Secure Software Design

Andey Robins

Spring 23 - Week 13

Web Security

Outline

- Web basics
- Web security
- Web vulnerabilities
- DevOps

Prepare yourself for acronym hell.

Web Basics

- HTML/CSS/JS/JSON
- Client/server model
- Cookies, Certs, and Authorities
- DOM
- Frameworks
- HTTP/REST/RPC

Foundational Technologies

- HTML
- CSS
- Javascript (and TypeScript)
- JSON (or XML and others)

HTML

Hyper Text Markup Language: an encoding schema for expressing the content on a web page as hyper text.

```
<html>
  <body>
    <p>I must not fear, fear is the mind killer...</p>
  </body>
</html>
```

CSS

Cascading Style Sheets: a means to express the styling of individual objects or classes of objects in the DOM.

```
app {
  background-color: #000;
  color: #fff;
}

h1 {
    font-size: 2em;
    font-family: 'courier';
}
```

JavaScript

Javascript: is a lightweight, interpreted, JIT compiled, functional programming language. It is used on the server side of 98% of websites today.

```
let fooElement = document.getElementById('foo');

fooElement.textContent = 'New text content';
```

Aside: The History of JS

Back in the Netscape days of the internet, Brendan Eich was brought in to embed Scheme into the next Netscape browser. Instead, they believed it was the correct move to develop a new language just for the web. Brendan then took 10 days, and the early versions of JS are the result.

TypeScript

TypeScript: A superset of JS which adds type-checking, interfaces, and classes to vanilla JS. Developed and released by Microsoft and Anders Hejlsberg, it's used very often in frontend JS applications making use of libraries.

```
let fooElement: Element = document.getElementById('foo');

let fooElement.textContent: string = 'New text content';
```

JSON

Javscript Object Notation: A common means of representing and exchanging information across the internet. It fits very nicely into javascript due to adopting the JS style of modeling data.

```
{
    name: "Andey Robins",
    age: 22,
    job: {
        title: "Graduate Research Assistant",
    }
}
```

Client Server Models

The **Client/Server Model** of design designates two locations for logic and a many-to-one* relationship between them. Computation can be performed at either point in the system, but a strict trust boundary exists between the two.

Static Pages

The simplest setup is a static site (i.e. every user receives the same content). Here the server is said to “serve” the content to the client. A complete HTML document (or whatever) is handed off through HTTP.

Dynamic Pages

Dynamic pages are resources that look different for each visitor. There are two main approaches to delivering dynamic content, placing the burden of rendering on either the client or server.

CSR

"Client-side rendering(CSR) involves rendering pages directly in the browser using JavaScript. All logic, data retrieval, templating, and routing are handled on the client, not on the server." - Scythe Studio

CSR Page

```
<html>
  <head>
    <title> CSR App </title>
  </head>

  <body>
    <div id="app"> </div>
    <script src="../src/index.js"> </script>
  </body>
</html>
```

SSR

"Server-side rendering (SSR) is an application's ability to convert HTML files on the server into a fully rendered HTML page for the client. The web browser submits a request for information from the server, which instantly responds by sending a fully rendered page to the client. Whenever the client navigates to a different page on the website, the server will do the work once more." - Scythe Studio

SSR Page

```
<!DOCTYPE html>
<html>
 <head>
   <title> Web Page Rendered on Server Side </title>
 </head>
<body>
  <h1> This is a Heading   </h1>
  <div>
    <p> This is a form </p>
      <form>
        <label for="fname">First name:</label><br>
        <input type="text" id="fname" name="fname"><br>
        <label for="lname">Last name:</label><br>
        <input type="text" id="lname" name="lname">
      </form>
  <div>
</body>
</html>
```

CSR vs SSR

Server Load -> **CSR**

SEO -> **SSR**

Interactivity -> **CSR**

Intial Load Time -> **SSR**

SPA/SaaS Style App -> **CSR**

Content Heavy -> **SSR**

CSR and SSR Security

- Who is making the requests?
- Where is the data located?
- What is an expected UX?

The DOM

The **Document Object Model (DOM)** is *the* datastructure for the web. It represents a page as a collection of objects, which can be created, modified, or deleted programatically by the documents which make up the page and the JS runtime.

Top Frameworks

- Next.js/Sveltekit
- Nuxt.js/Svelte
- Express
- React/Angular/Vue

Batteries Included

- These have everything you need in one system
- Often opinionated
- Minimal transferability

Ex: Next.js, Sveltekit, Gatsby, etc.

Figure 1: Nuxt, Next, SK, Marko, Qwik, Gatsby, VuePress, Astro, Elder

Batteries Not Included

- Just components
- Do one thing and do it well
- Can often be embedded in other applications

Ex: React, Svelte, Vue, Angular, etc.

Aside: Runtimes

As for backend runtimes, there are several in the JS world right now. They are ranked in order of importance/prevalence.

1. Node
2. Deno
3. Bun

Communication Protocols

These are the potential ways to send and request information over the web.

HTTP

- Basic
- Verb driven
- Infinitely configurable
- No model baked in

REST

CRUD

- **C**reate, **R**ead, **U**pdate, **D**elete
- Maps to HTTP Verbs (POST, GET, PUT, DELETE)
- Good for data-first applications
- Built on top of HTTP

RPC

Web Security

Security Objects

While we'll talk more about security in the web, it's important to provide some discussion surrounding the common, security primitives that we use in the web.

- Cookies
- Certificates
- CAs

Cookies

Certificates

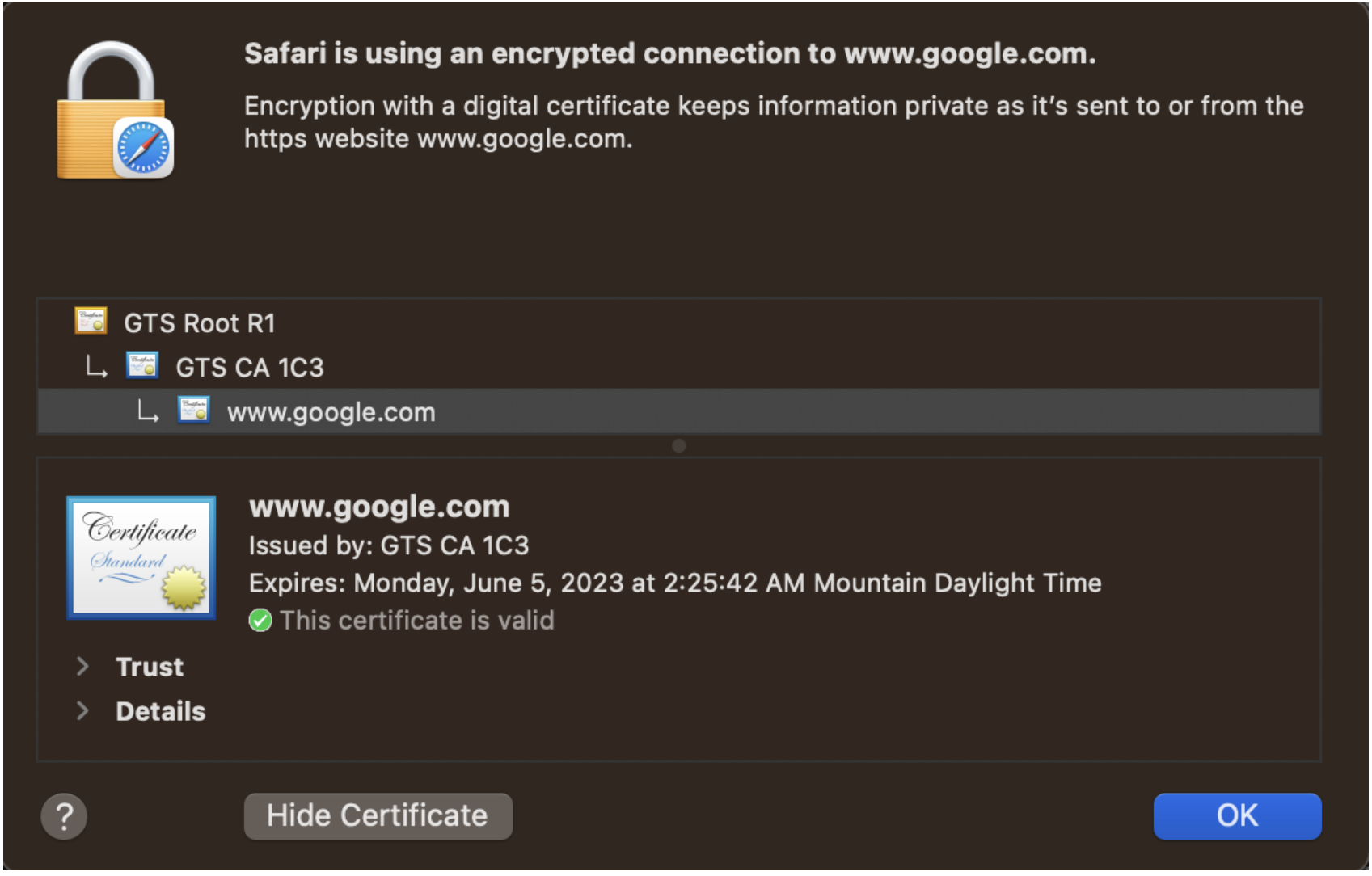

Figure 2: Example Certificates for Google Images

Certificate Authorities

HTTP vs HTTPS

CORS

Cookies and Subdomains

Web Vulnerabilities

XSS

XSRF

Common Mitigation

OWASP Top 10

1. Broken Access Control
2. Crypto Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side-Request Forgery

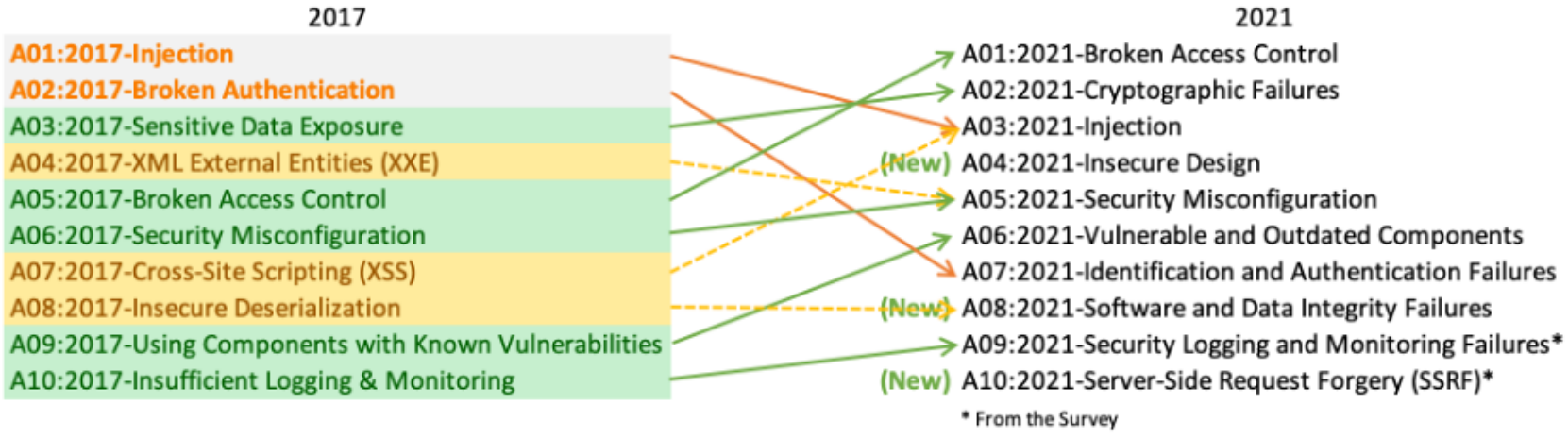

Figure 3: The mapping of 2017 OWASP to 2021 OWASP

DevOps and the Web

Questions

Next Time

- Security Testing
- Project Time!!