</talentlabs>

Express Lecture 12

Security

</talentlabs>

Agenda

- SQL Injection
- XSS (Cross-site-scripting)
- Man-in-the-middle attack and HTTPS
- Don't put sensitive information in the URL
- helmet.js

SQL Injection

</talentlabs>

SQL Injection

What is SQL injection?

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is **the placement of malicious code in SQL statements, via web page input.**

SQL Injection

Example 1:
Request to: http://localhost:3000/products/1 or id is not null;

Route code:

```
`select * from product where id = ` + requestedId
```

Result SQL:

```
select * from product where id = 1 or id is not null;
```

False or True -> True

Originally, our API is for user to get only 1 product, but now it is hacked to return all products!!

SQL Injection

Example 1:

Request to: http://localhost:3000/products/1; drop table product;

Route code:

```
`select * from product where id = ` + requestedId
```

Result SQL:

```
select * from product where id = 1; drop table product;
```

It drops your entire database.

SQL Injection is illegal, don't do it to other people websites.

Historical SQL injection stories
https://en.wikipedia.org/wiki/SQL_injection#Examples

- In August 2014, Milwaukee-based computer security company Hold Security **disclosed that it uncovered a theft of confidential information from nearly 420,000 websites through SQL injections.[65] The New York Times** confirmed this finding by hiring a security expert to check the claim.[66]
- In early 2021, **70 gigabytes of data was exfiltrated from the far-right website Gab through a SQL injection attack**. The vulnerability was introduced into the Gab codebase by Fosco Marotto, Gab's CTO.[69] A second attack against Gab was launched the next week using OAuth2 tokens stolen during the first attack.[70]

Demo Video: https://www.youtube.com/watch?v=wcaiKgQU6VE
(0:23~2:10)

Avoid SQL Injection

With Parameter Binding (?), the injected SQL code is escaped. Therefore

1. Always use **Parameter Binding when handling user input** in a SQL statement.
2. **Never use string concatenation** to handle **user inputs**.

```
.raw(`select * from manufacturer where id = ?`, [req.params["id"]])
```

Database will help you to join the statement, don’t do it yourself!

XSS (Cross-site-scripting)

XSS

2 steps:

1. Hacker **write JavaScript into your database**.
2. Your website **display those JavaScript code directly** without protection.

Create a Product:

```
{
	name: “<script>alert(“You are hacked.”)</script>”
}
```

XSS is illegal, don’t do it...

How The Self-Retweeting Tweet Worked: Cross-Site Scripting (XSS) and Twitter
https://www.youtube.com/watch?v=zv0kZKC6GAM

Avoid XSS

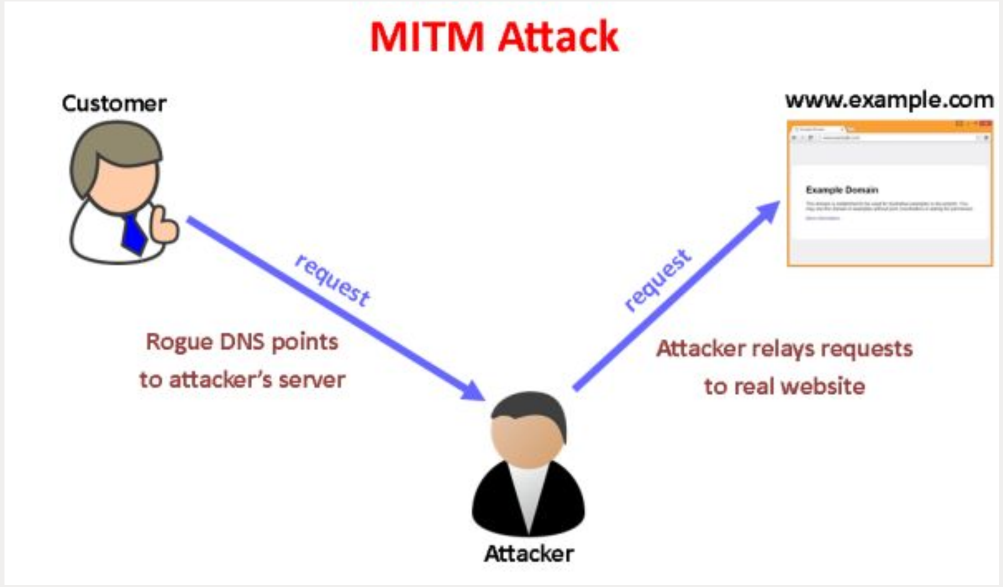
Frontend:
Most modern frontend frameworks will handle this for you. For example, react will not render some dangerous HTML tags directly, like the <script> tag.

Backend:
We can apply XSS filters to remove dangerous content before it got into our database. Like this one
https://www.npmjs.com/package/xss-filters

Man-in-the-middle-attack and HTTPS

</talentlabs>

Who is in the middle?

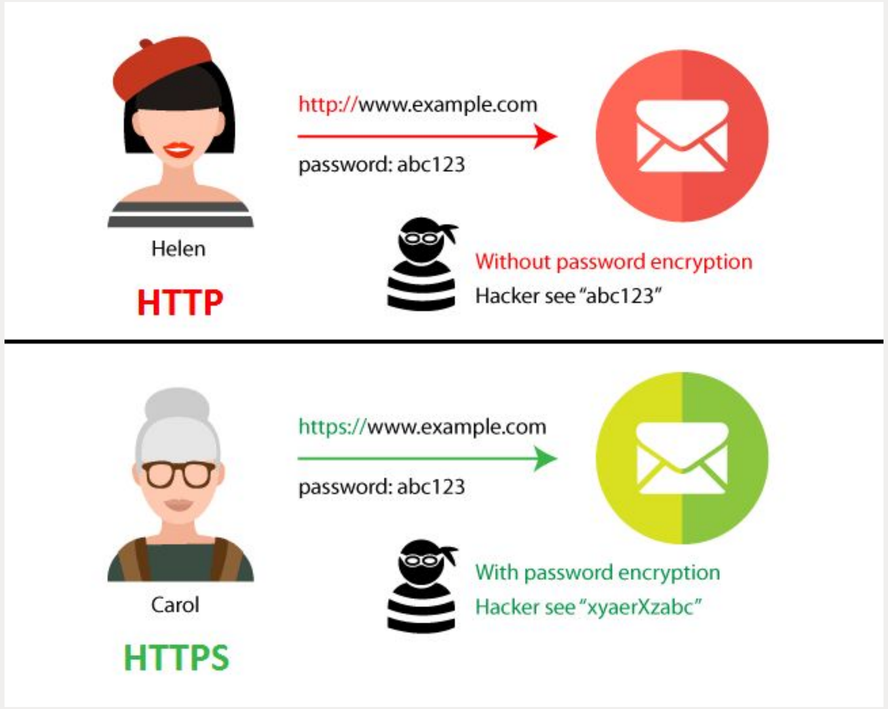

MITM Attack
Customer
www.example.com
Example Domain
request
request
Rogue DNS points to attacker's server
Attacker relays requests to real website
Attacker

- Internet provider.
- Router might have a back door.
- Wifi might be hacked or using a “free” wifi :)
- “free” VPN

Remember, HTTP message is text. If someone is in the middle, they can read your payload.

HTTPS

It protects against man-in-the-middle attacks, because it provides bidirectional **encryption** of communications.



Is HTTPS completely safe?

No, it depends on how the encryption is done. Malwares can “hack” your encryption.

Users had expressed concerns about scans of SSL-encrypted web traffic by **Superfish Visual Search software pre-installed on Lenovo machines** since at least early December 2014.[citation needed] This became a major public issue, however, only in February 2015. The installation included a universal self-signed certificate authority; the certificate authority allows a man-in-the-middle attack to introduce ads even on encrypted pages. **The certificate authority had the same private key across laptops**; this allows third-party eavesdroppers to intercept or **modify HTTPS secure communications without triggering browser warnings** by either extracting the private key or using a self-signed certificate.

Demo:
https://youtu.be/-rSqbgI7oZM?t=665
(11:00~14:00)

Ensure HTTPs

On the backend server side, most of the cloud providers like Google Cloud or Amazon Web Service Cloud support HTTPs servers. When you deploy your application, make sure you enabled the HTTPs option.

On the client side, there is not much we can do… installed malwares are hard to solve.

Don’t put sensitive information in the URL

Browser history

As a backend developer if you allow sensitive information as the URL parameter.

https://example.com/?id_card_number=123456

Hacker can easily get people ID card number by looking at the browser history.

helmet.js

Avoid standard attacks

https://github.com/helmetjs/helmet

It is easy to use and it protect your app from many different standard attacks.

```
const express = require("express");
const helmet = require("helmet");

const app = express();

app.use(helmet());
```