability** depends on the level of exploitation reached. Typically, successful exploitation of a file upload vulnerability results in a compromise the target host which could, given the correct set of circumstances result in an adversary uploading malicious payload to the server such as a reverse shell and successfully gaining shell level access to the server; potentially exposing sensitive/personal data which could be modified or deleted. If an attacker were to gain shell level access to the server this could be used as a potential point to launch a lateral movement and penetrate deeper into the network. Other attacks such as DoS attack, or using the server to store and distribute viruses or malware could also be executed. Typically, it depends on how the target application handles the uploaded files, and how well the uploaded files are restricted from the rest of the network and what controls exist to prevent malicious files from being uploaded, and/or executed.

[Total: 5 Average: 4.6/5]

Leave a Comment

Your email address will not be published. Required fields are marked *

Comment
Name *
Email *
Website

By using this form you agree with the storage and handling of your data by this website. *

Submit

Recent Posts

- Lateral Movement Explained
- What is VAPT
- What is Directory Traversal
- Does SameSite Provide Sufficient CSRF Defence?
- What is XSS (Cross-site Scripting)?
- Local File Inclusion (LFI)
- LLMNR / NBT-NS Spoofing Attack
- SSL & TLS HTTPS Testing
- Unrestricted File Upload Testing