Web security

Dr. Tuğberk Kocatekin

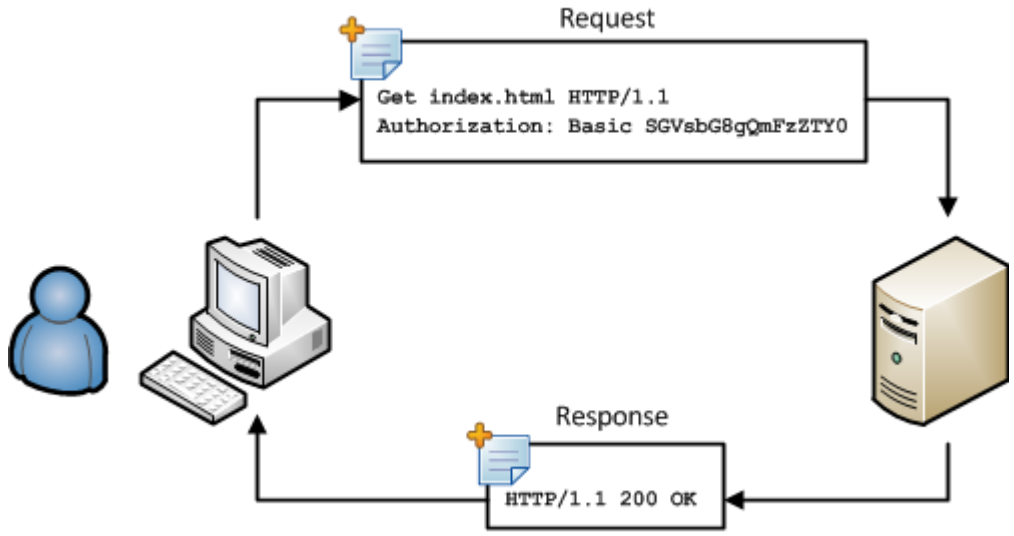
Istanbul Arel University

What is web?

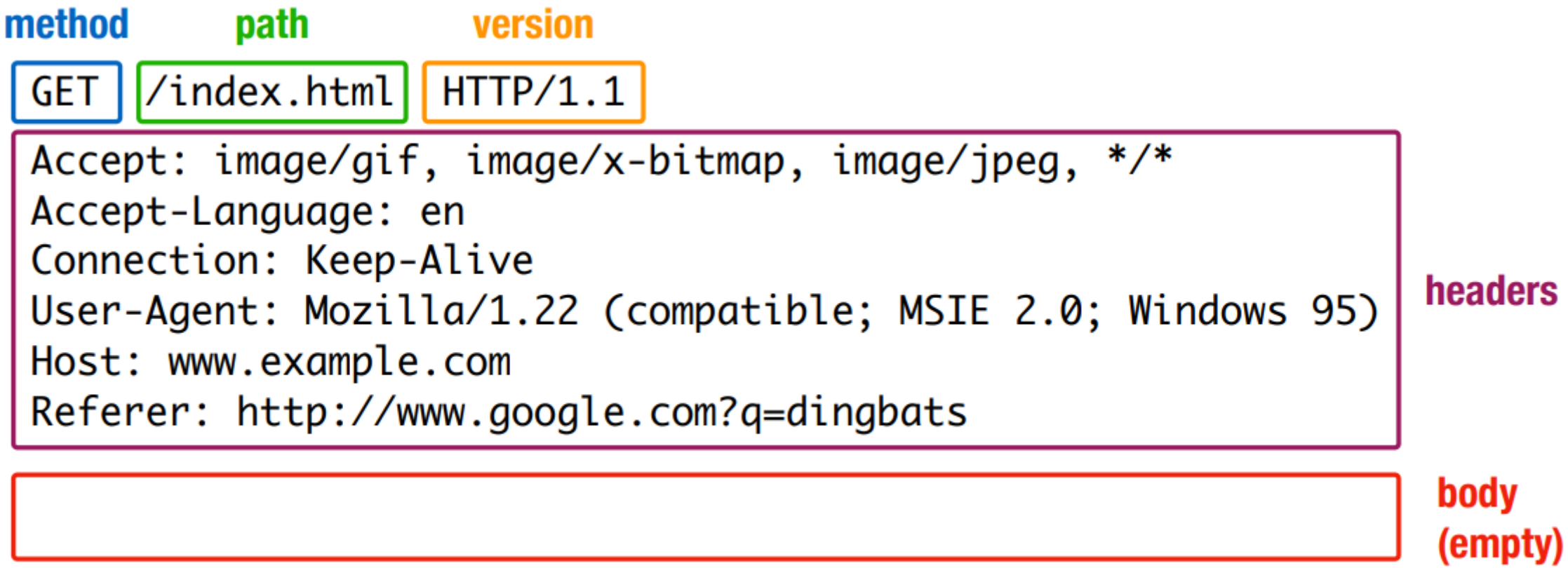
- Not only websites.
- Almost everything lives in the web.

HTTP: Hyper text transport protocol

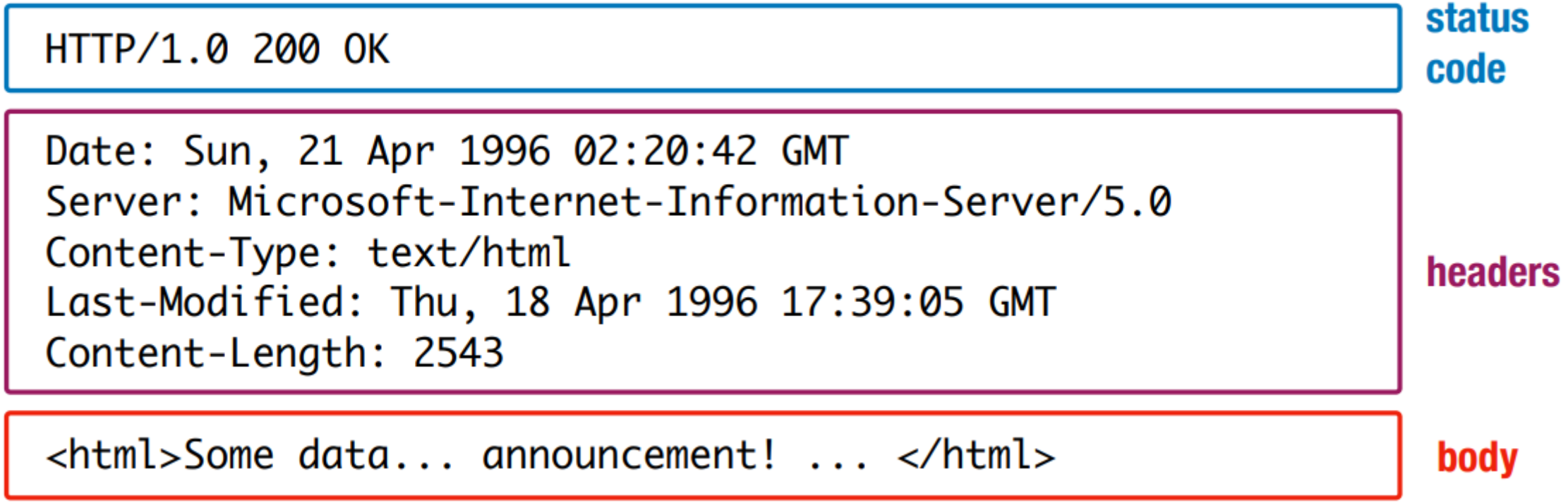
- A protocol which allows fetching HTML files from a webserver.
  - Request and response



HTTP Request



HTTP Response



HTTP: GET & POST

- While building the form, you provide the method you want to send request. Most used HTTP methods are:
  - POST: Mostly used to **create** new resources.
  - GET: Used to **read or retrieve**. (Should never modify resources on the server)
  - PUT: Used to **update** capabilities.
  - PATCH: Used to **modify** capabilities.
  - DELETE: Used to **delete** a resource.
- Old browsers did not support methods other than POST and GET.

HTTP (cont.)

- When you want to read data from the server, use GET. While we are simply surfing, we are using GET.
- We can also submit forms to websites using POST.

- HTTP is **stateless**: Each request is executed independently. It has no knowledge of previous requests that were executed.

Javascript

- A scripting language.
- Works on client side.
- Codes run inside the browser.

Cookie

- A small file storing information.
- Logins, scores, session states, user preferences, themes, analysis and recording of user data.
- Never store credentials.

Sessions

- When you log in, server provides a random seed and stores it in the cookie.
- Later when you make requests with that cookie information, the server knows who you are.

TLS & HTTPS

- Used to be SSL, now TLS. S stands for Secure.
- Always use HTTPS. There is no security in HTTP.
- The packets are plaintext.
- Can be read and modified.
- Chrome marks HTTP sites as **insecure**.
- Every webpage should be HTTPS, not only Login page.

MITM: Man in the middle attack

- Attacker can disguise himself as the router
- Every package goes through him
- Can open, read and even modify packages
- Think of multiple scenarios where this can be possible (cafe, hotels, schools)
- HTTPS prevents this.

HTTPS and certificates

- Lock icon indicates that website uses HTTPS.

- When you click on the lock, you can see information about the certificate.
- You should be careful as to whether forms are sending information over HTTP instead of HTTPS.
- However, seeing a lock doesn't necessarily mean it is secure.

Cryptographic hash functions

- Should have following properties:
- 1. No reversing
- 2. No same output for two different input
- 3. A small change in input should create a big chance in output
- SHA: Secure Hash Algorithm (standart)

a: ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb
b: 3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d
ab: fb8e20fc2e4c3f248c60c39bd652f3c1347298bb977b8b4d5903b85055620603

Rainbow tables

- People start hashing every possible solution and storing it.
- Later, when a hash comes, they can check it with the rainbow table.

Storing user credentials

- Never store credentials as plaintext.
- Hashing algorithm must be a standart one (don’t use MD5)
- Instead of only hashing, use a **salt** for better security

Salting

- When an attacker retrieves the database, he will see **hash** instead of **plaintext.**
- However, by using a **rainbow table** or **brute force attack (dictionary attack)**, he can crack the password.
- Salt is generated by a *cryptographically secure function*. So, for every input, a unique hash is created.

Salting

- Here, no salting. Alice and Bob has the same password.
- However, still we have to create a unique salt. If we use the same salt, the hash will be the same. **(hash(password+salt))**
- The salt is also stored in the database. Although it is in plaintext, it makes it very hard for the attacker to create a rainbow table.

username	hash
alice	4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
jason	695ddccd984217fe8d79858dc485b67d66489145afa78e8b27c1451b27cc7a2b
mario	cd5cb49b8b62fb8dca38ff2503798eae71bfb87b0ce3210cf0acac43a3f2883c
teresa	73fb51a0c9be7d988355706b18374e775b18707a8a03f7a61198eefc64b409e8
bob	4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b
mike	77b177de23f81d37b5b4495046b227befa4546db63cfe6fe541fc4c3cd216eb9

XSS: Cross-Site Scripting

- Generally, you add certain Javascript code which executes in the client side.

- Ex:
  - You have written an entry which has XSS vulnerability. You injected JS code. Everyone who reads will run the code and you can alert stuff, etc.

Myspace Hack

- One of the most used social websites of its time.
- Allowed users to post HTML content to their pages. They escapted certain characters such as <script>, <body>, onclick, javascript: etc.
- However, you can still run Javascript inside CSS!

Preventing XSS

- Filtering is hard. You are sure to forget something.
- Using frameworks is a better way. However, don’t forget they can forget too.
- Using a whitelist approach is better than using a blacklist approach.
- It is called CSP.

CSP: Content Security Policy (CSP)

- Eliminates XSS attacks by whitelisting the trusted sources of scripts.
- Browser only executes those in whitelist and reject others.

CSRF: Cross-Site Request Forgery

- Exploits where a website transmits unauthorized commands as a trusted user.
- User is tricked into submitting an request to a website unintentionally.

- Ex:
  - A specific URL can be designed for the action the attacker wants. Let’s say the attacker wants you to delete/write something. If the website is vulnerable to CSRF, by making you click a link, that action will be done by you.

XSS vs CSRF

XSS	CSRF
You need JS.	You don’t need JS.
Code is injected in the website.	Code can be somewhere else (it may be just an URL)
If vulnerable to XSS, its also vulnerable to CSRF.	Being protected from XSS does not necessarily mean CSRF.
Attacker can do much more things.	The attack depends on the vulnerability of the URL.

CSRF Token

- There is a secret value in every form which the server can validate on submit.
- Unless you know the token,

  you can’t submit stuff from afar.

```
<!doctype html> == $0
<html>
  <head>
    <title>Laravel | CSRF Protection</title>
  </head>
  <body>
    <section>
      <h1>CSRF Protected HTML Form</h1>
      <form method="POST">
        <input type="hidden" name="_token" value=
        "6JhQN8yVuLg2dCafKw7QvCeonvYFUFuVjNQb0OL6">
        <input type="text" name="username" placeholder="Username">
        <input type="password" name="password" placeholder="Password">
        <input type="submit" name="submit" value="Submit">
      </form>
    </section>
  </body>
</html>
```

SQL Injection

- When developers try to build queries which uses data provided by the user, they may miss to escape some characters.
- This leads to attacker providing SQL data instead of plaintext data.

SQL Injection (cont.)

- HTML form has username and password.
- These are sent to server.
- Server uses these data in the SQL command.
- If the data is non-malicious, no problems.
- Else; a new SQL command (unintended) will run.

SQL Injection (cont.)

```
$login = $_POST['login'];
$pass = $_POST['password'];
$sql = "SELECT id FROM users
        WHERE username = '$login'
        AND password = '$password'";

$rs = $db->executeQuery($sql);
```

SQL Injection (cont.)

- You need to escape special characters.
- In early PHP versions mysql_ commands were vulnerable.
- Later they fixed it and provided better versions: mysqli, PDO, etc.
- You can do serious damage such as:
  - Drop Table
  - Run xp_cmdshell (mssql lets you run a Windows server)
    - Add new user
    - Delete some users etc.

SQL Injection (Prevention)

- You should not trust user input.
  - A simple solution is to escape certain characters (also good for xss)
- Use better solutions
  - ORM (Object Relational Mapper)
  - Prepared / Parameterized SQL

Parameterized SQL

- Values should be sent separately. In the previous example *$username* and *$password* was in the SQL command.

```
sql = "insert into users(name, email) VALUES (?,?)"
cur.execute(sql, [$username, $password])
```

CORS: Cross-Origin Resource Sharing

- Normally, a javascript script (due to **same-origin policy**) can only interact with resources from the same origin. This is good for security.
- However, sometimes you may want to interact with resources from other origins (e.g. *Web fonts*)
- Controlled by *access-control-allow-origin*

- *Detailed information can be found here!*

References

- https://cs155.github.io/Spring2021/lectures/09-web-attacks.pdf
- https://cs155.github.io/Spring2021/lectures/08-web.pdf