Systems and Software
Verification Laboratory

UFAM

MANCHESTER 1824
The University of Manchester

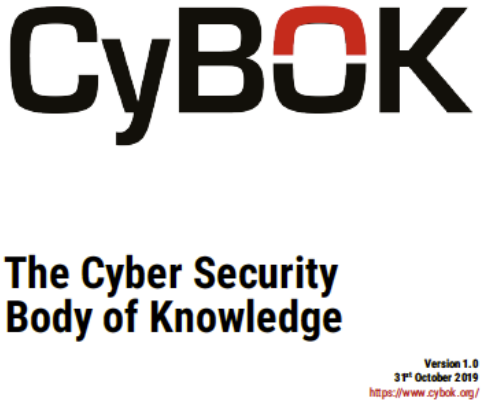
Mobile and WebSecurity: Overview

Rafael Menezes
UFAM / PPGI
rafael.sa@icomp.ufam.edu.br

Lucas Cordeiro
UoM / Department of Computer Science
lucas.cordeiro@manchester.ac.uk

Secure Mobile Programming

- Lucas Cordeiro (Formal Methods Group)
  - *lucas.cordeiro@manchester.ac.uk*
  - Office: 2.28
  - Office hours: 15-16 Tuesday, 14-15 Wednesday
- Textbook:
  - *The Cyber Security Body Of Knowledge – Web and Mobile Security* (Chapter 15)



Intended learning outcomes

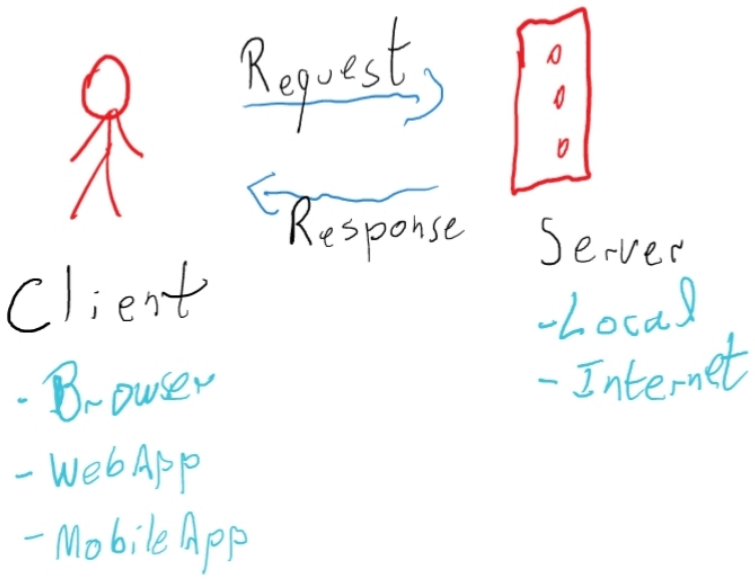
- Review **HTTP protocol** and its extensions
- Understand the **similarities** between mobile and web applications
- Understand flaws for **user authentication**
- Learn **Security Policies** for web applications
- Identify **client and server** vulnerabilities
- Construct **secure web applications** with sandboxing and policies

Mobile and Web development

- Many of the applications are not native applications written in compiled programming languages such as Java or Kotlin and C/C++. These apps are based on web technologies. Instead, they are based on web technologies including server-side Python, Ruby, Java or JavaScript scripts and client-side JavaScript

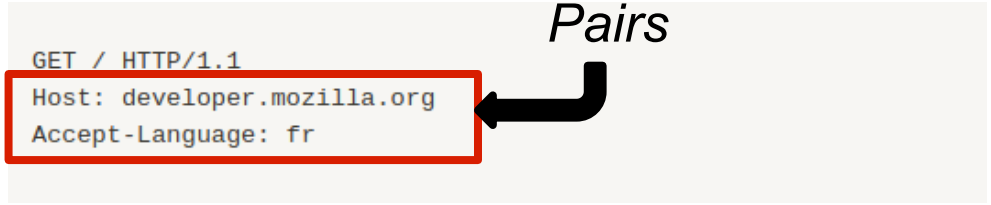
Hypertext Transfer Protocol

- The **Hypertext Transfer Protocol** (HTTP) is a text-based protocol using TCP/IP
  - A client initiates a session by sending an HTTP request to an HTTP server
  - The server returns an HTTP response with the requested file



HTTP Request/Response

- A request includes the http version along with **zero or more pairs of *Name:Value***
  - This request may contain information about the client such as **cookies** and its **web browser**




```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the request
```

HTTP Codes

- RFC 2616 and 7231 defines codes to indicate how the request was handled

Range	Meaning
100-199	Informational Response
200-299	Successful Response
300-399	Redirects
400-499	Client Errors
500-599	Server Errors

Request For Comments (RFC)

- Request for Comments (RFC) are documents that contain technical notes about the Internet
- RFC cover a range of topics about computer networks, including HTTP and its extensions, notably:
  - RFC 2616: Hypertext Transfer protocol – HTTP/1.1
  - RFC 2617: HTTP Authentication
  - RFC 2109: State Management (cookies)

Uniform Resource Locators (URL)

- URL is a text string that addresses and identifies resources on a server in the form:

scheme://credentials@host:port/resourcepath?query_parameters#frame

https://www.google.com/search?client=firefox-b-d&q=example+https+url

Google search URL

Uniform Resource Locators (URL)

scheme://credentials@host:port/resourcepath?query_parameters#frame

- **scheme:** Protocol used e.g http, https, ftp
- **credentials (optional):** user:password followed by @
- **host:** IP or DNS of the server

Uniform Resource Locators (URL)

scheme://credentials@host:port/resourcepath?query_parameters#frame

- **port (optional):** Port used, if left blank a default port is used based on the service e.g., 80 (HTTP), 443 (HTTPS)
- **resourcepath:** The location of the resource using unix notation, beginning from **/**

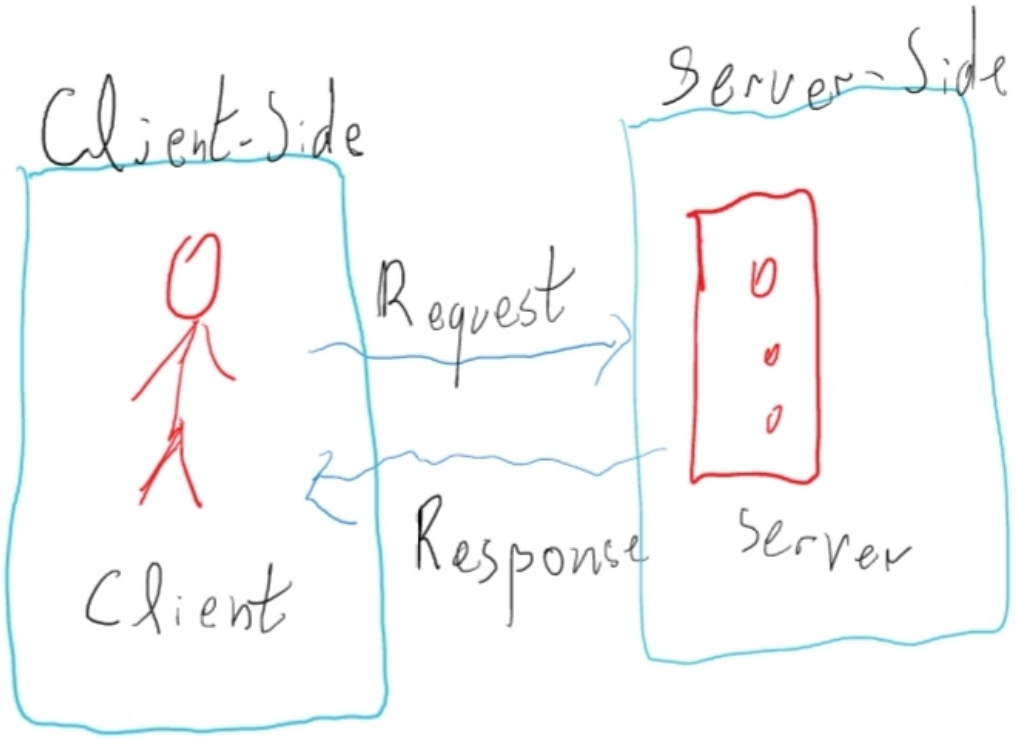
Uniform Resource Locators (URL)

scheme://credentials@host:port/resourcepath?query_parameters#frame

- **query_parameters:** Parameters passed to the server using *name=value* notation
- **frame:** Extra information used for clients e.g. put the web page into a specific header

Web Development

- **Web applications** are split into two main parts: **client-side** and **server-side** using HTTP

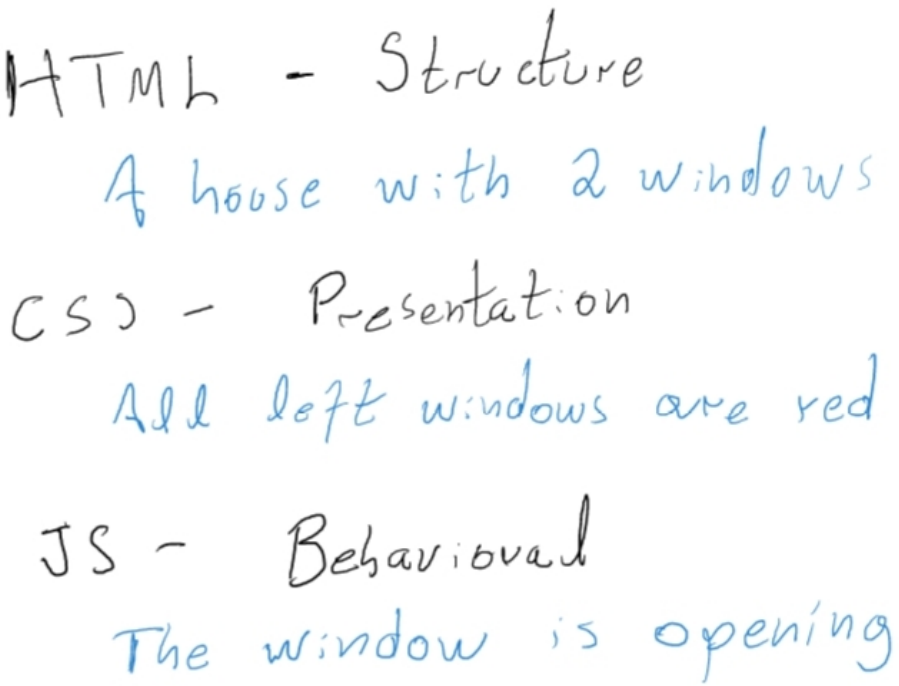


Web Development

- Client-side consists of the interactions happening on the client
  - In the web, this client may be the web browsers which renders the HTML, JS and CSS
- Server-side consists of interactions happening on the server
  - In the web, this is implemented using a web server in conjunction with a framework

Client-Side

- For web clients, the web browser renders the **page layout (HTML)**, the **page logic (JavaScript)** and the **page style (CSS)**



Hypertext Markup Language (HTML)

- HTML is a hierarchical tree structure of tags, attributes and text
- The **Domain Object Model (DOM)** defines the logical structure of a HTML document

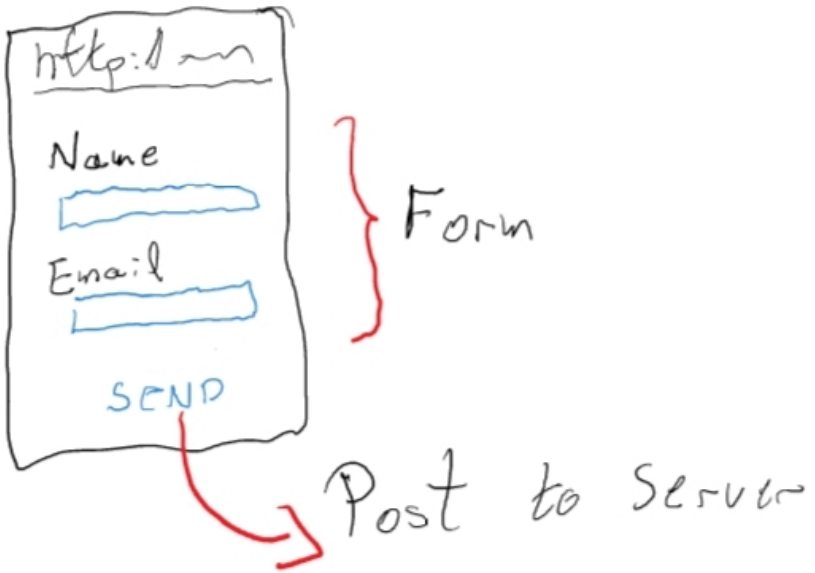
```
<!doctype html>

<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>hello world!</title>
    <meta name="description" content="html">
    <meta name="author" content="Rafael">
    <link rel="stylesheet" href="css/styles.css">
  </head>

  <body>
    <script src="js/scripts.js">
    </script>
  </body>
</html>
```

WebAssembly

- It is a stack-based virtual machine language and mainly aims to execute at native speed on client machines. Code written in WebAssembly is memory safe and benefits from all security features provided by regular code associated with a website. WebAssembly code is sandboxed, enforces the same origin policy.

Web Client Interaction

- The basic flow of a web application is to send a HTTP Request to a server and it will reply with the HTML file



Mobile Development

- Mobile Applications also follow a similar flow of web applications in the sense that the app is the client instead of the browser

Application Stores

- Application stores are centralized digital distribution platforms that organize the management and distribution of software in many web and mobile ecosystems.
- Application stores allow providers to control which applications are available in their stores, which allows them to ban applications.

Application Stores

- Deployed security vetting techniques include static and dynamic analysis applied to application binaries and running instances of applications. In addition to security vetting techniques, application stores require applications to be signed by developer or application store keys.

Sandboxing

- Both modern mobile and browser platforms make use of different sandboxing techniques to isolate applications and websites and their content from each other

Application Isolation

- Modern mobile platforms provide each application with their sandbox running in a dedicated process and their own filesystem storage. Mobile platforms take advantage of underlying operating system process protection mechanisms for application resource identification and isolation.

Content Isolation

- Content isolation is one of the major security assurances in modern browsers.The main idea is to isolate documents based on their origin so that they cannot interfere with each other.
- The Same-Origin-Policy (SOP) affects JavaScript and its interaction with a document's DOM, network requests and local storage. The core idea behind SOP is that two separate JavaScript execution contexts are only allowed to manipulate a document's DOM if there is an exact match between the document host and the protocol, DNS name and port numbers

Cookies

- Web servers can associate stateful information with particular clients by using HTTP cookies. Cookie information is stored by the client. Allowing clients and servers to include their unique session identifiers in each HTTP request-response, avoiding the need for repeated authentication.

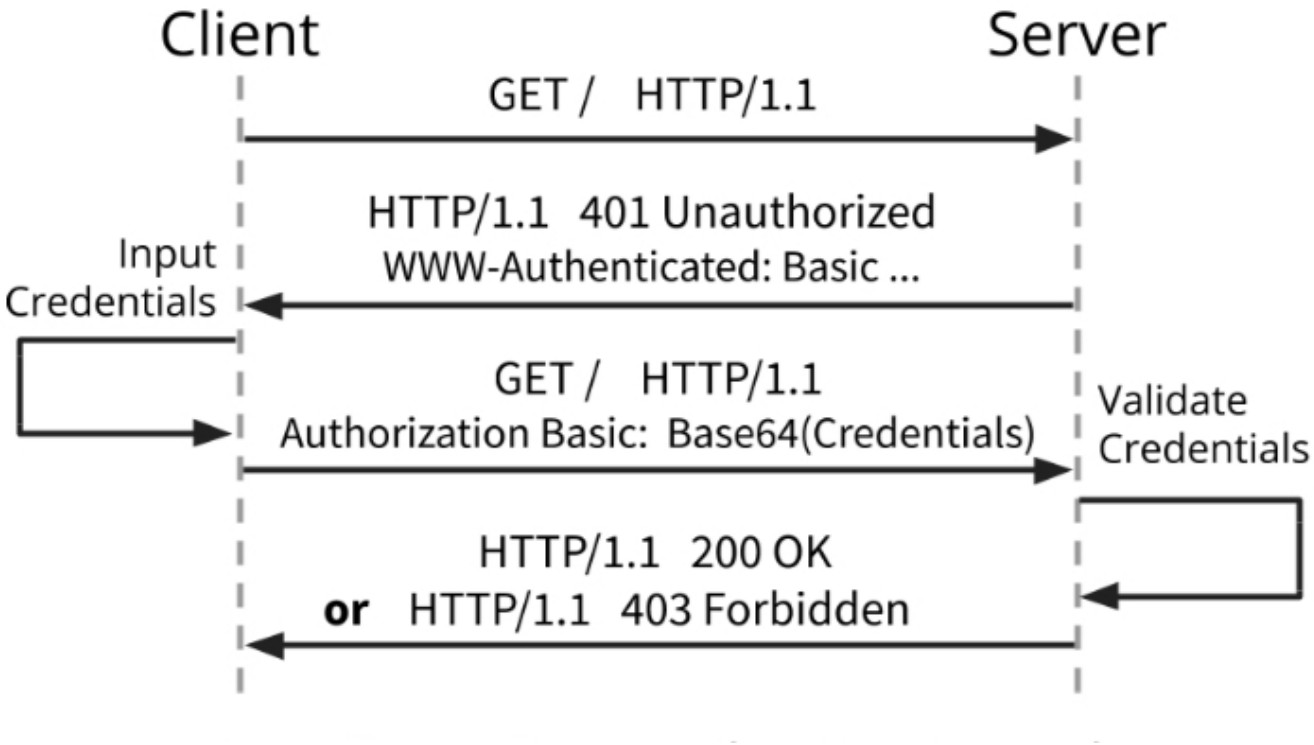
Authentication

- Authentication is a **security mechanism** designed to enable users to **assert** their **identity** on applications
- The most common for authentication are:
  - Passwords
  - Tokens
  - PIN's
  - Patterns
  - Devices and biometric features
- For web, iana maintains a list of Authentication Schemes

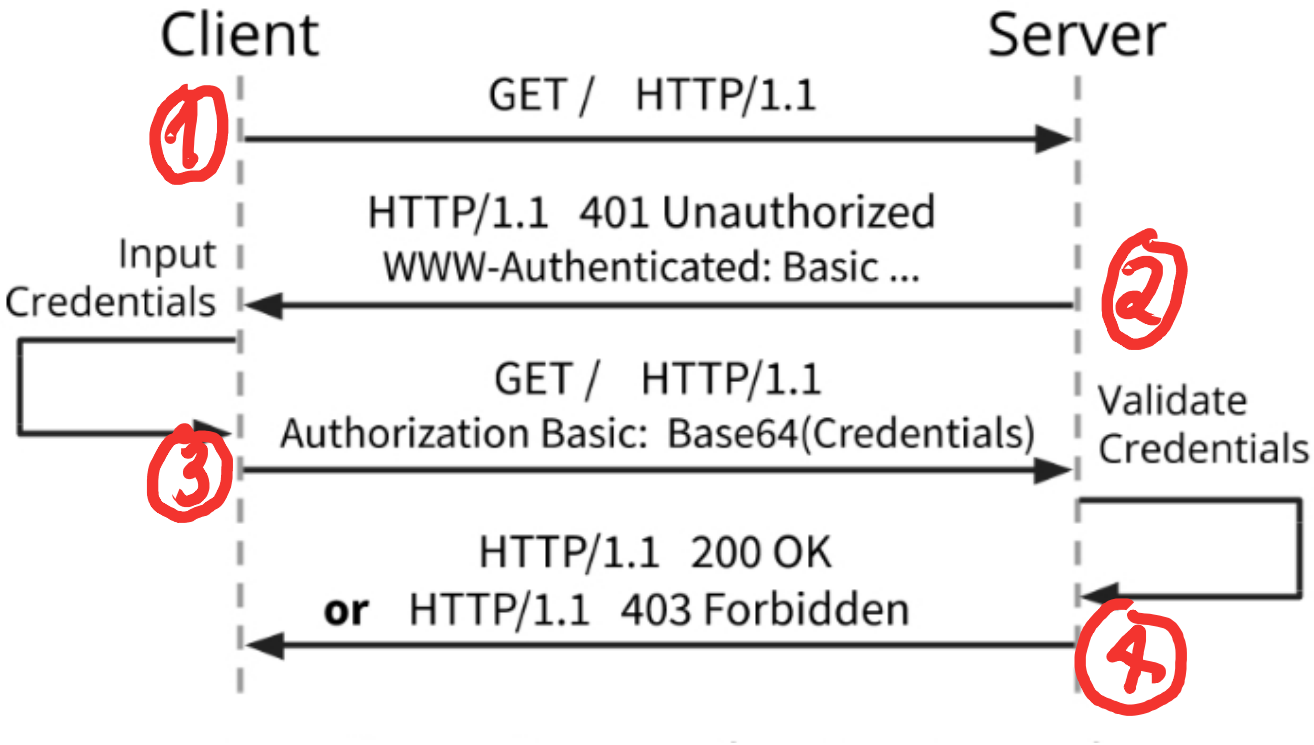
HTTP Basic

- Defined in RFC 7617 is a simple way to authenticate the user by sending the credentials through parameters
- The credentials are transmitted as user-id/ password pairs, encoded with Base64
- The communication *must* be secure (encrypted).

HTTP Basic

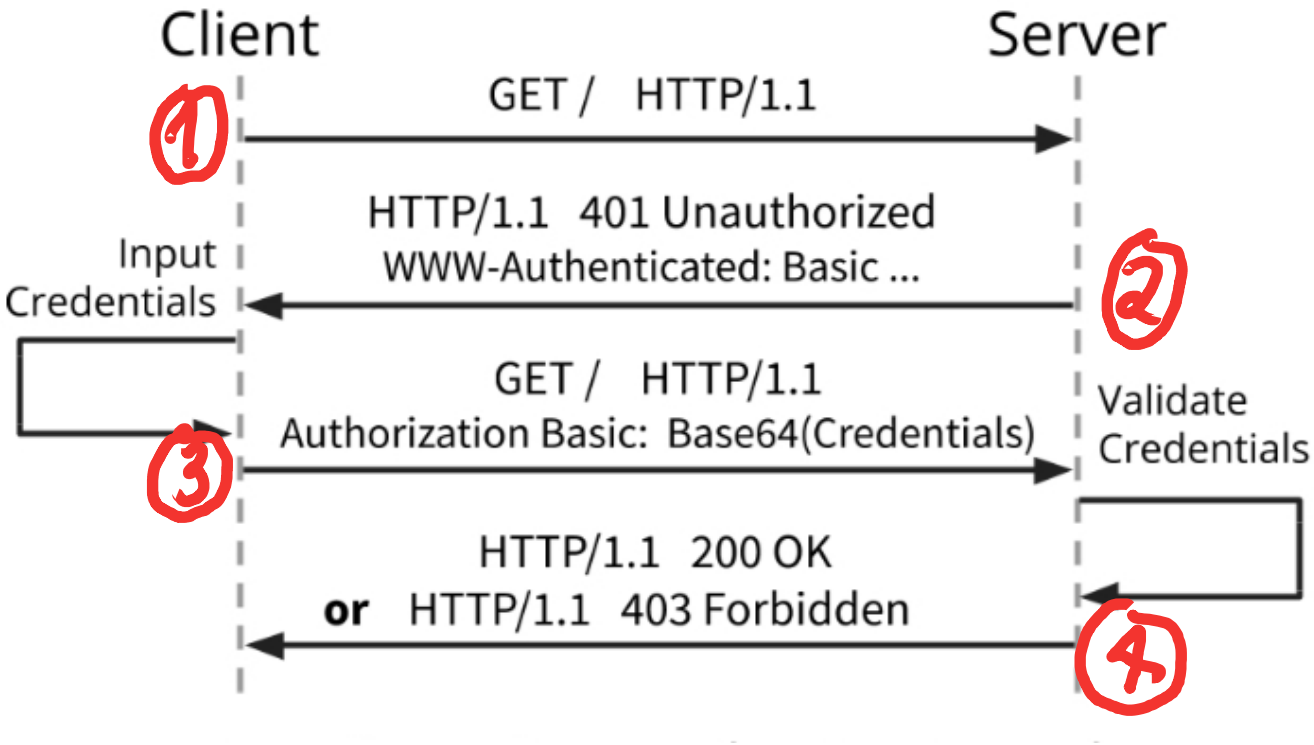

Client
Server
GET / HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticated: Basic ...
Input
Credentials
GET / HTTP/1.1
Authorization Basic: Base64(Credentials)
Validate
Credentials
HTTP/1.1 200 OK
or HTTP/1.1 403 Forbidden

HTTP Basic



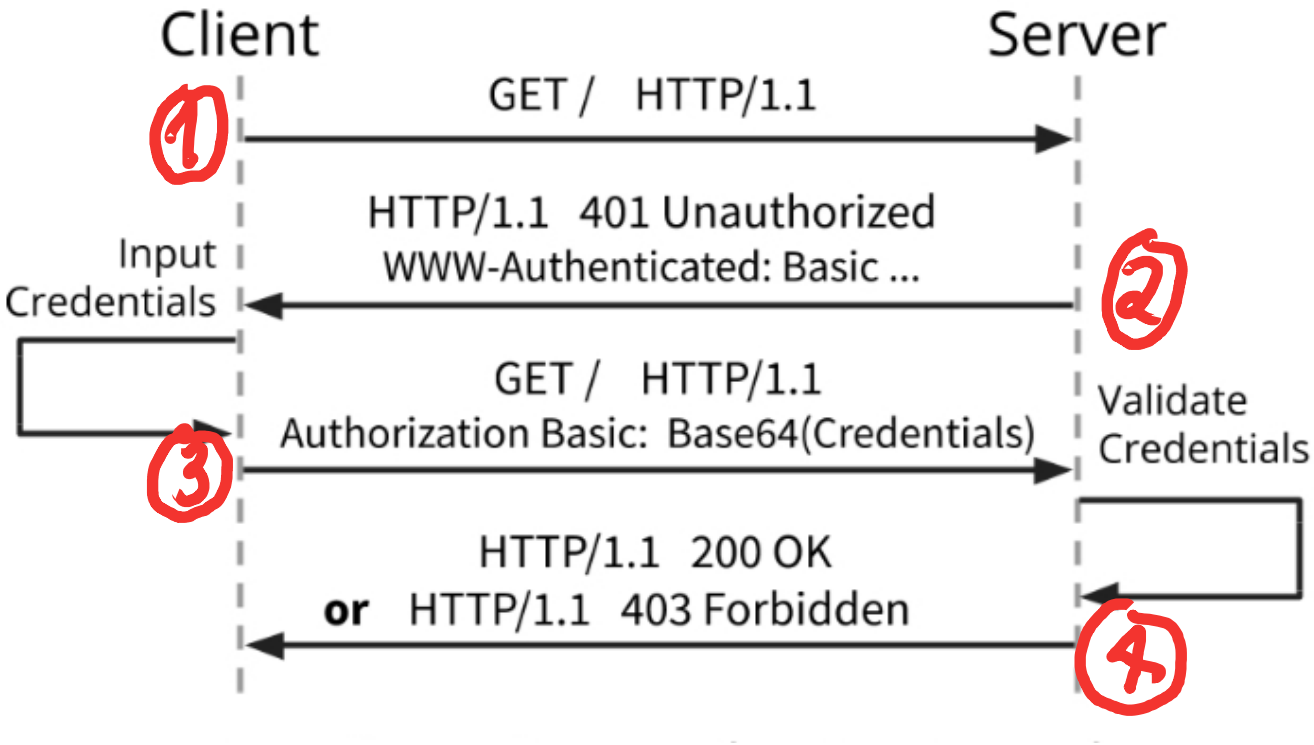
1: The client asks for a protected resource

HTTP Basic



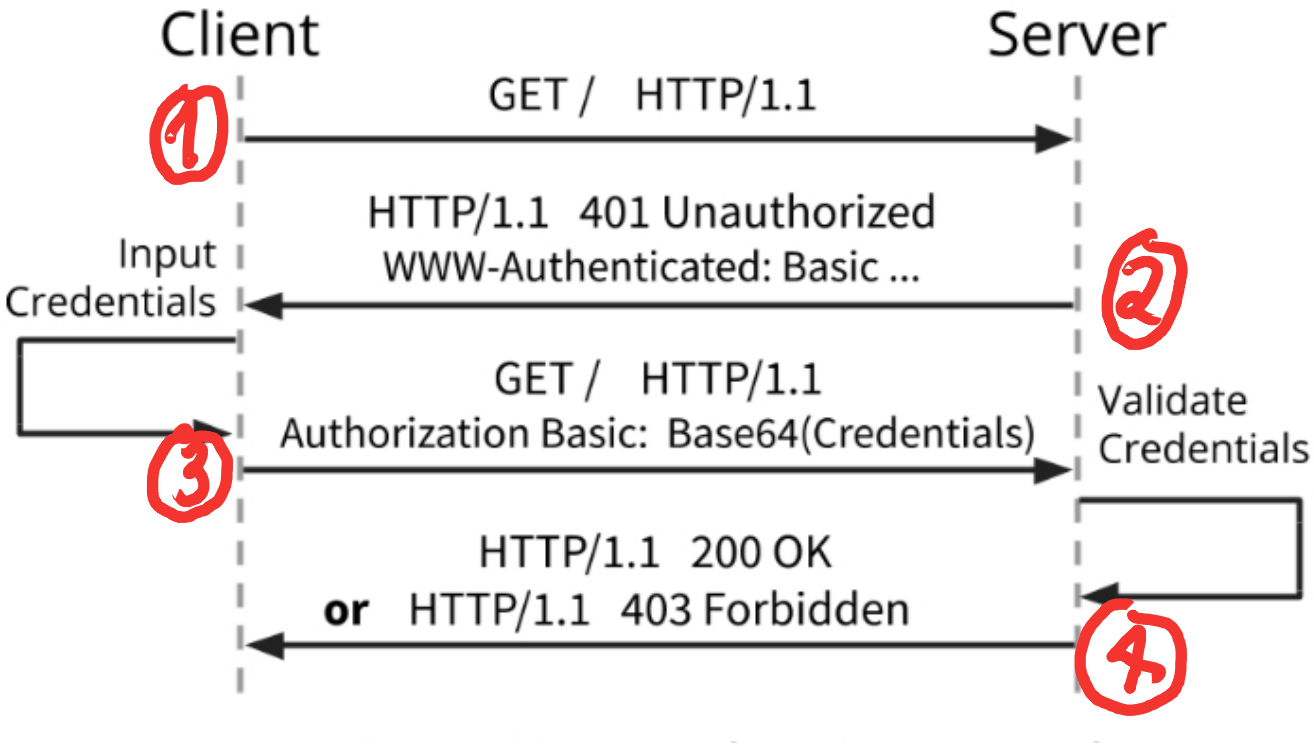
2: The server sends an unauthorized response (code 401) to the client

HTTP Basic



3: The client adds it's credentials into the request

HTTP Basic



4: The server validates the credentials and returns with the content requested or a forbidden status

HTTP OAuth 2.0

- OAuth enables third-party applications to obtain limited access to an HTTP service by orchestrating an approval between the resource owner and the HTTP service using secure token instead of login credentials
- Defined in RFC 6749

OAuth 2.0 Roles

- RFC 6749 defines the following roles

Role	Meaning
Resource owner	An entity capable of granting access to a protected resource
Resource server	The server hosting the protected resources,
Client	An application making protected resource requests on behalf of the resource owner and with its authorization
Authorization server	The server issuing access token to the client after successfully authenticating the resource owner and obtaining authorization

OAuth 2.0 Flow

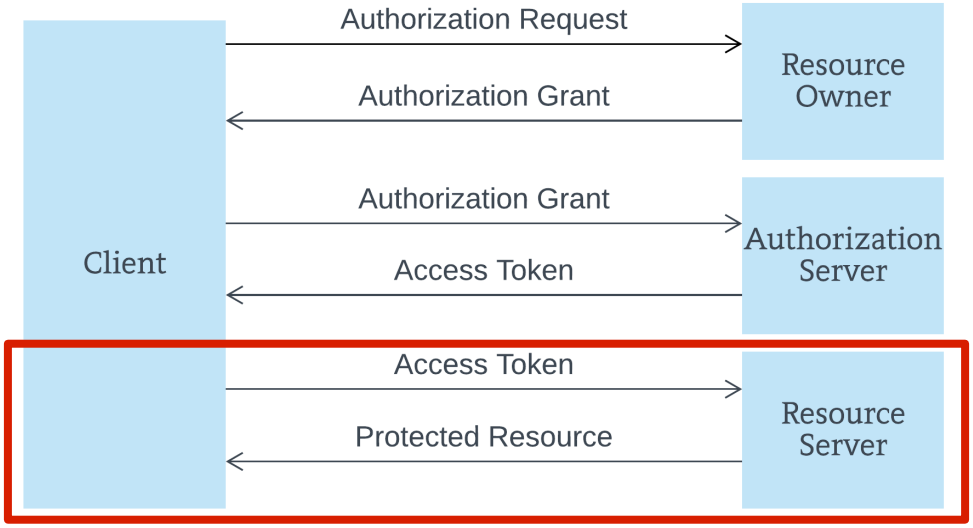
- The client request authorization from the resource owner
  - This can be made directly to the resource owner, or via the authorization server
  - The client then receives an authorization grant

OAuth 2.0 Flow

- The client requests an access token by authenticating with the authorization server and presenting the authorization grant
- If the grant is valid, then the client receives an access token

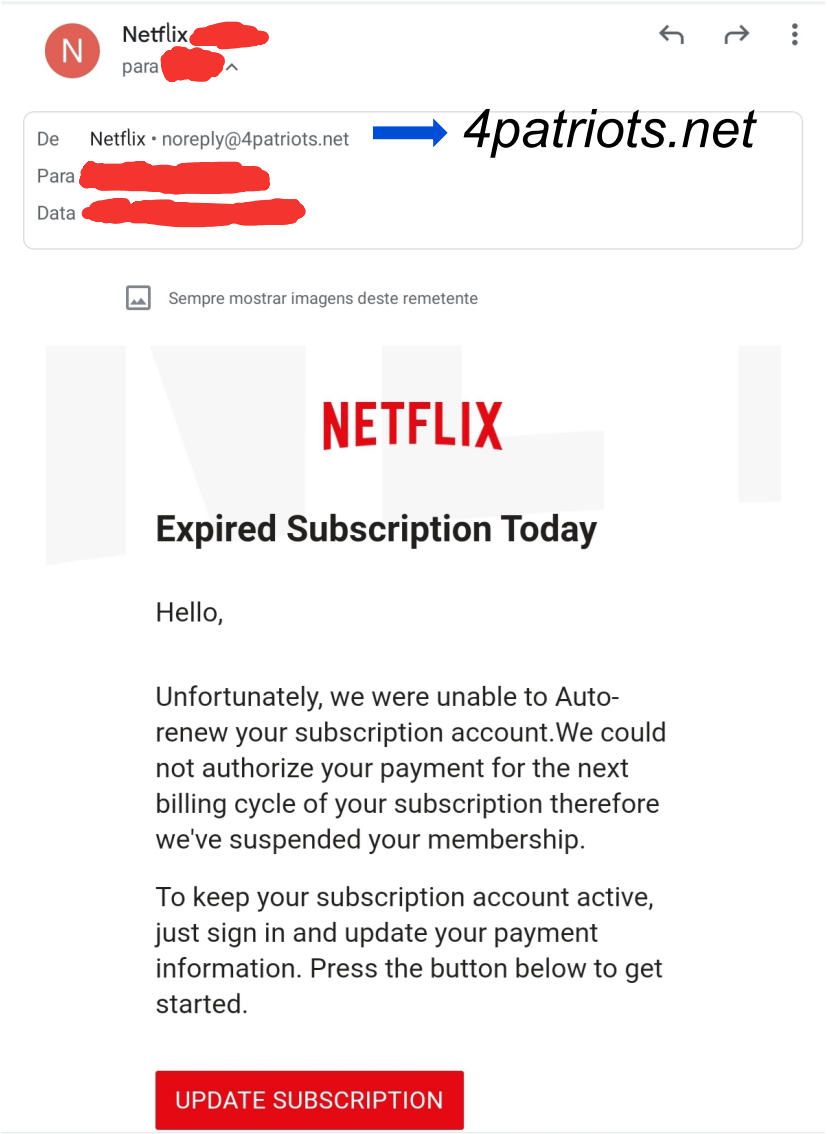
OAuth 2.0 Flow

- The client requests the protected resource from the resource server using the token
- If the token is valid, then the server returns the content



Phishing

- Attacks by disguising fraudulent service as a genuine
- The trick consists of making the victim believe that is interacting with the genuine site by manipulation of email and URLs



Clickjacking

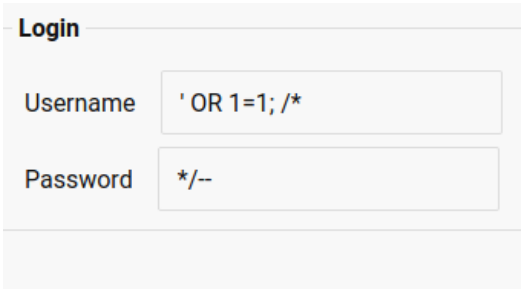
- This attack consists of manipulating the visual appearance of a website and trick users to click into a transparente or opaque layer over original websites
- It can be used to launch other attacks such as **Cross-Site Request Forgery**

SQL Injection

- Many web applications allow users to enter information through forms, and from which the developer may use

SQL Injection

- If the developer does not sanitize the information when generating the SQL command the attacker may add statements.



```
SELECT * FROM users
WHERE user = '' OR 1=1; /* '
AND password = '*/--'
```

SQL Injection

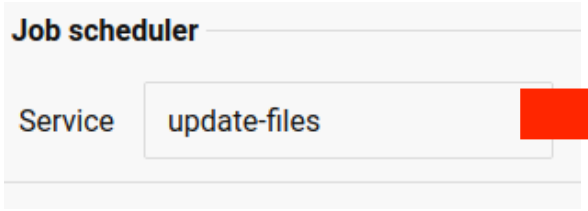
- To mitigate this flaw, the developer should use prepared statements instead of embedding user input into SQL statements

  *SELECT * FROM users WHERE user = ? AND password = ?*

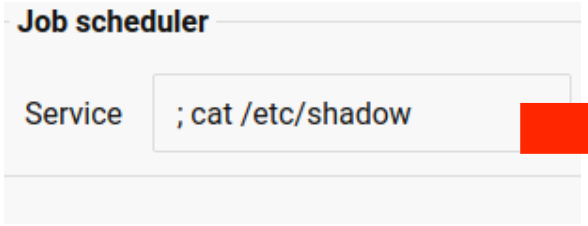
- The developer must always sanitize user input against not expected inputs, symbols and using specialized libraries

Command Injection

- Similar to SQL injection, if the values passed by the user are directly used as a script, an attacker may pass malicious code that will be executed
- Example: a job-scheduler service for unix that returns the log of the command.



/usr/local/bin/scheduler update-files



/usr/local/bin/scheduler ; cat /etc/shadow

Cross-Site Scripting (XSS)

- XSS enables attackers to inject malicious scripts (JavaScript) into benign websites
- This occur when users are able to submit scripts that may be executed by other end-users.
  - Such as message forums where users receive message from each other.
- Since the website itself is propagating the script, the client browser cannot detect the malicious code

Cross-Site Scripting (XSS)

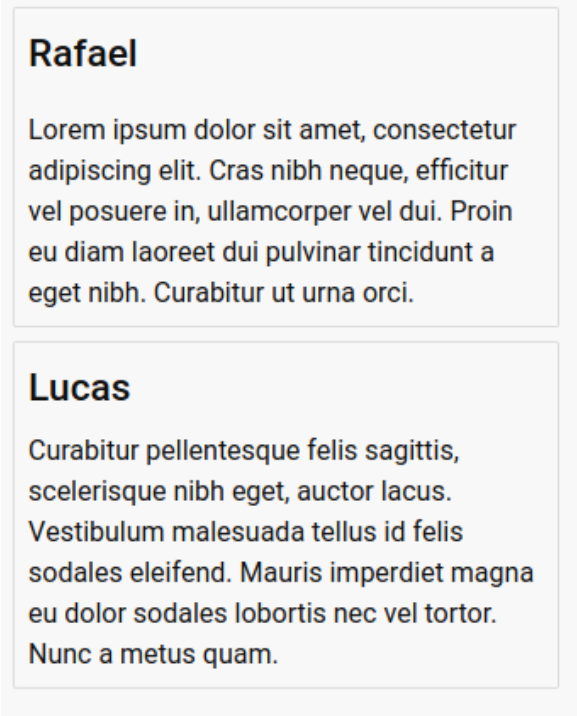
- **Stored**: when the XSS attack is permanently stored on the target server.
  - Also called Type-I XSS

- **Reflected:** the script is not permanently stored on the server, but reflected by the server such as a link to a malicious script.
  - Also called Type-II XSS

Cross-Site Scripting (XSS)

- **Example:** suppose that an application generates a page using saved user comments:

```
<div class="user_comment_box">
  <h3>$user.name</h3>
  <p>$user.comment</p>
</div>
```

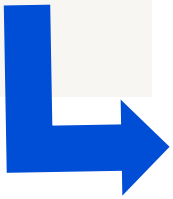


Cross-Site Scripting (XSS)

- What would happen if an attacker added a script into the comment?

```
This is a genuine Type-I comment!
<script>
  var el = document.createElement('div');
  el.innerHTML = '<p>I am running this on the client!</p>';
  document.body.appendChild(el);
</script>
```

```
This is a genuine Type-II comment!
<script
  src="https://mydomain.com/script/attack.js">
</script>
```



Rafael

This is a genuine Type-II comment!

Lucas

This is a genuine Type-I comment!

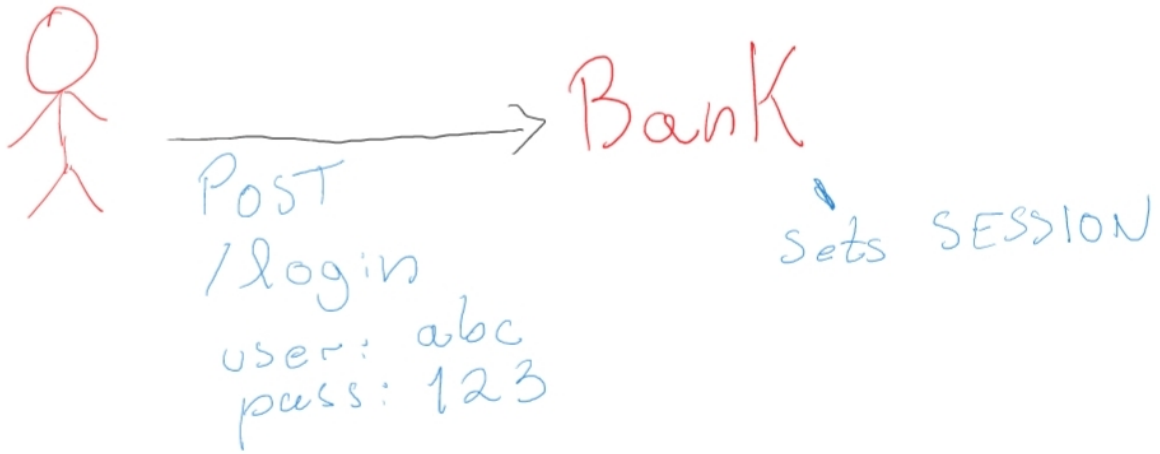
I am running this on the client!

Cross-Site Request Forgery (CSRF)

- CSRF attacks mislead victims into submitting malicious HTTP requests to remote servers
- The remote server will think that the user itself is making a request through genuine means, and will use any cookies available
- The attacker may use this to create a request to post in a social media or transfer from the bank account of a victim

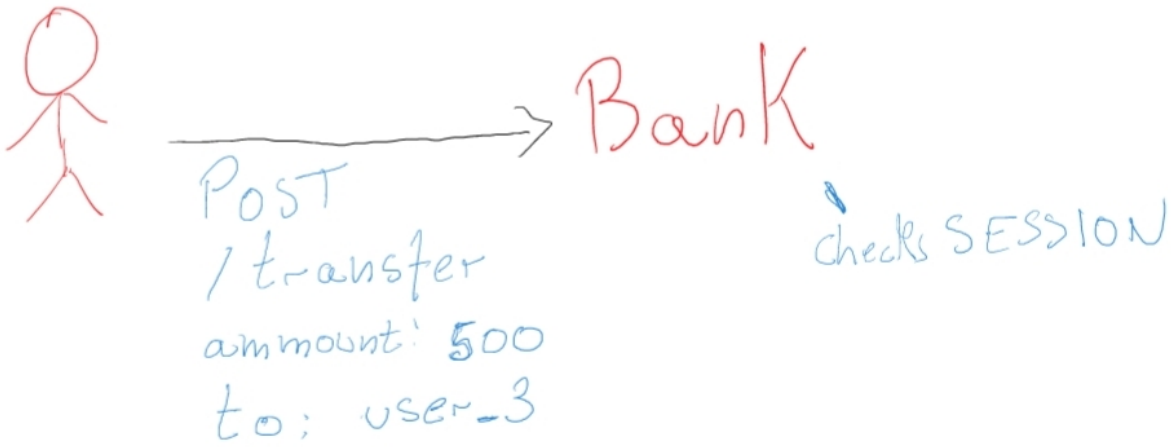
Cross-Site Request Forgery (CSRF)

- **Example:** Banking, assume that after logging into your baking account the web application maintains the session with your identity



Cross-Site Request Forgery (CSRF)

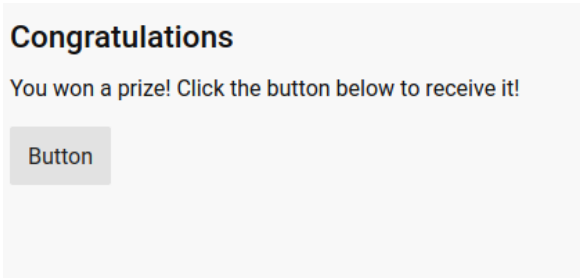
- Now, you can do banking operations without having to reauthenticate with HTTP requests



Cross-Site Request Forgery (CSRF)

- Third-party apps can take advantage of this

```
<h3>Congratulations</h3>
  <p>You won a prize! Click the button below to receive it!</p>
  <input type="button" value="Button" id="bt" onclick="transferToMe()"/>
  <script>
      function transferToMe() {
          xhttp.open("POST",
                     "https://bank.com/transfer",
                     true);
          xhttp.setRequestHeader("Content-type",
                                 "application/x-www-form-urlencoded");
          xhttp.send("ammount=1000&to=user_27");
      }
  </script>
```



Cross-Site Request Forgery (CSRF)

- To prevent CSRF attacks, it is recommended to include random tokens in request which are unique per sessions and using a rng in order to prevent attackers from predicting them.

Firewall

- To protect a webserver, a firewall should be configured to only allow access from outside where access is needed. Access should be limited to ports like 80 and 443 for HTTP requests via the Internet and restricting system configuration ports for SSH and alike to the internal network.

Load Balancers

- Load balancers control HTTP traffic between servers and clients and provide additional access control for web application resources. They can be used to direct requests and responses to different web servers or ports, balance traffic load between multiple web servers and protect areas of a website with additional access control mechanisms.

Load Balancers

- Load balancers can also serve for rate limiting purposes. They can limit request size, allowed request methods and paths or define timeouts. The main use of rate-limiting is to reduce the potentially negative impact of denial of service attacks on a web server and prevent users from spamming systems, as well as restrict and prevent unexpected behavior

Load Balancers

- Additionally, load balancers can be used to provide secure TLS connections for web applications. When managing TLS, load balancers serve as a network connection endpoint for the TLS encryption and either establish new TLS connections to the application service or connect to the web application server using plain HTTP

Summary

- Web applications: HTTP, URL, CSS, HTML, JavaScript.
- Client flaws: Phishing and Clickjacking.
- Server flaws: SQL Injection, Command Injection, XSS and CSRF.