Web Exploitation Workshop

Khael Kugler

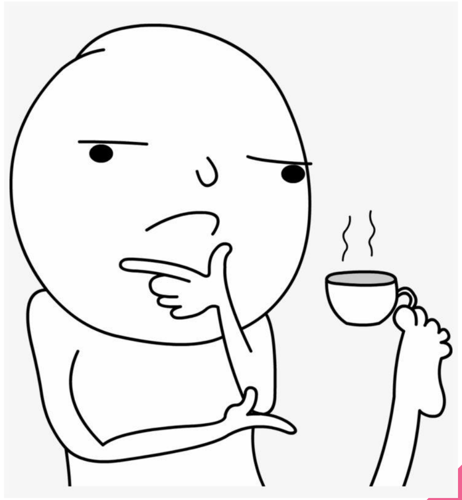
whoami

- Khael Kugler
- Security Engineer at Praetorian
  - Web, IoT/Embedded (mostly medical), External/Internal Networks
- ISSS + Hash Alum
- #1 UT Bug Bounty
  - 🪦 rip, you'll be missed 🪦
- Defcon RTV Wargames Winner

praetorian

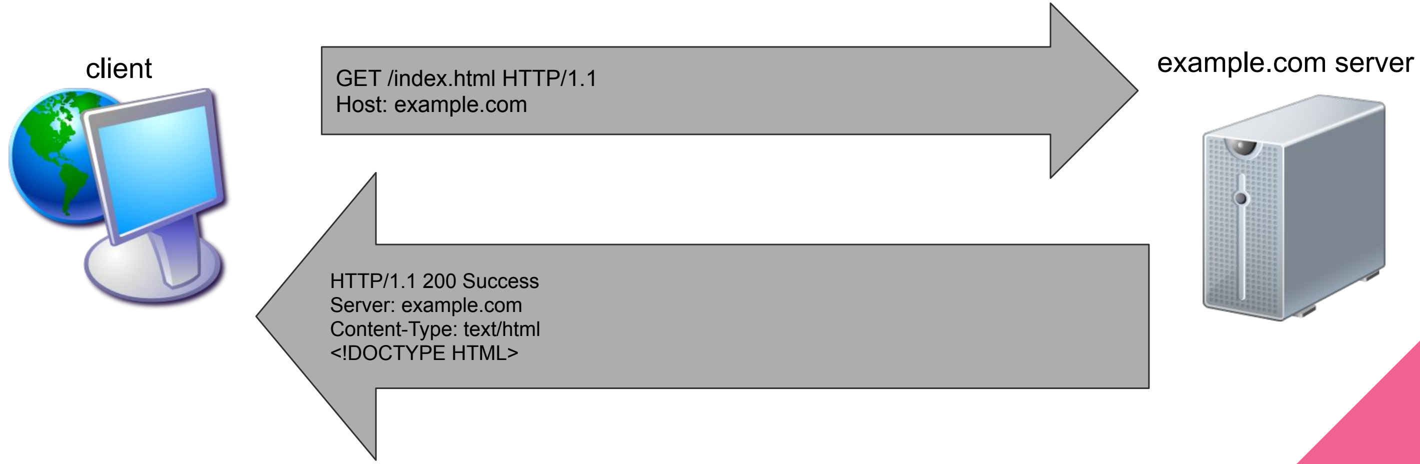
A bit of contextualization

- Localhost?? What is that
- Security?? What is that
  - I could not exploit a single vulnerability on these slides
  - I probably had about 4 CTF solves
- Security might seem daunting
  - Guess I gotta do SWE...



What is this “web”? Requests and Responses:

- HTTP requests and responses
- Request:
  - Asking a server for some data
- Response:
  - The server sends that data

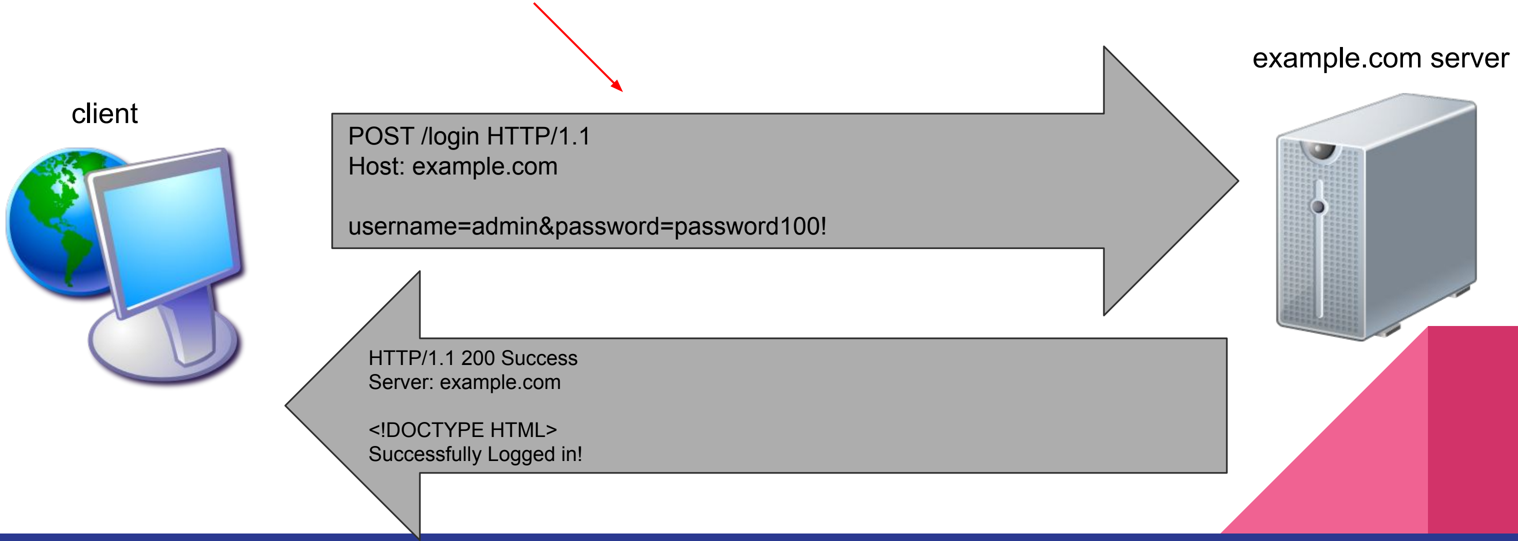


```
GET /index.html HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 Success
Server: example.com
Content-Type: text/html
<!DOCTYPE HTML>
```

Two Most Common HTTP Requests

- GET Request:
  - Retrieving data from a server
- POST Request
  - Providing data to a server

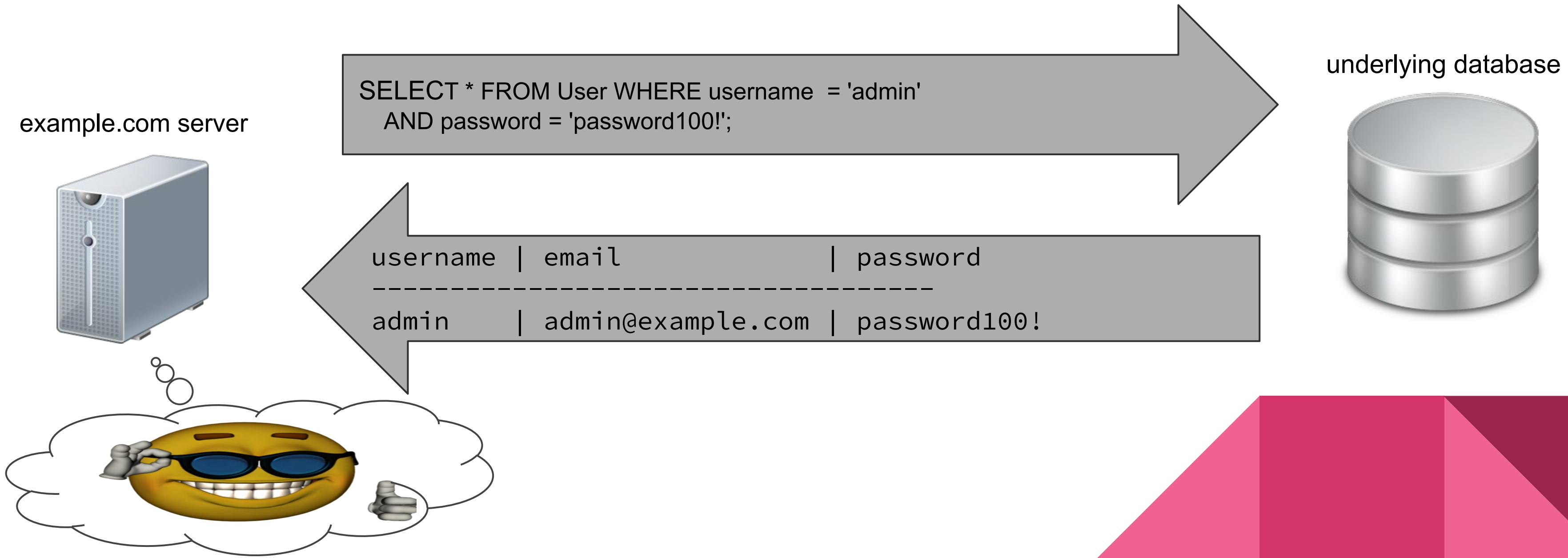


```
POST /login HTTP/1.1
Host: example.com

username=admin&password=password100!
```

```
HTTP/1.1 200 Success
Server: example.com

<!DOCTYPE HTML>
Successfully Logged in!
```

But how do servers keep track of information?

- Databases!
  - Can store whatever your heart desires

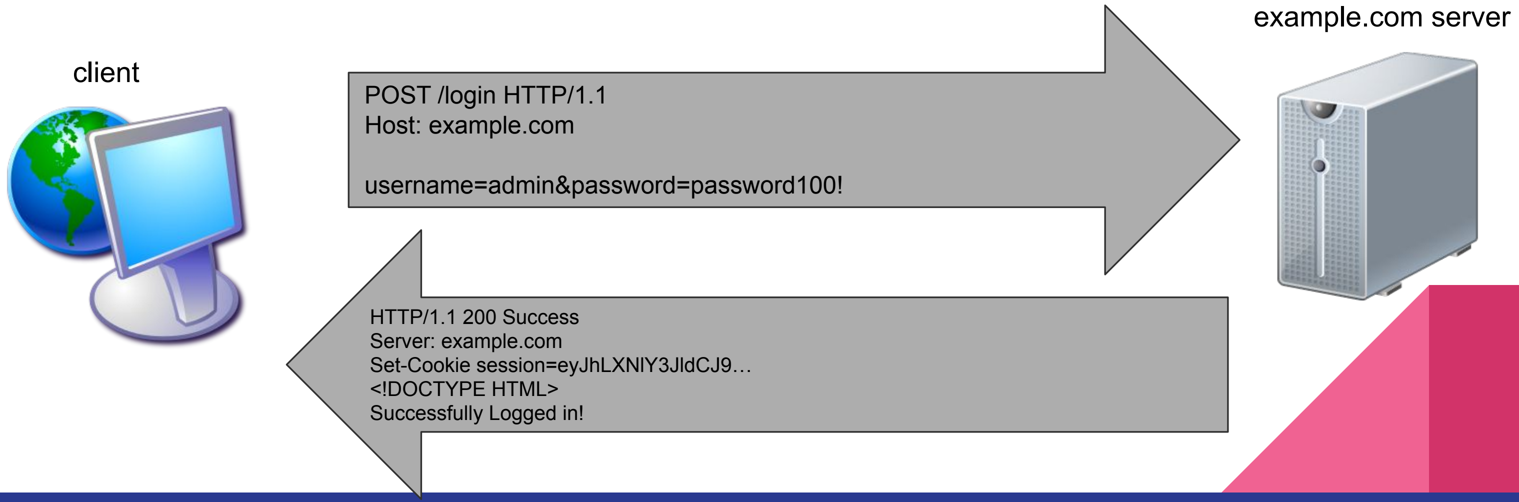


```
SELECT * FROM User WHERE username  = 'admin'
   AND password = 'password100!';
```

```
username | email             | password
-------------------------------------------
admin    | admin@example.com | password100!
```

Authentication Tokens

- Servers keep track of users via an authentication token
  - Usually a cookie or an HTTP Authorization header
  - When the user presents that token, the server will consider them authenticated

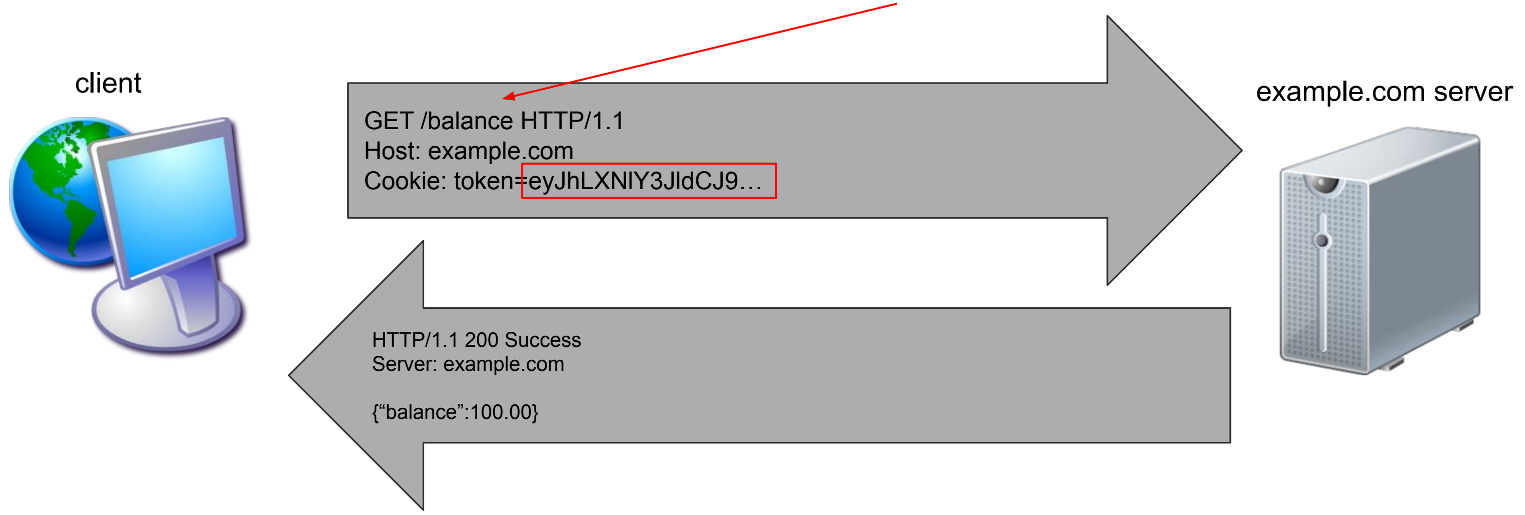


```
POST /login HTTP/1.1
Host: example.com

username=admin&password=password100!
```

```
HTTP/1.1 200 Success
Server: example.com
Set-Cookie session=eyJhLXNlY3JldCJ9…
<!DOCTYPE HTML>
Successfully Logged in!
```

Auth tokens continued…

- The user can now access authenticated resources



```
GET /balance HTTP/1.1
Host: example.com
Cookie: token=eyJhLXNlY3JldCJ9…
```

```
HTTP/1.1 200 Success
Server: example.com

{“balance”:100.00}
```

Auth tokens can be implemented in an infinite number of ways, always investigate them to see how they’re used!

Common Vulnerabilities

Injection - Command Injection

- The holy grail
- Inserting your own commands into the application's execution

```
$IP_to_ping = input("Ping an IP:")
system("ping -c 1 " + $IP_to_ping + ";")
```

https://example.com/ping?ip=127.0.0.1;id

```
ping -c 1 127.0.0.1;id
```

```
Your ping results:
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.017/0.017/0.017/0.000 ms

uid=0(root) gid=0(root) groups=0(root)
```

Injection - SQL Injection

- Accessing a SQL database without permission
- Usually via apostrophe or quotation mark to escape a parameter

```
$query = "SELECT balance FROM
users WHERE user_id = '$id'; "
```

https://example.com/balance?id=5' OR 1=1;--

```
$query = "SELECT balance FROM users WHERE
user_id = '5' OR 1=1;-- ';";
```

```
Your balance:
User1: $12,500.00
User2: $67,094.22
User3: $152.00
...
```

Injection - Cross-Site Scripting (XSS)

- Injecting HTML/JavaScript tags into a page
  - Generally through a URL parameter or through stored page data
- What does this allow us to do?
  - Control the content someone is looking at (great for phishing!)
  - Navigate someone off of the page they're looking at
  - Interact with the current user's cookies or perform actions as the user on the server

```
<script>alert('hello world')</script>
```

Injection - Path Traversal

- Navigating to a file that isn't expected by the server
  - Use this to read from (or write to!) sensitive files
  - Application config files, database files, `/etc/passwd` for system users, `.ssh/` for ssh keys

https://example.com/viewpdf?pdf=/about_us.pdf

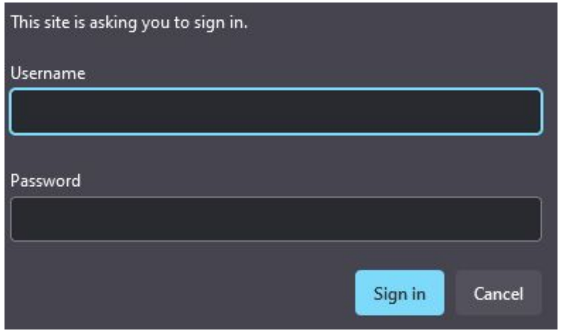
```
$file = $_GET['pdf']
$content = read($file)
echo $content
```

https://example.com/viewpdf?pdf=../../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Access Issues - Weak/Default Credentials

- This is WAY more common than it should be
- With many services, the default credentials can be looked up
  - Usually admin/admin, admin/password, admin/<empty>


This site is asking you to sign in.

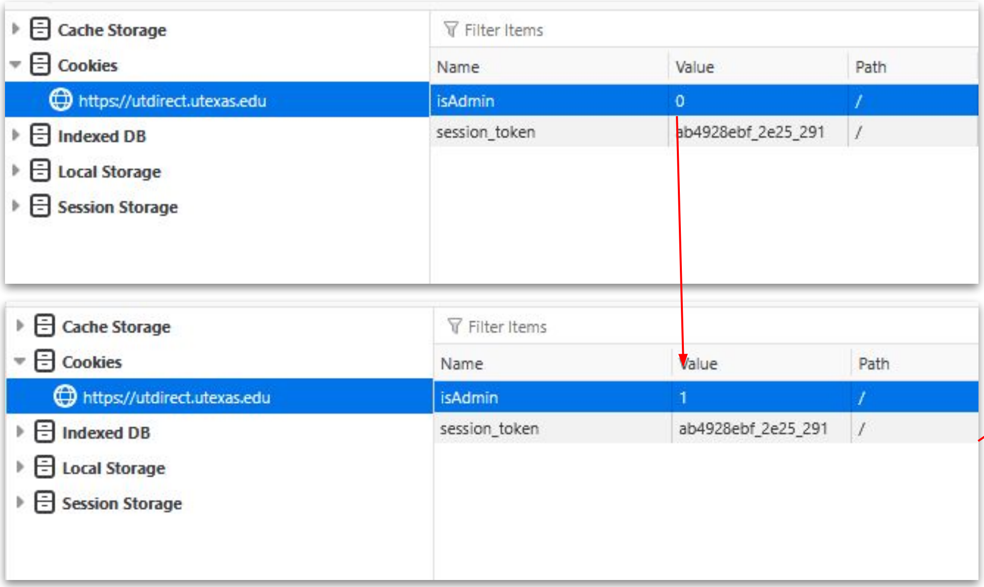
Username

Password

User: admin
Password: admin1

Access Issues - Insecure Session Control

- We've already taken a look at this (cough UTCS cough)
- Many different ways to go wrong
    - Try decoding the session token and modifying data within it

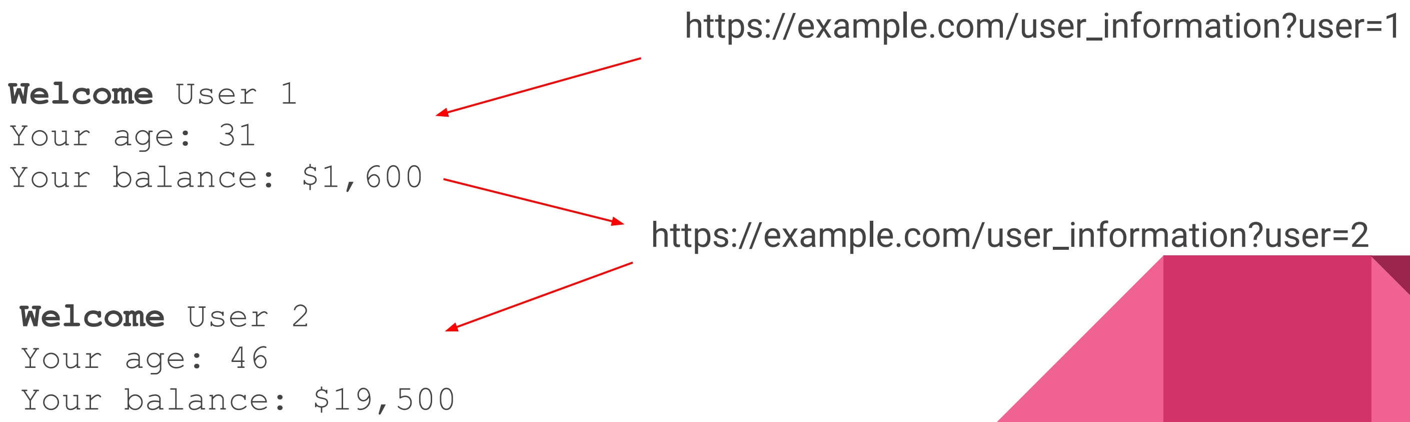


Name	Value	Path
isAdmin	0	/
session_token	ab4928ebf_2e25_291	/

Name	Value	Path
isAdmin	1	/
session_token	ab4928ebf_2e25_291	/

Access Issues - Insecure Direct Object Reference (IDOR)

- Scary-sounding, easy in practice
  - Scarily common, too
- Accessing objects directly!
  - Are you user 1? Try and access user 2 or 3!
  - Looking at the receipt for order #4023? Try #4022!



https://example.com/user_information?user=1

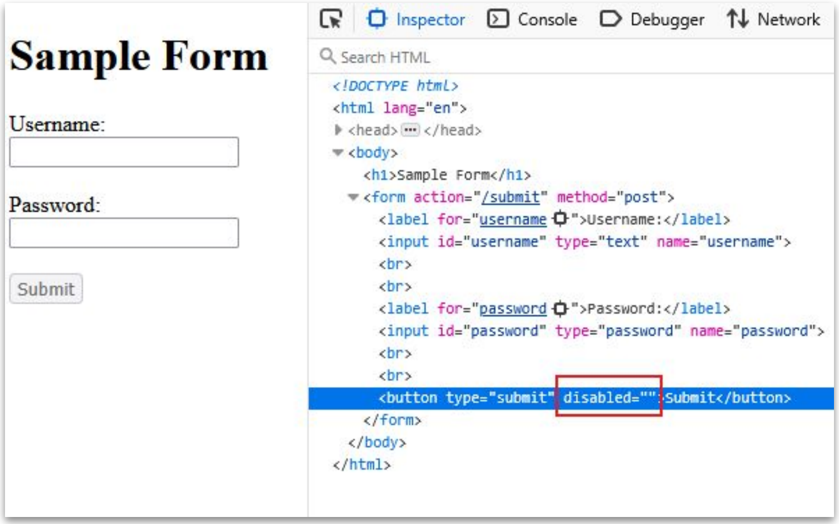
```
Welcome User 1
Your age: 31
Your balance: $1,600
```

https://example.com/user_information?user=2

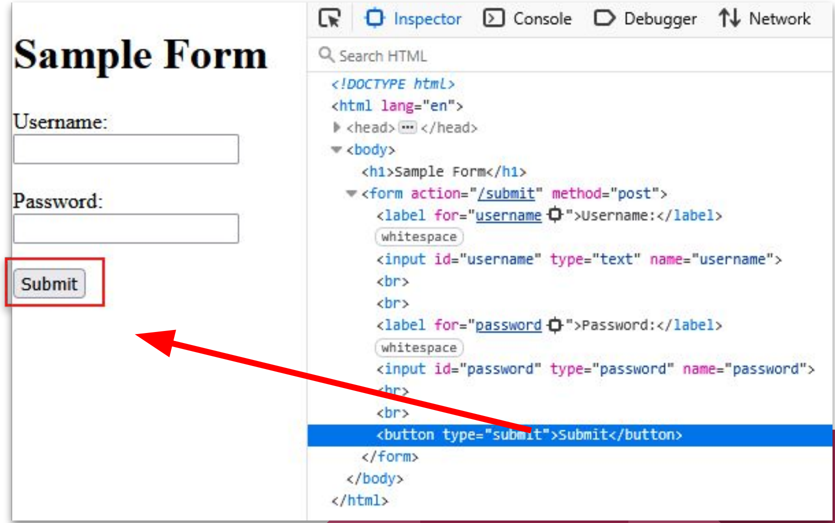
```
Welcome User 2
Your age: 46
Your balance: $19,500
```

Workflow - Client-side Protections

- Removing HTML/JS client-side prevention mechanisms



```
<!DOCTYPE html>
<html lang="en">
  <head> … </head>
  <body>
    <h1>Sample Form</h1>
    <form action="/submit" method="post">
      <label for="username">Username:</label>
      <input id="username" type="text" name="username">
      <br>
      <br>
      <label for="password">Password:</label>
      <input id="password" type="password" name="password">
      <br>
      <br>
      <button type="submit" disabled="">Submit</button>
    </form>
  </body>
</html>
```

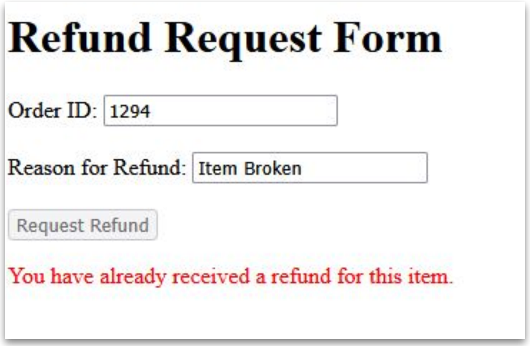


```
<!DOCTYPE html>
<html lang="en">
  <head> … </head>
  <body>
    <h1>Sample Form</h1>
    <form action="/submit" method="post">
      <label for="username">Username:</label>
      whitespace
      <input id="username" type="text" name="username">
      <br>
      <br>
      <label for="password">Password:</label>
      whitespace
      <input id="password" type="password" name="password">
      <br>
      <br>
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

Workflow - Functionality Abuse

- Developers won't always cover all of the edge cases
  - What happens if you supply invalid data?
  - Try passing a negative number, a string, or some malformed data!
- Pairs nicely with bypassing client-side protections
  - Developer code might avoid test cases due to client-side protections

https://example.com/api/v1/refund_item?user=1&item=1294

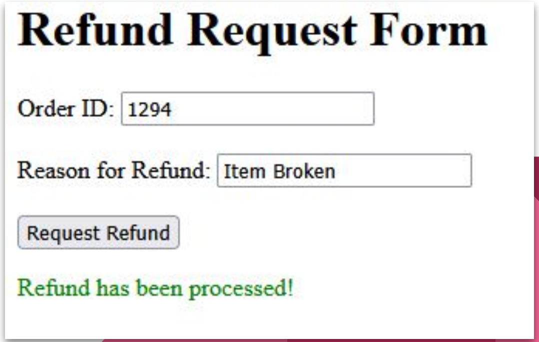


Refund Request Form

Order ID: 1294

Reason for Refund: Item Broken

You have already received a refund for this item.



Refund Request Form

Order ID: 1294

Reason for Refund: Item Broken

Refund has been processed!

Some Other Common Vulnerabilities

- Outdated Components
  - You don't always have to hack the app itself! Sometimes you can target what it's running on
  - Look for service version numbers, and search them up
- Insecure Configurations
  - Server running in debug mode
  - Insecure functions used
  - Allowing weak user passwords
- Secrets laying around
  - Take a peek at old github commits
  - Check out source code
  - https://github.com/praetorian-inc/noseyparker

khaelkugler.com/notes.html has explanations and exploitations of all of these :)

Workshop Time (cc food?)

http://34.57.194.166/

- Check for injection issues!
  - Command injection, SQL injection, XSS, path traversal
- Check for access issues!
  - Bad passwords, insecure session tokens, stealing tokens with JavaScript
- Check for workflow issues!
  - Steal money, bypass client-side restrictions
- No restrictions! Go crazy 😈
  - Mess with other users, steal money, and take the server for yourself
  - Increase impact wherever possible

Source code

github.com/KhaelK138/InsecureWebApp

Hacking Notes

khaelkugler.com/notes.html

Thank you!