

Week 3: Into the Web App-iverse

Web Application Hacking

<https://da.gd/jFxpU3>



SIGN IN PLEASE

<https://da.gd/jFxp3>

whoami

Gabriel Fok | baseq

4th Year CS

ISSE @ Boeing

CCDC

Linux Team 2020-2021

Linux Lead 2021-2022

Captain 2022-2023

CPTC

Team Member 2021-2022

Co-Captain 2022-2023

Business Lead 2023-2024



whoami

Derrick Tran | Dumosuku

5th year CIS

Intern @ Google / Mandiant

CPTC

Team Member 2022-2023

Co-Captain 2023-2024



Next on Bronco CPTC ...

When	What
July 8th	Introduction to CPP Cyber
July 15th	Intro to Penetration Testing
July 22nd	Hacking Web Applications
July 29th	Hacking Linux
August 5th	Hacking Windows
August 12th	Consulting
August 19th	Tryouts
August 26th	Full CPTC Team Selected

← You
are
here

Previously on CPTC ...

- Penetration Testing Methodology
- Kali Linux
- Client-Server Model
- Ports, network connections, and shells

Agenda

1

The Basics of Web

How web applications work

2

WAPTM

Web App Penetration Testing Methodology

3

Web App Vulnerabilities

There's a lot, focus on understanding

4

Lab

Learn by doing

01

What are Web Applications




What are Web Applications?

 Interactive web-pages

> Client (User) Interacts with frontend



 Applications that run on web servers

> Their purpose is to provide service(s)

How Web Apps Work

Client Sends Request

Client crafts HTTP request

Client sends HTTP request



Server capabilities include: database, command execution, file read/write

Server Handles Processing

Server receives HTTP request

Server determines requested resource(s)

Server runs requested functions/processes



Server Sends Response

Server sends response code and response data, if applicable

HTTP Request Methods

GET: Request a resource

POST: Send data to a server for processing

PUT: Set a resource on the server

DELETE: Delete a resource on a server

HEAD: Request a page without its contents

OPTIONS: Request allowed methods

HTTP Response Code Categories

Code	Category
100-199	Informational
200-299	Success
300-399	Redirect
400-499	Client Error
500-599	Server Error

Common HTTP Response Code Examples

Success: 200

Permanent Redirection: 301

Access Denied: 403

Not Found: 404

Internal Server Error: 500

Example POST Request

Header

```
POST /purchase.php HTTP/1.1
Host: redemption.nft
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Origin: http://redemption.nft
Connection: close
Referer: http://redemption.nft/register.php
Cookie: PHPSESSID=0qgp1s8fb7lsf13av4qbojc4l7
Upgrade-Insecure-Requests: 1
```

```
ownerID=24&recipientID=25
```

Example POST Request Part II

Request Line: **POST** **/purchase.php** HTTP/1.1

Request Headers: **Host:** redemption.nft

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
Gecko/20100101 Firefox/91.0

Request Body: ownerID=24&recipientID=25

POST HTTP Method

/purchase.php Path to the page where the request is sent

Host | User-Agent Request Headers

redemption.nft Data provided by the browser

ownerID | recipientID Client-side provided data

Example HTTP Response

Response Line: 200 OK

Response Headers: Content-Type: text/html; charset=utf-8

Date: Fri, 26 Feb 2021 18:00:00 GMT

Server: Apache2

Set-Cookie: secret=myvalue

Response Body: <html>Hello</html>

200 HTTP Response Code

OK HTTP Response Message

Content-Type | Date Response Headers

secret=myvalue Data provided by the browser

<HTML> Response Body

3 Types of Data Lifetimes

Application/Stored

- Stored on the server, persistent

Session

- Stored on either client/server, duration of session

Request

- Sent from the client, unique per request

Example Server Side Code

Request Data

SQL

```
<?php
    if(isset($_POST["btn"])) {
        include("connect.php");
        $item_name=$_POST['iname'];
        $item_qty=$_POST['iqty'];
        $item_status=$_POST['istatus'];
        $date=$_POST['idate'];

        $q="insert into grocerytb(Item_name,Item_Quantity,Item_status,Date)
            values('$item_name',$item_qty,
                '$item_status','$date')";

        mysqli_query($con,$q);
        header("location:index.php");
    }
?>
```

02

WAPTM

Web App Pentesting
Methodology



Web App Pen Testing Methodology



Discovery



Configuration

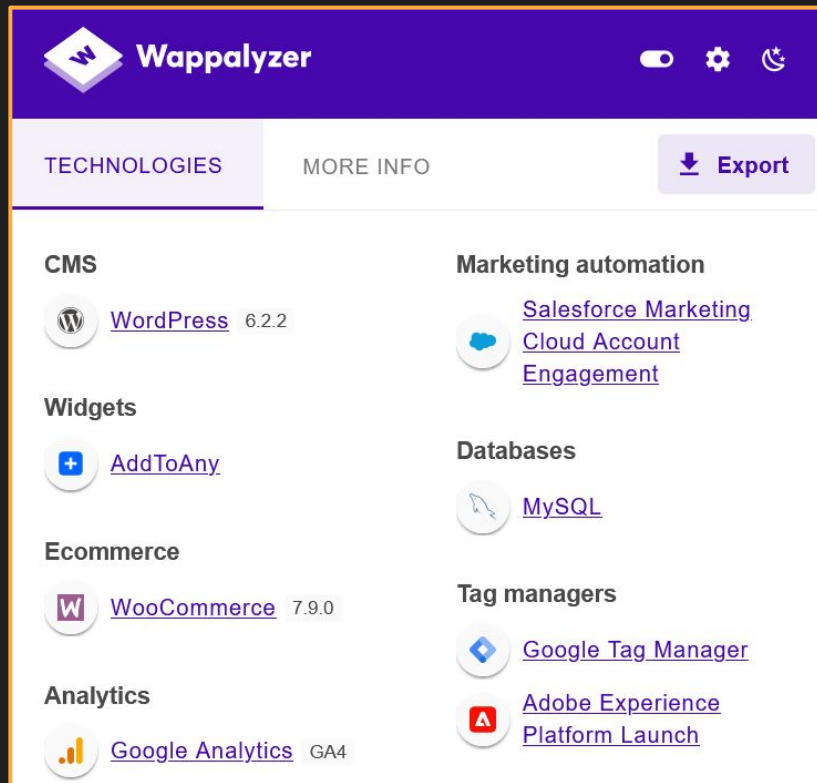


Data Validation

View Source

```
15 <nav class="navbar navbar-expand-xl navbar-dark bg-dark" style="background-color: #ffffff;">
16   <div class="container-fluid">
17     <a class="navbar-brand" href="index.php">
18       
19     </a>
20     <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#main_nav" ari
21     <span class="navbar-toggler-icon"></span>
22   </button>
23   <div class="collapse navbar-collapse" id="main_nav">
24     <ul class="navbar-nav ms-auto">
25       <li class="nav-item"><a class="nav-link" href="browse.php">Browse Files</a></li>
26       <li class="nav-item"> <a class="nav-link" href="search.php">Search Listing </a> </li>
27       <li class="nav-item"><a class="nav-link" href="create_listing.php">Create Listing</a></li>
28     <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown">
29       admin </a>
30     <ul class="dropdown-menu dropdown-menu-end">
31       <li><a class="dropdown-item" href="profile.php"> View Profile </a></li>
32       <li><a class="dropdown-item" href="auth_check.php"> Dashboard </a></li>
33       <li><a class="dropdown-item" href="logout.php"> Logout </a></li>
34     </ul>
35   </li> </ul>
36 </div> <!-- navbar-collapse -->
37 </div> <!-- container-fluid -->
38 </nav> <div class="hero" style="padding:7rem;">
39   <h1>Page Under Development</h1>
40   <h2>Testing file upload functionality</h2>
41 </div>
42
43 <div class="uploadBackground">
44   <form action="upload.php" method="post" enctype="multipart/form-data">
45     Select file to upload:
46     <input class="upload" type="file" name="fileToUpload" id="fileToUpload">
47     <input type="submit" value="Upload File" name="submit">
48   </form>
49 </div>
```

Wappalyzer




The screenshot shows the Wappalyzer interface with a purple header. The main content area is divided into two columns. The left column lists technologies under categories: CMS (WordPress 6.2.2), Widgets (AddToAny), Ecommerce (WooCommerce 7.9.0), and Analytics (Google Analytics GA4). The right column lists technologies under categories: Marketing automation (Salesforce Marketing Cloud Account Engagement), Databases (MySQL), and Tag managers (Google Tag Manager, Adobe Experience Platform Launch). An 'Export' button is visible in the top right.


Wappalyzer

TECHNOLOGIES MORE INFO [Export](#)


CMS

-  [WordPress](#) 6.2.2


Widgets

-  [AddToAny](#)


Ecommerce

-  [WooCommerce](#) 7.9.0


Analytics

-  [Google Analytics](#) GA4



Marketing automation

-  [Salesforce Marketing Cloud Account Engagement](#)

Databases

-  [MySQL](#)

Tag managers

-  [Google Tag Manager](#)
-  [Adobe Experience Platform Launch](#)

Burp Suite

Burp Suite interface showing the HTTP history and the details of a selected request.

HTTP History Table:

#	Host	Method	URL	Params	Status	Length	MIME type	
1	http://redemption.nft	POST	/register.php	✓	302	3611	HTML	php
2	http://redemption.nft	GET	/login.php		200	2974	HTML	php

Request Details (Raw):

```
1 POST /register.php HTTP/1.1
2 Host: redemption.nft
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 87
9 Origin: http://redemption.nft
10 Connection: close
11 Referer: http://redemption.nft/register.php
12 Cookie: PHPSESSID=0qgp1s8fb7lsf13av4qbojc4l7
13 Upgrade-Insecure-Requests: 1
14
15 username=hehe&password=password&confirm_password=password&wallet_id=hehe&account_type=2
```

Gobuster



Gobuster can use wordlists to verify whether or not an endpoint exists by attempting to visit them

```
gobuster dir -u http://redemption.nft -w ./raft-large-directories-lowercase.txt -x php
```

```
=====
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:                http://redemption.nft/
[+] Method:             GET
[+] Threads:           10
[+] Wordlist:           /usr/share/seclists/Discovery/Web-Content/raft-large-directories-lowercase.txt
[+] Negative Status codes: 404
[+] User Agent:         gobuster/3.1.0
[+] Extensions:        php
[+] Timeout:           10s
=====
2022/09/28 03:05:52 Starting gobuster in directory enumeration mode
=====
/search.php             (Status: 200) [Size: 3143]

[...]

/browse.php             (Status: 403) [Size: 135]
/listing.php            (Status: 302) [Size: 2094] [--> login.php]
```


Wordlists



Passwords

`/usr/share/wordlists/rockyou.txt`

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

Directories

`/usr/share/seclists/Discovery/Web-Content/raft-large-directories-lowercase.txt`

`/usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt`

Virtual Hosts/Subdomains

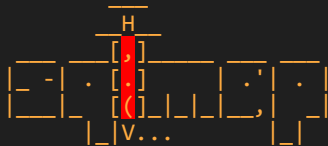
`/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt`

SQLMap



SQLMap automatically checks for sql injection vulnerabilities by attempting many different payloads

```
sqlmap -r ./req.txt
```

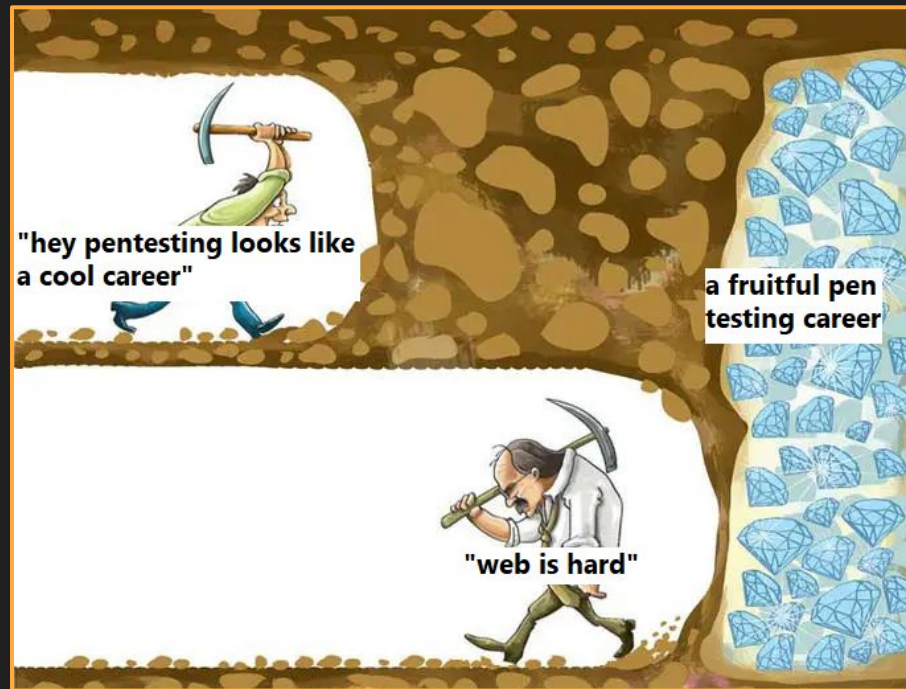


```
[02:13:59] [INFO] testing connection to the target URL
[02:14:02] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[02:14:02] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[02:14:02] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[02:14:31] [INFO] GET parameter 'q' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'q' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
---
Type: UNION query
Title: Generic UNION query (NULL) - 6 columns
Payload: q=asdf') UNION ALL SELECT 49,49,49,49,49,49-- -
---
```

Why Web is Hard

Reasons

- Guess heavy
- Usually blind
- Many server technologies
- Lots of possible vectors (injections)
- WAFS AGHHHHHHH



03 Web App Vulnerabilities

Sanitize ***all*** the inputs!!!!

Genie: You have three wishes

Me: ; DROP TABLE Wishes

Genie:



SQL Injection



TLDR: SQLi is crafting malicious backend SQL statements

`http://redemption.nft/search.php?q=lmao`


```
└─ SELECT * FROM listing WHERE ('listingName' LIKE '%lmao%')
```

Application makes a SQL query to a database



How can we **exploit** this with SQLi?

How Can We Exploit SQLi?

 `http://redemption.nft/search.php?q=lmao%')OR+1=1-- -`

`')`: ends the **'listingName'** part of the SQL statement

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

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

`[...]`: Original SQL statement

```
SELECT * FROM listing WHERE ('listingName' LIKE '%lmao%') OR 1=1-- -
```

Local / Remote File Inclusion



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



`http://redemption.nft/browse.php?file=sink.png`



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



How can we exploit this with LFI/RFI?

How Can We Exploit LFI/RFI?



`http://redemption.nft/browse.php?file=sink.png`

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

`file`: name of the parameter

`sink.png`: 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!

What is Command Injection?



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

`http://redemption.nft/purchase.php?ownerID=24&recipientID=25`



The application runs `purchase.php` to get information. It takes 2 values which are used as variables within a command.



How can we **exploit** this with command injection?

How Can We Exploit Command Injection?

Normal POST Request:

ownerID=24&recipientID=25



purchase.php will trade the item by swapping owner id 24 and recipientID 25

Injecting a Command into the POST Request

ownerID=24&recipientID=25
;<command>



purchase.php will trade the item by swapping owner id 24 and recipientID 25 and execute a command

How can we exploit Command Injection?

```
POST /purchase.php HTTP/1.1
Host: redemption.nft
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Origin: http://redemption.nft
Connection: close
Referer: http://redemption.nft/register.php
Cookie: PHPSESSID=0qgp1s8fb7lsf13av4qbojc4l7
Upgrade-Insecure-Requests: 1
```

```
ownerID=24&recipientID=25;ping -c 2 x.x.x.x
```

[...]: request parameters

;: ends the command and allows for us to use another

ping: is our command

What is Mass Assignment (Object Injection)?



Intercepting the post request data of a form allows an attacker to pass their own values to the server

Client

```
<form>
  <input name="username" type="text">
  <input name="password" type="text">
  <input name="confirm_password" type="text">
  <input name="wallet_id" type="text">
  <input name="account_type" type="hidden">
  <input type="submit">
</form>
```



Server (PHP)

```
$param_username = $username;
$param_password = $password;
$param_wallet_id = $wallet_id;
$param_account_type = $account_type;
```

How can we abuse this POST request?

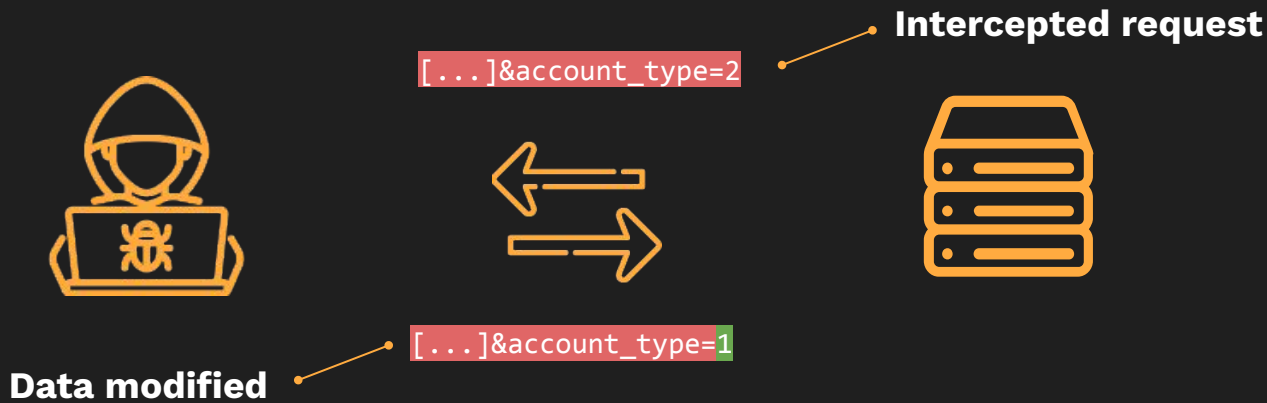
```
POST /register.php HTTP/1.1
Host: redemption.nft
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Origin: http://redemption.nft
Connection: close
Cookie: PHPSESSID=0qgp1s8fb7lsf13av4qbojc4l7
Upgrade-Insecure-Requests: 1
```

```
username=bobathin&password=asdf&confirm_password=asdf&wallet_id=1234&account_type=2
```

How Can We Take Advantage of This?



The hacker can intercept the original POST request, make modifications, and resend it to the server.



Server Side Template Injection



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

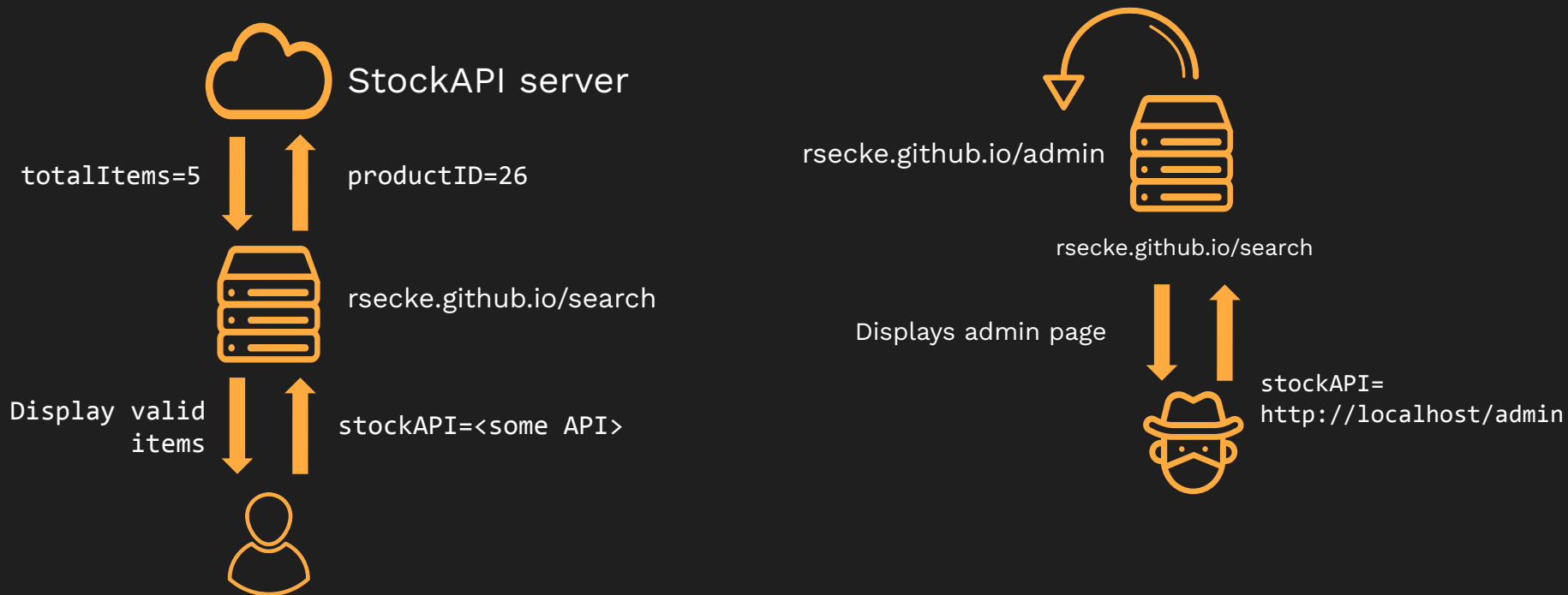
Examples: Jinja2 (Python) & Twig (Java)

<http://redemption.nft/comments>



The same previous application, but it is a Flask application

Server Side Request Forgery



Insecure Access Controls



Parameter-Based Access Methods

- User rights determined at login
admin:0



Referer-Based Access Control

- Authorization based on previous site
covertops.xyz/admin
covertops.xyz/admin/deleteUser



Insecure Direct Object References

- Occurs when user-input is used to determine which objects to access
redemption.nft/search.php/listingID=0

04 Lab/Homework

