

# 0. Introduction

This document is a quick guide to performing web application security audits.

It lists the frequent steps that should be performed in an audit to try to remove as many vulnerabilities as possible.

Its purpose is to serve as an initiation guide to the beginner and as a reminder to the expert.

Based, among others, on the book "A Web Application Hacker's Handbook" (Dafydd Stuttard and Marcus Pinto).

Created and elaborated by Carlos Javier González Cortés.

[www.hackinglethani.com](http://www.hackinglethani.com)

[www.linkedin.com/in/cj-gonzalez](https://www.linkedin.com/in/cj-gonzalez)

[VERSION 1.0]

Translated with [www.DeepL.com/Translator](http://www.DeepL.com/Translator)

# 1. Application Mapping

In this section the ideal thing is that you gather information, make lists of interesting pages, write yourself notes, etc.

It is recommended that you wait until you have all the information before you start exploiting it.

# Explore visible content. Burp

- ☐ Configure Burp and proxy..
- ☐ Visit each link of the web application. View with JavaScript enabled and disabled. View cookies. Do it in several browsers.
- ☐ Register or login (if you have credentials).
- ☐ Look in the passive spider if it has found some page where you have not been. See "Linked From" for details.
- ☐ Use the active spider, with the pages you've been discovering as seeds. Remove dangerous pages that could cause you to log out, or that could dirty the server database.

~~~~~

Notes: The idea in this section is that you get a list with the different pages that the application has.

# Consult public resources

- ☐ Searches the Internet for indexed or saved content. .
- ☐ Use advanced Google options (site:\_\_\_\_\_, quotation marks, etc.)  
>>> As of 03/2019 you can find a google dorks guide at [www.hackinglethani.com/google-dorks](http://www.hackinglethani.com/google-dorks)
- ☐ Search for names and emails you find. Search technical forums.
- ☐ Search for WSDL docs to generate possible function names and parameters used.  
>>> It is a descriptor file that gives useful information about the implemented functionalities.

~~~~~

## Notes:

On this page you can find the different advanced Google options:

<https://medium.com/@neelsnow/the-growth-hackers-guide-to-google-dorks-ed303cc672d9>

# Discover hidden content

- ❑ See how the app responds to requests for non-existent content. Compare responses.  
>>> Sometimes in these error messages you can find XSS or suggestions from other pages of the application.
- ❑ Get a list of common files and directory names.  
>>> Use logic: if there is an AddDocument.jsp and a ViewDocument.jsp, there is probably a RemoveDocument.jsp
- ❑ View HTML comments and form elements disabled on the client side, for server-side hidden content.  
>>> If there are commented areas, try to decomment them, it can give you access to hidden parts of the application.

~~~~~

Notes:  
In this section you should add hidden pages and files, and those pages that cannot be accessed from a link.

# Discover default content. Nikto, Dirb

- Use **NIKTO** to detect default or known content.

>>> Warning: very slow. It can be dangerous to use it given the large number of requests it makes, make sure you have permission to use it.

- Use **DIRB** to detect default or known content.

>>> Depending on the type of web page, you may need to add some flag, such as -w if the page responds to all requests.

- Check manually to avoid false positives. Try the IP, not just the hostname.

>>> Use shodan to check if there are hidden ports or relevant information in the headers.

- Make requests to the root directory of the server, using different User-Agent.

>>> If the domain has a WAF, some tools you use may not work without specifying the user agent.

~~~~~

Notes:

You can find different User-Agent in:

<http://www.useragentstring.com/pages/useragentstring.php>

# Enumerate functions with identifier

- ❑ Identifies pages where you pass parameters (such as /main.php?func=A21)
  - >>> It is possible that you can perform SQL injection or other injection attacks by modifying these parameters.
- ❑ Apply discovery techniques. See if there are function names in the parameters, and see if that can tell you when a function exists and when it does not.

~~~~~

Notes:

# Debug parameter testing

- ☐ The idea is to try to find a page where there are hidden debug parameters (like debug=true)  
>>> These parameters are used during the development phase, and may be forgotten when going into production there.
- ☐ Use common debug names (debug, test, hide, source) and common values (true, yes, on, 1).  
>>> In POST put it in the URL string and in the body of the request. Use **Cluster Bomb Attack** in **Burp Intruder**.
- ☐ Check if the app reacts differently when making any of these requests.

~~~~~

Notes:



## 2. Application Analysis

In this section you will analyze the application itself (functionality, technology, etc). You'll be able to point out possible vulnerabilities, but you still have to wait to exploit them.

# Identify functionality

- ☐ Identifies what the app was created for and what each function is used for.
- ☐ Identifies the application's security mechanisms and how they work.
  - >>> Write down any information you think might be useful when you try to skip them.
- ☐ Identify error messages, redirects, off-site links, non-standard things...
  - >>> (HTTP errors such as 404 or 500).

~~~~~

Notes:

In the error messages you should check if any information is given that should not be displayed (the exception of a function, information about the server version, etc.).  
You should also check if the error shows what you entered, or if it shows up differently when you enter special characters.

# Identify technologies used. SSL

- Identifies each different technology used on the client side (forms, scripts, cookies, Java Applets, ActiveX controls and Flash objects)
  - >>> A good addon to get this information is Wappalyzer (<https://www.wappalyzer.com/>)
- Researches technologies used on the server side (scripting languages, app platforms, databases and email systems that interact with the back-end).
- Look at the "server" header returned. Test in different areas of the application.
  - >>> If it gives information about the technology used by the server, it is a vulnerability.
- Use **HTTPrint** tool to fingerprint
- Use **TestSSL** to check for TLS/SSL vulnerabilities (<https://github.com/drwetter/testssl.sh>)
  - >>> Also check with Qualys <https://www.ssllabs.com/ssltest/> (Important click on 'Do not show the results on the boards')
  - >>> Check insecure renegotiation of the TLS protocol:
    - \$ openssl s\_client -connect www.pagina.com:443
    - R + enter
  - >>> Others:
    - \$ sslscan 1.2.3.4:443
    - \$ sslyze sslyze --regular 1.2.3.4:443
    - \$ nmap -sV --script ssl-enum-ciphers -p 443 <host>
    - \$ nmap -sV --script ssl-enum-ciphers, ssl-dh-params -p 443 <host>
  - >>> BREACH vulnerability test:
    - \$ ncat --ssl code.iadb.org 443H
  - >>> More info: [https://www.owasp.org/index.php/Testing\\_for\\_Weak\\_SSL/TLS\\_Ciphers,\\_Insufficient\\_Transport\\_Layer\\_Protection\\_\(OTG-CRYPST-001\)](https://www.owasp.org/index.php/Testing_for_Weak_SSL/TLS_Ciphers,_Insufficient_Transport_Layer_Protection_(OTG-CRYPST-001))
- With the obtained in the mapping, look at the extensions to deduce technologies used.
  - >>> For example, if there are pages that end in .aspx, use ASP.NET
- Identify script names and parameters that indicate there may be third-party code.
  - >>> Use Google's wildcard "inurl:\_\_\_\_\_"

~~~~~

## Notes:

Some interesting tools for this part:

- nslookup <dominio> → Server domain name, address
- dnsenum <dominio> → Host addresses, name servers, mail servers, zone transfer
- wafw00f <dominio> → Tells us if there is any antivirus or firewall behind the page

Interesting browser plug-ins:

- Advanced cookie manager
- Firebug
- Live HTTP headers
- PassiveRecon → Lots of info! DNS, SOA, MX, Architecture, Dominios, pages sharing domain, google docs...
- IP Address and Domain Information → IP, localization, DNS server, geolocation

Wappalyzer → tells you if it has apache, email, javascript, and headers info.

# Test data transmission via client

- ❑ Locates instances where hidden fields, cookies, or URL parameters are used to transmit data.  
>>> To do this, it intercepts requests with Burp.
- ❑ Modifies the value of the item according to its role. Look when it processes it arbitrarily and when it carries out some control.  
>>> If, for example, the website only lets you write numbers in a telephone form, you may be able to enter other characters when intercepting the request.
- ❑ Check if you send opaque, obfuscated or encrypted customer data for later use or to remove them.  
>>> If, due to the format, they seem to have done the obfuscation by hand (without using any standard) or have used some weak hash, it is possible that it is insecure and can be broken.
- ❑ Check if you are using ASP.Net ViewState. See if it contains info, if it can be modified (with Burp). See if you can change another parameter and re-encode by sending the ViewState (if it accepts you can enter arbitrary data in the processing of the application).

~~~~~

Notes:  
A good page to check which type of hash is a certain CyberChef. This page offers many options for cryptography, steganography, and conversion.  
<https://gchq.github.io/CyberChef/>

# Map the attack surface. Nmap

- Understands the internal structure and functionality of the server side, and how to show it in client.

>>> For example, a function for placing orders is how to interact with the database

- Launch nmap

>>> The following is a nmap's cheatsheet:

>: nmap -sT <ip> -v → open ports, tcp, state and service

>: nmap -sS <ip> -v → same as above, but stealthy

>: nmap -sF <ip> -v → uses FIN packages instead of SYN. Normally not very useful.

>: nmap -sX <ip> -v → Christmas package. Very noisy.

>: nmap -sN <ip> -v → Null scan. Tricks a security server into generating a response without setting a bit.

>: nmap -sP <ip> -v → Ping scan. It tells us which computers are active on the network.

>: nmap -sU <ip> -v → UDP open ports. Slow and noisy.

>: nmap -Pn -sV --version-intensity 5 <ip> → common ports and services

>: nmap -sS -p- <ip> -v -f → All ports.

>: nmap -sS <ip> -v --mtu8 → to split packets (and jump firewall). also mtu16, mtu32, etc.

>: nmap -sS -sV <ip> -v → to know the version of services.

>: nmap -sS -sV -O <ip> -v → to scan ALL ports.

>: nmap -A -T4 <ip> -v -p1-65535 → very powerful. Extremely slow.

>: nmap -sS -sV --script default <ip> -v → scripts that enhance information and vulnerabilities.

>: nmap -sS -sV --script safe <ip> -v → scripts that enhance information and vulnerabilities.

>: nmap -sS -sV --script vuln <ip> -v → scripts that enhance information and vulnerabilities.

>: nmap -sC <ip> -v → launches all scripts.

>: nmap -sV -O dominio.com → operating system.

>>> With -oA <path> you can save it in xml, nmap and gnmap format.

>>> The scripts are in /usr/share/nmap

>>> Another interesting scanner is amap:

\$ amap -d <ip> <puerto>

- For each item of functionality, identify the types of common vulnerabilities associated with it.

>>> For example, file upload functions are vulnerable to Path Traversal, messages between users to XSS, and "contact us" functions to SMTP injection.

- Make a plan of attack, prioritizing the most interesting and dangerous.

>>> Go for critical vulnerabilities!

~~~~~

Notes:

- Heartbleed

\$ nmap -p 443 --script ssl-heartbleed --script-args vulns.showall example.com

- Ticketbleed

\$ nmap -p 443 --script tls-ticketleed --script-args vulns.showall example.com

### **3. Test browser extensions (APPLET) / mobile Apps.**

Just in case you're analyzing an applet.

# Decompiling the customer

☐ Review all calls made to the applet methods. Determine which data returned by the applet is sent to the server. If they are opaque, it will have to be decompiled to obtain the source code.

☐ Download the applet code by putting the URL in the browser and saving the file locally. The name of the file is specified in the tag of the applet "code", and it will be in the path "codebase" if the tag is there, if not it will be in the same directory.

☐ Decompile the code with

- > Jad -> Java
- > SWFScan, Flasm/Flare -> Flash
- > .NET Reflector -> Silverlight
- > archivos .jar, .xap o .swf -> WinRAR

Extension | Technology

- .dex --> .jar
- .ipa --> iPhone/iPad
- .class .jar --> Java
- .swf --> Flash
- .xap --> Silverlight

☐ Understand the code. See if it has public methods that can be used, to unfuse or input.

☐ Modify the code to avoid validation. Recompile

☐ If the decompiling is not obfuscated, it is a vulnerability.

~~~~~

Notes:

You can use d2j-dex2jar to decompile and analyze .jar, as well as view classes with jd-gui.  
Flash files can be analyzed with SWFScan.



# Debugging

- Monitor all traffic between client and server.
  - >>> If there is serialized data, use **Burp's built-in AMF** or **DSer Burp** for java.
- If the app is too big, use JavaSnoop for Java, Silverlight Spy for Silverlight.

~~~~~

Notes:

# Test controls. ActiveX

□ Identify .cab or tags <OBJECT>. Validations can be eliminated by modifying the data processed with a debugger or by altering the program execution path.

>>> Also use COMRaider, Filemon and Regmon. Look for ActiveX vulnerabilities or dangerous methods such as "LaunchExe".

~~~~~

Notas: Only if there is ActiveX

# Mobile Apps. Android and Iphone.

- ☐ On mobile devices, check whether you have a valid certificate to prevent man-in-the-middle (sslpinning) attacks.
- ☐ Check if there are automatically stored screenshots, and if you can take screenshots with your phone in sensitive areas.
- ☐ Check if the application detects that the phone is rotated.
- ☐ Check cryptography and security with IntroSpy.
  - >>> Check the following fields in IntroSpy > Settings: Logs, File System, User preferences, KeyChain, Common crypto, Security Framework, HTTP, Pasteboard, URL Schemes and XML.
- ☐ Checks for dangerous functions.
  - > malloc
  - > free
  - > strcpy
  - > strcat
  - > strncat
  - > strncpy
  - > sprintf
  - > vsprintf
  - > gets
- ☐ Android.
  - >>> Pay special attention to sqlite databases.
  - >>> Check with IntroSpy the information dumped to logcat, the debug console: `$ adb logcat |grep pid_del_introspy > introspy.txt`
  - >>> Look for the "debuggable" flag.
  - >>> Check configuration errors with Drozer:
    - > Get information from the application with `$run app.package.info -a <app>`
    - > Identify if it is vulnerable to any of the preconfigured attacks with `$run app.package.attacksurface <app>`
  - > You can find more information and ways to use the tool at <https://labs.mwrinfosecurity.com/tools/drozer/>
- ☐ Iphone.
  - >>> Attention to activated autocompletion, with which new words are stored in dictionary.
  - >>> Checks binary protection: PIE flag for ASLR, cryptid binary, Stack Smashing Protection (Stack Overflow vulnerability), Automatic Reference Counting (memory corruption vulnerability).
  - >>> Look for the flag "debuggable".
  - >>> Recovers the database in iOS and generates the report with IntroSpy.

~~~~~

## Notes:

You can also launch automatic tools such as ninjadroid or MobSF. The problem with these tools is the amount of false positives they offer, but you can get interesting information by checking the reports they generate.

## 4. Authentication Tests

In applications that have a private part, a critical point is the authentication system.

In this section we look for vulnerabilities of the same, to be able to access without privileges.

# Test the quality of passwords

- See all pages that have credential sending functions (login, registration, pwd, etc.). Identify control cases on client side (length limit, Javascript checks).
- See what validations are done, and check that they are done on the server as well.
  - >>> Intercept the requests and try to skip those restrictions. If you skip them, it's a vulnerability.
- Try several types of passwords (short, just letters, no capitals and minus, dictionary words, user name).
- See if there are any disabled elements in forms. If you find them, send them to the server along with the other parameters of the form.
  - >>> This task can be automated with **Burp's** "HTML Modification".

~~~~~

Notes:

# Test user enumeration

- Identify sites where a user is introduced (forms, hidden forms, cookies...).
  - >>> Usually they are the login, registration, password change, logout and password recovery.
- For each location make two requests, one with a valid user and one with an invalid one. See if they give different answers.
  - >>> Check details: HTTP status code, redirections, HTML differences, server response time.
- If you can register, try registering 2 times with the same username.
  - >>> If you block the second attempt, you already have user enumeration.
  - >>> If not, try which password works. See what happens when you change your password.
- Before making an automated attack, consider the detected blocking mechanisms. If you're trying to figure out passwords, do it in width, not depth.

~~~~~

Notes:

# Test brute force

- Identify sites where passwords are entered.
- Test with valid user and false passwords. Make 10 failed logins. If the account is not blocked, send a request with valid credentials. If you leave, there is an insecure account blocking policy.  
>>> WARNING! If you only have one user, this should be the last test to perform. If the user is blocked, you will not be able to continue auditing.
- If you don't know users, list or guess one and try many passwords to see if it is blocked.  
>>> If the app is in production, the customer may not find it very funny that you block their users.

~~~~~

## Notes:

Also try default passwords. In distributions like Kali or Parrot there are several dictionaries with common passwords in /usr/share/wordlists

# Test second authentication factor

- ☐ Check that it is not possible to skip the 2nd authentication factor and that old or repeated values cannot be used.
- ☐ Check that the token obtained in the second factor is associated to the operation from which it is requested.
- ☐ Verify that a token of another user cannot be used.
- ☐ Checks that the 2nd authentication factor is blocked after several failed attempts.

~~~~~

Notes:



# Test account recovery functions

- ☐ Search for "I forgot my password". See how it does the recovery process. See if users can choose a recovery challenge during logging. If they can, register several times to get a list of challenges, and try to attack the one that seems easiest.
- ☐ If the app sends an email for recovery, look for vulnerabilities. Try to recover with an account you own. If you send a single recovery URL, repeat several times and try to pull a pattern.

~~~~~

Notes:

# Test "Remember Me" features.

- ☐ See what persistent cookies are created when Remember Me is turned on. Try users with similar names to see if reverse engineering is possible.

~~~~~

Notes:

# Test how predictable self-generated credentials are

- ☐ If the app generates automatic users or passwords, it creates several in search of patterns.
- ☐ If they are predictable, get possible valid users/pwd.

~~~~~

Notes:

# Check transmissions of unsafe credentials

- ☐ If they are transmitted through the URL, it is vulnerable to " disclosure " in the history, in the screen, in logs and in the header "Referencer".
  - >>> If they are in a cookie, it is vulnerable if an XSS attack is made (or if they are sent from the server to the client). Write it down for later.
- ☐ If credentials are sent via HTTPs but login via HTTP, it is vulnerable to a man in the middle.
  - >>> It is clear transmission of sensitive information!
- ☐ If the app sends an activation URL sent by other means, register several times and check patterns.
- ☐ Try to reuse a single activation URL several times. If it doesn't work, try blocking the account and reusing it.

~~~~~

Notes:

# Test hashes

- ☐ If you have hashed passwords, look for users with the same hash. Try common passwords for the most repeated value.
- ☐ Try to recover the password using an offline rainbow table.
  - >>> Use Cyberchef to see what kind of hash it is.

~~~~~

Notes:

# Test logical faults

If the implementation is not good, it is possible to obtain security flaws from logical flaws.

# Test Fail-Open conditions

□ For each function that checks credentials, repeat many times modifying each parameter in unexpected ways.

>>> Unexpected ways:

- empty string
- delete by name/value
- set very long and very short values
- set same parameter several times, with the same values and with different values

~~~~~

Notes:

# Test multi-step mechanisms

- Identify multi-step mechanisms, look at the purpose of each stage and what parameters are sent.
- Repeat many times, modifying each parameter in unexpected ways.
  - >>> Unexpected ways:
    - Make each step in a different order
    - Start in half and follow normal flow
    - Repeat skipping a different stage each time

~~~~~

Notes:  
If something changes in the answer, keep it in mind and combine it with the other things you find.



## 5. Session Management Tests

If the application does not make a correct session management, it is possible to use it to make privilege escalation or impersonation.

# Understanding the mechanism

□ If the application uses session tokens, check what data is used to identify users. To do this, go to a page in the private part of the application and try to remove data until you remove the token and it won't let you in. Maybe the app uses a lot of session tokens, but when it comes down to it, just check one.

□ See where the tokens are validated, and modify them to see if there are parts of the token that are not checked.

>>> Also try changing only 1 byte at a time.

~~~~~

Notes:

# Test the meaning of tokens

□ Take several tokens from different users. If you can, register with similar names like "A, AA, AAA, AAAA, AAAB, AAAC, AAAC, AABA..." If you need other data such as email, do the same.

>>> Can be done automated with PadBuster

□ Detects token encoding or obfuscation. See if there is a relationship between the length of the name and the length of the token.

>>> If there is a sequence of a character in the token that corresponds to a sequence of characters in the username, then it is an XOR ofuscation. Search hexadecimal, characters, Base64, etc.

~~~~~

Notes:

# Test the predictability of tokens

☐ Generate and capture a large number of session tokens in succession, making requests to the server to give new tokens. Identify patterns between tokens.

>>> Use Burp Sequencer to view statistics. Try decoding too if it's not clear what that token is.

☐ Get the same amount of tokens after 5 minutes, and repeat, to see if they are time-dependent. If you identify patterns, do it again with another IP and another user. Check to see if the same pattern is detected.

>>> If you can exploit it, it is enough to make an attack with which you supplant another user's token.

☐ If the session ID looks custom, use Burp Intruder's payload "bit flipper" to modify each bit in the session token sequentially. Look for a string in the response that indicates that by modifying the token you have logged in as another user.

~~~~~

Notes:

# Test insecure token transmission

- See what goes in HTTP and what in HTTPS. If cookies are used as a session token mechanism, look when the security flag that prevents them from being transmitted over HTTP is activated.
  - >>> See if anywhere the session token goes through HTTP. If so, it is exposed to interception.
- If in non-authenticated zones you use HTTP and then pass to HTTPS for login and private area, look which HTTP tokens pass to HTTPS, if any pass, it is possible to intercept it.
- If the HTTPS part has URLs to HTTP, see if the tokens are sent.
  - >>> If so, check if they are still valid or not.

~~~~~

Notes:

# Test token disclosure in logs

- ☐ If you detect any logs, see if he saves the session tokens. See who can access those features.
- ☐ Look at sites where session tokens go by URL in the Referer header.
- ☐ If you get a method to get tokens out of a user, try even the admin.

~~~~~

Notes:

# Check the mapping of tokens to sessions

☐ Log in twice with the same account, both with different browsers and with different computers. Check if both are active. If they continue, it's a vulnerability.

☐ Log in and out several times with the same account, both with different browsers and with different computers. See if you create a new token each time, or follow it.

>>> If it's the same, the app doesn't use session tokens well, so it has no way to protect itself from concurrent logins or session timeouts.

☐ If the tokens seem to have some meaning, separate the part that identifies the user from the rest, and try to get other users in.

~~~~~

Notes:

# Test the end of session

□ See if the session expires: log in and get a token. Wait a while without using the token. Use the token by going somewhere private. If it works, the token is still active. Try until you see how long it takes for the token to deactivate.

>>> It can be automated by increasing the Burp Intruder request interval

□ Checks if there is a logout function. if there is, tests if it really invalidates the session on the server using the old token to enter the private area. If the session continues, it is vulnerable to session hijacking attacks.

~~~~~

Notes:



# Test the fixation session

- ☐ If the app uses tokens for unregistered users, login and check if your token is the same. If it is, the app is vulnerable to session fixation.
- ☐ It is also vulnerable if you don't give tokens to non-authenticated users, you go to the login page and if you can log in even if you've already logged in, you log in again with another user, and the token hasn't changed you.
- ☐ Modify the token by an invented value (maintaining the structure), and try to login. If it allows you, it is vulnerable to session fixation.
- ☐ If you don't support login, but you process sensitive data, try the 3 previous tests in the area where there are them (my payments or whatever).

~~~~~

Notes:

# Test CSRF

□ Review the key features of the app and look at the specific requests used to create an active session. If it has no session tokens, unpredictable data or other, it will be vulnerable.

□ Create an HTML that requests the desired request without user interaction. For GET requests you can put a <img> in which the src is the vulnerable URL. For POST requests, you can create a form with hidden fields for all the parameters necessary for the attack, using js to auto-send the form as soon as the page loads. While logging in, use the same browser to load the HTML. Verify that the desired action is taken in the app.

□ If the app uses additional tokens with requests to avoid CSRF attacks, test its robustness as with session tokens. It also proves that the app is vulnerable to UI redress.

>>> To test clickjacking:

```
<html>
  <head>
    <title>Clickjack test page</title>
  </head>
  <body>
    <p>Test clickjacking</p>
    <iframe src="http://web.com" width="500" height="500"></iframe>
  </body>
</html>
```

~~~~~

Notes:

You can do this automatically with Burp, by clicking on the request, clicking on "Engagement tools" > "Generate CSRF PoC"

## 6. Access Control Tests

# Understand access control requirements

- Ideally, you should have several accounts with different privileges. This can only be done if you manage to scale privileges or if the domain owner gives you an admin user in addition to the normal use.

~~~~~

Notes:

# Test with multiple accounts

- ❑ On pages with vertical privileges (different user levels can access different information), first look with the admin which zones he has access to. Then do it with a less privileged user and see what changes. In Burp it can be done with "compare site maps".
- ❑ On pages with horizontal privileges (users at the same level have access to different parts) do the equivalent test using two different accounts at different levels.
- ❑ Do manual checks of the access control logic. Repeat requests with unauthorized users.
- ❑ Carefully check multi-step mechanisms.

~~~~~

Notes:

# Test limited access

- ☐ If you don't have access to privileged accounts, it will be more difficult to know the necessary URLs, identifiers and parameters.
- ☐ If you discovered any in the mapping, you can test them out there.
- ☐ Try to get identifiers from other users with different privileges. If there is no way, you will have to analyze only the part your user can access.
- ☐ If you find a way to predict other users' ids, make an automatic attack to get interesting data. You can filter the relevant information with Burp's "Extract Grep".

~~~~~

Notes:

# Tests for insecure access methods

- ☐ Some pages restrict access by parameters. Look for `edit=false` or `access=read` in some request and modify it to see if it works.
- ☐ Some do it in HTTP Referer, try doing something you have permissions to do and change the Referer. If it blocks the request, it is that it uses the header to do the check. Try to do something you don't have permission to do by modifying the Referer, see if it lets you.
- ☐ If HEAD is a docked method on the site, try making requests using HEAD in the URLs.

~~~~~

Notes:

## 7. Vulnerability tests based on input

Here you can see the different vulnerabilities that can have the parameters entered by the user.



# Fuzzing all request parameters

□ The objectives of fuzzing are URL parameters, request body parameters, HTTP cookies and headers such as Referer or User-Agent.

□ Burp Intruder can be used for fuzzing.

>>> Select a list of payloads that you want to pass, depending on whether you want to do general or specific fuzzing.

>>> Examples of payloads for fuzzing according to type:

> SQL Injection

```
'  
'--  
' ; waitfor delay '0:10:0' -- a  
1; waitfor delay '0:10:0' -- a
```

> XSS y Header Injection

```
xsstest  
"><script>alert(1)</script>
```

> OS Command Injection

```
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &  
| ping -i 30 127.0.0.1 |  
| ping -n 30 127.0.0.1 |  
& ping -i 30 127.0.0.1 &  
& ping -n 30 127.0.0.1 &  
; ping 127.0.0.1 ;  
%0a ping -i 30 127.0.0.1 %0a  
' ping 127.0.0.1 '
```

> Path Traversal

```
../../../../../../../../../../../../etc/passwd  
../../../../../../../../../../../../boot.ini  
../../../../../../../../../../../../etc/passwd  
../../../../../../../../../../../../boot.ini
```

> Script Injection

```
;echo 111111  
echo 111111  
response.write 111111  
:response.write 111111
```

> File inclusion

```
http://<servidor>/  
http://1.2.3.4/
```

□ Configure a set of strings in the Grep function that indicate that the response has changed.

>>> For example: error, exception, illegal, invalid, fail, stack, access, directory, file, not found, varchar, ODBC, SQL, SELECT, 111111... As well as the payloads that you have introduced.

~~~~~

Notes:

github repository very interesting for this task:

<https://github.com/danielmiessler/SecLists>

# Test SQL Injections

- ❑ Enter single quotes (') in all the fields you find.
- ❑ If you get an error message related to the database, there is a SQL Injection vulnerability.
- ❑ Normally only very old and poorly maintained pages have standard or error-based SQL Injection. However, it is common to find domains vulnerable to Blind SQL.
  - >>> To discover them, it is common to try payloads of the type:  
' OR '1'='1' -- a
  - >>> Next, look at the request with Burp and repeat it by inserting something of the type:  
' OR '1'='2' -- a
  - >>> If the answers are different, then you can ask questions to the DB after the OR, and get all the values of the DB.
  - >>> Adding to that request something like: AND WAITFOR DELAY '0:0:10' we find a Blind SQL based on time if the first answer takes 10 seconds longer than the second.
- ❑ Once you have detected the injection, use an automated tool to exploit it.
  - >>> The best tool for this is SQLMap. It is important to set it up correctly to get the most out of it.

~~~~~

Notes:  
The payloads shown are examples, and depending on the type of database the language is different.  
You can find cheat sheets of each database at the following URL:  
<http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>  
The following post can help you better understand SQL Injection and SQLMap: <http://hackingethani.com/es/sql-injection-introduccion/>

# Test XSS and other injections

# Identify reflected XSS

- For each parameter, try writing values and check if those values appear in the answer.
  - >>> For example, if the domain has a search panel and you enter "asdagdsfa" the answer is "There are no results for asdagdsfa' ", you may be able to perform an XSS
- Try to enter `<script>alert(1)</script>`
  - >>> It may have filters that sanitize the values, removing dangerous chains such as `<string>`. But still try other ways, such as `<a onclick=alert(1)>click me!</a>`.
  - >>> If it filters the `alert(1)`, try other methods such as `confirm(1)` or `prompt(1)`.
- If you notice any change in the source code or on the web, it is almost certain that you can perform the XSS, but you will have to go modifying it.
  - >>> To do this check where it is being written, and go adding what you need to be well formed. Maybe you need to close other tags or functions (adding at the beginning `>`, `)` or `}` ).

~~~~~

Notes:

You can find a cheat sheet of ways to avoid anti XSS filters in the following link:

[https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet)

# Test HTTP header injection

- ☐ In any header that shows the value of a request parameter, enter a carriage return (%0d) and a line break (%0a).

>>> If a new header line appears that responds to the server, it is vulnerable to HTTP header injection.

- ☐ If the server uses some filter, check how viable it is to skip it:

>>> foo%00%0d%0abar  
>>> foo%250d%250abar  
>>> foo%%0d0d%%0a0abar

~~~~~

Notes:

# Test open redirections

- ☐ If one of the parameters is a URL that the server redirects to, try changing it to another one to see if you can redirect the user wherever you want.

~~~~~

Notes:

# Test stored attacks

- ☐ Check if the data is stored and then displayed. If this is the case and you have managed to perform an XSS, then you will be shown each time that data is shown to you.
- ☐ Look for parameters that are saved and then shown to other users.
  - >>> if a vulnerable parameter introduced by a user with few privileges is shown to an administrator user, it is possible to perform a privilege escalation.

~~~~~

Notes:

# Test command injection

- If one of the fuzzing command injection attacks has been a very long response time, it can be vulnerable.
  - >>> Try more interesting commands as ls or dir.
- If you can't see the results directly, you can try other options:
  - >>> Try creating a reverse shell with netcat or telnet.
  - >>> Try to dump the command in a file and save it in a path that you can access from your computer.
- If you have a shell, do a whoami to see what privileges you have and move on to post-exploitation and escalation of privileges.
- If the application escapes some characters from the command, try inserting an escape character (\) before each character.

~~~~~

Notes:



# Test Path Traversal

- If a parameter appears to contain a filename or part of a filename or directory, try modifying that parameter and add ../ without changing the path
  - >>> For example, if the parameter is file=foo/file1.txt, put file=foo/bar/../file1.txt instead.
  - >>> If the answer is the same, then it is vulnerable.
- If the previous test worked, then maybe you can go backwards with ../ and get into the files that interest you.
  - >>> Maybe it has a filter by file type, so if you tell that parameter something other than a txt (for example), it won't show it.
  - >>> To avoid this try entering a null or new line bit to see if it doesn't detect the file type:
    - > ../../../../../../boot.ini%00.jpg
    - > ../../../../../../etc/passwd%0a.jpg
  - >>> If the filter checks that the path starts with a certain directory, it tries something like /images/../../../../../../../../etc/passwd
- Interesting files if you have access to read files:
  - >>> OS and app password files.
  - >>> Server and app configuration files.
  - >>> Files that may contain database credentials.
  - >>> Files used by the app, such as MySQL database files or XML files.
  - >>> Server executable source code.
  - >>> Application logs that can have usernames or session tokens.
- Interesting things if you have access to file writing:
  - >>> Create scripts in user home folders.
  - >>> Modify files like in.ftpd to execute arbitrary commands when the user connects.
  - >>> Write scripts in a web directory with execution permissions and call it from your browser.

~~~~~  
Notes:

The dotdotpwn program allows automatic path traversal tests.

# Test file inclusion

- If you can upload files, try uploading a shell.

~~~~~

Notes: use eicar (signature to test if you can upload virus)

There is the eicar in parrot: /usr/share/metasploit-framework/data/eicar.com

You can change the .com for any extension. If it lets you upload it, you have a high vulnerability.

## 8. Specific Functionality Tests

# Test SMTP injection

- In requests related to email functionality, enter payloads such as the following:

<youremail>%0aCc:<youremail>

<youremail>%0d%0aCc:<youremail>

<youremail>%0aBcc:<youremail>

<youremail>%0d%0aBcc:<youremail>

%0aDATA%0afoo%0a%2e%0aMAIL+FROM: +<youremail>%0aRCPT+TO: +<youremail>

%0aDATA%0aFrom: +<youremail>%0aTo: +<youremail>%0aSubject: +test%0afoo%0a%2e%0a

%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM: +<youremail>%0d%0aRCPT  
+TO: +

<youremail>%0d%0aDATA%0d%0aFrom: +<youremail>%0d%0aTo: +<youremail>

%0d%0aSubject: +test%0d%0afoo%0d%0a%2e%0d%0a

~~~~~

Notes:

You can use pages that generate temporary mails, such as <http://www.yopmail.com/es/>

To impersonate an email, you can use pages like <https://emkei.cz/>

# Test native software vulnerabilities

Desbordamiento de búfer, vulnerabilidades de Integer y de String.

# Test buffer overflow

- ☐ Try introducing very large strings that can exceed the buffer size.
- ☐ Check for abnormal responses, such as HTTP 500 errors, error messages, unexpected connection shutdown, the app does not respond...

~~~~~

Notes:  
You can find more information about the buffer overflow in the following post: <http://hacking1ethani.com/buffer-overflow-en/>

# Test integer vulnerabilities

- In numeric fields try to enter limit values that may result in a change in the size of the integer:  
0x7f y 0x80 (127 y 128)  
0xff y 0x100 (255 y 256)  
0x7ffff y 0x8000 (32767 y 32768)  
0xffff y 0x10000 (65535 y 65536)  
0x7fffffff y 0x80000000 (2147483647 y 2147483648)  
0xffffffff y 0x0 (4294967295 y 0)

- If the data is in hexadecimal, try both the little-endian and big-endian versions.

~~~~~

Notes:

The following combination may be interesting: 4 8 15 16 23 42

## Test strings format vulnerabilities

- Send long chains of special characters:

%n  
%s  
%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! etc...  
%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! etc...

~~~~~

Notes:



# Test SOAP injection

- Try injecting a closing XML tag, like `</foo>`. If there are no errors, it's probably not being inserted into a SOAP message, or it's being sanitized.
- If there are errors, try inserting an opening and a closing tag, such as `<foo></foo>`. If the error disappears, then it is vulnerable
- If what you put is copied into the answer, try putting these two values in order. If you see that one is returned as the other, or simply appears "test", it is certain that you are inserting the entry in a message based on XML.  
test<foo/>  
test<foo></foo>
- Try inserting comments ( `<!--` and `!-->` ) in the parameters. There may be parts of the code that allow access to some area of the app or that give an error message that gives information. Modify where you put the comments, since you don't know the order in which the parameters appear.

~~~~~

Notes:

# Test LDAP injection

- ☐ Searches for parameters where data entered by the user is used to provide information from a directory service.
- ☐ Send an asterisk \*. If you return many results, you may be vulnerable to LDAP.
- ☐ Try entering many closing parentheses )))))). If an error occurs, the app is vulnerable. Then try this:  

```
)(cn=*
```

```
*))(|(cn=*
```

```
*))%00
```
- ☐ Try adding extra attributes at the end of the entry, using commas. Errors will indicate if the attribute is valid in the current context.

```
cn
c
mail
givenname
o
ou
dc
l
uid
objectclass
postaladdress
dn
sn
```

~~~~~

Notes:

# Test XPath injection

- Try entering this, and see if something different appears in the application without giving an error:  
' or count(parent::\*[position()=1])=0 or 'a'='b  
' or count(parent::\*[position()=1])>0 or 'a'='b
- If the parameter is numerical try the following:  
1 or count(parent::\*[position()=1])=0  
1 or count(parent::\*[position()=1])>0
- If there is a difference but the app doesn't cause an error, you can extract information to know the current parent node:  
substring(name(parent::\*[position()=1]),1,1)='a'
- Once you know which is the parent node, you can pull out the whole XML tree by making requests like this:  
substring(//parentnodename[position()=1]/child::node()[position()=1]/text(),1,1)='a'

~~~~~

Notes:

# Test XXE injection

- If users send XML to the server, it may be possible to perform an external injection attack. If you know a field that is returned to the user, try specifying an external entity, such as the following:

POST /search/128/AjaxSearch.ashx HTTP/1.1

Host: mdsec.net

Content-Type: text/xml; charset=UTF-8

Content-Length: 115

<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///windows/win.ini" > ]>

<Search><SearchTerm>&xxe;</SearchTerm></Search>

- If no field can be found, specify an external entity such as "http://192.168.1.1:25" and monitor the page response time. If it takes too long to respond, you may be vulnerable.

~~~~~

Notes:

## 9. Logical Fault Tests

# Identify key attack surface

□ Identify instances with the following characteristics:

- > Multi-step processes
- > Critical security features such as login
- > Transitions such as going from being anonymous to self-registering to login.
- > Functionality based on user context
- > Checks and adjustments made on orders or quantities.

~~~~~

Notes:

# Test handling of incomplete input

- For each parameter, removes both the name and the value of the request. Check the responses returned by the server.

~~~~~

Notes:

# Test trust limits

□ Try things like going from a process in which a user is recovering his account, to a specific page of the user, to know how the application reacts and in which cases you can make an escalation of privileges in this way.

~~~~~

Notes:



## 10. Shared Hosting Problem Tests

# Test segregation in shared infrastructures

□ If it is a shared hosting, ask things like:

- >>> Does the remote access use a secure protocol?
- >>> Can clients access files, data or other resources that they should not have access to?
- >>> Can clients get an interactive shell with the hosting environment and execute commands

□ If in one of the shared hosting apps you can execute commands, SQL Injection, or access private files, check if you can scale this attack to other applications.

~~~~~

Notes:

# Test segregation between ASP-Hosted applications

- If a common database is used for applications in a shared environment, check the configuration of the database, version, patches, table structure and permissions, etc. Any defect can prove that you can scale the attack from one app to another.

~~~~~

Notes:

To analyze the configuration of the database you can use tools such as NGSSquirrel.

# 11. Web Server Tests

Wordpress: usar wpscan

# Test default credentials

- ☐ Make a port mapping of the server to see what services you could access.
- ☐ Check if there are default credentials for the technology the server uses.

~~~~~

Notes:

# Test default content

- ☐ Use Nikto to identify content that is present on the server but is not part of the application.
- ☐ Identify the default content of the technology used by the server.  
>>> For this you can use pages such as [www.exploit-db.com](http://www.exploit-db.com)

~~~~~

Notes:

# Test dangerous HTTP methods and headers

- Use the OPTIONS method to list the HTTP methods the server accepts.

```
>>> $ nmap -sS -p 443 --script http-methods dominio.com
>>> $ nmap -sS -p 443 --script http-enum dominio.com

>>> $ nc dominio.com 80 -v
  OPTIONS /alguna_ruta_concreta_o_solo_la_barra HTTP/1.0
  OPTIONS / HTTP/1.1
```

- The server must have the following headers:

- > Content-Security-Policy
- > Public-Key-Pins
- > X-Frame-Options
- > X-XSS-Protection
- > X-Content-Type-Options
- > Strict-Transport-Security

~~~~~

## Notes:

To check the headers you can do it through the following website: <https://securityheaders.io>  
You must check the "Hide results" option.

# Test proxy functionality

- ☐ Using GET and CONNECT, try to use the web server as a proxy to connect to and receive content from other internet servers.
- ☐ Using GET and CONNECT, try to connect to different IPs and ports within the host.
- ☐ Using GET and CONNECT, try to connect to common ports on the same web server indicating 127.0.0.1 as the target in the response.

~~~~~

Notes:



# Test the wrong virtual hosting configuration

☐ Send GET requests to the root directory using:

- >>> The correct Host header.
- >>> The host header with errors.
- >>> The IP of the server in the Host header
- >>> No Host header (only works on HTTP/1.0)

☐ Compare the answers. It is possible that when you put the IP in the Host header you will get a list of directories.

~~~~~

Notes:  
You can use Nikto with the -vhost option to identify default content that didn't appear in the initial mapping.

# Test web server software bugs

- ☐ Run a scanner like Nessus to identify known vulnerabilities of the web server you are attacking.
- ☐ Check sites such as Security Focus, Bugtraq, cvedetails etc. for exploitable vulnerabilities.
- ☐ If possible do a local installation of the technology used by the server.

~~~~~

Notes:

## 12. Miscellaneous Checks

# Check DOM-based attacks

- Look for one of the following methods, which can be used to access DOM data via a modified URL:

```
>>> document.location
>>> document.URL
>>> document.URLUnencoded
>>> document.referrer
>>> window.location
```

- If any of the following calls are being made, the application may be vulnerable to XSS:

```
>>> document.write()
>>> document.writeln()
>>> document.body.innerHTML
>>> eval()
>>> window.execScript()
>>> window.setInterval()
>>> window.setTimeout()
```

- If any of the following calls are being made, you may be vulnerable to redirection attacks:

```
>>> document.location
>>> document.URL
>>> document.open()
>>> window.location.href
>>> window.navigate()
>>> window.open()
```

- XML-RPC Testing

```
<?xml version="1.0"?>
<methodCall>
<methodName>system.listMethods</methodName>
<params>
<param>
</param>
</params>
</methodCall>
```

~~~~~

Notes:

# Check local privacy vulnerabilities

- ☐ Check in the Set-Cookie header of the server if it has the expires attribute, and how long it lasts.
- ☐ If there are persistent cookies that contain (or provide access to) sensitive data, an attacker who makes use of the cookie will be able to access that data.
- ☐ Check that no sensitive information is sent by GET.
- ☐ Searches for possible information stored in the browser or in the specific storage of the technology you are using.

~~~~~

Notes:

# Test Same-Origin configuration policy

- ☐ Check if the application is vulnerable to Cross-Domain requests by modifying the Origin header of the requests and indicating a different domain, and examine what it puts in the Access-Control header that the server returns to you in the response.

~~~~~

Notes:

## 13. Information Leaks

# Metadata

□ If you find documents through a directory listing, because you can access them through the app or through google searches, analyze their metadata.

>>> A good way to do this is with FOCA, where not only can you analyze metadata, but also search documents in Google, Bing and Duck Duck Go, with the ability to add the API Key Google and Shodan to search their APIs.

FOCA Open Source 3.4

Project Plugins Options TaskList About

No project  
Network  
Domains  
Metadata

**Foca**  
OPENSOURCE

Select project  
Project name  
Domain website  
Alternative domains  
Folder where save documents  
Project date  
Project notes

Create Import Cancel

| Time | Source | Severity | Message |
|------|--------|----------|---------|
|------|--------|----------|---------|

Conf Deactivate AutoScroll Clear Save log to File

>>> It is not recommended that you analyze metadata of sensitive documents online, on pages such as <https://metashieldclean-up.elevenpaths.com/> , as it is possible that this information will be stored.

Notes:



# Error messages

☐ Error messages ask for a lot of debugging information and server information.

☐ You can use google dorks to investigate common error messages.

>>> Example:

"unable to retrieve" filetype:php

~~~~~

Notes: