Authentication, Cookies, Sessions, & JWT

CS 390 – Web Application Development

J. Setpal

October 18, 2023

Outline

1. Why it's Worth Your Time
2. Authentication
3. Cookies, Sessions, JWT
4. ETC

Outline

WIWYT – Authentication, Cryptography

- Enable user specific access and experiences.

WIWYT – Authentication, Cryptography

- Enable user specific access and experiences.
- Build secure, breach-resilient systems.

WIWYT – Authentication, Cryptography

- Enable user specific access and experiences.
- Build secure, breach-resilient systems.
- Store information securely.

WIWYT – Cookies, Sessions, JWT

- Enable state-persistent client-server communication.

WIWYT – Cookies, Sessions, JWT

- Enable state-persistent client-server communication.
- Prevent unnecessary / reptitive user-inputs.

Outline

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

The need for authentication boils down to a lack of trust

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

The need for authentication boils down to a lack of trust (eg. checking your PUID during an exam).

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

The need for authentication boils down to a lack of trust (eg. checking your PUID during an exam).

Q: How do we establish this verification routine?

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

The need for authentication boils down to a lack of trust (eg. checking your PUID during an exam).

Q: How do we establish this verification routine?
A: **Shared Knowledge**.

What is Authentication?

Authentication is a **verification routine** used to ensure that a user is who they say there are.

The need for authentication boils down to a lack of trust (eg. checking your PUID during an exam).

Q: How do we establish this verification routine?
A: **Shared Knowledge**. "If you know/have/are `xyz`, and I know that only you know/have/are `xyz`, you're you!"

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching

b. **Has:** Security Key

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

So, how do we go about:

a. Build Authentication Policy?

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

So, how do we go about:

a. Build Authentication Policy?
b. Securely Storing Data?

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

So, how do we go about:

a. Build Authentication Policy?
b. Securely Storing Data?
c. Implement Authentication?

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

So, how do we go about:

a. Build Authentication Policy?
b. Securely Storing Data?
c. Implement Authentication?

Breadth-First Answer: Use popular pre-existing frameworks.

Authentication in the Web

What are examples of shared knowledge? Information the user:

a. **Knows:** Passwords, Tokens, Pattern-Matching
b. **Has:** Security Key
c. **Is:** Biometrics

So, how do we go about:

a. Build Authentication Policy?
b. Securely Storing Data?
c. Implement Authentication?

Breadth-First Answer: Use popular pre-existing frameworks.
Depth-First Answer: **Let's talk about it**.

Setting Password Policy

Q: What's the better password?

a. `a*s!c,s,[T wd(*UE#)dw$I!wl;kmw` (30 characters)

b. `Don't or fox find pinch swarm!` (30 characters)

Setting Password Policy

Q: What's the better password?

a. `a*s!c,s,[T wd(*UE#)dw$I!wl;kmw` (30 characters)

b. `Don't or fox find pinch swarm!` (30 characters)

A: Trick Question! They're equally complex.
Search Space: $7.72 \cdot 10^{57}$ possible combinations.

Setting Password Policy

Q: What's the better password?

a. `a*s!c,s,[T wd(*UE#)dw$I!wl;kmw` (30 characters)

b. `Don't or fox find pinch swarm!` (30 characters)

A: Trick Question! They're equally complex.
Search Space: $7.72 \cdot 10^{57}$ possible combinations.

Idea: Humans interpret information differently from computers.

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.
d. CAPTCHAs.

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.
d. CAPTCHAs.
e. **2 Factor Authentication**.

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.
d. CAPTCHAs.
e. **2 Factor Authentication**.

Q: Should we force regular password changes?

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.
d. CAPTCHAs.
e. **2 Factor Authentication**.

Q: Should we force regular password changes?
A: No. Why not?

Setting Password Policy

How can we capitalize on this?

a. Set minimum password lengths
b. Check against known breaches. Ex: `freerainbowtables.com`, `haveibeenpwned.com`
c. Rate-limited Requests.
d. CAPTCHAs.
e. **2 Factor Authentication**.

Q: Should we force regular password changes?
A: No. Why not?

Forcing a change incentivizes building passwords using a pattern, or remembering them insecurely.

Saving Data Securely

We know how to work setup our data, but what about saving it?

Saving Data Securely

We know how to work setup our data, but what about saving it?

We need to make it such that even *when* there is a breach,
end users are minimally impacted.

Saving Data Securely

We know how to work setup our data, but what about saving it?

We need to make it such that even *when* there is a breach,
end users are minimally impacted.

Enter **Encryption**. It's the process of encoding information such that an unauthorized individual is unable to access a given set of information.

Encoding vs Encrypting

An important concept is the difference between encoding and encrypting.

Encoding vs Encrypting

An important concept is the difference between encoding and encrypting.

While encryption involves encoding data, the two are **not interchangeable** terms.

Encoding vs Encrypting

An important concept is the difference between encoding and encrypting.

While encryption involves encoding data, the two are **not interchangeable** terms.

Encoding data is used only when talking about data that is not securely encoded. Base64 is an encoding, SHA-256 is encryption.

We'll discuss encryption in further detail during the web security modules of this course.

Outline

Why do we need Cookies?

HTTP is a **stateless** protocol.

Why do we need Cookies?

HTTP is a **stateless** protocol. As a consequence, session information does not propagate between requests.

Why do we need Cookies?

HTTP is a **stateless** protocol. As a consequence, session information does not propagate between requests.

Having a consistent user-experience requires information persistence.

Why do we need Cookies?

HTTP is a **stateless** protocol. As a consequence, session information does not propagate between requests.

Having a consistent user-experience requires information persistence.

Cookies (also Sessions & JWT) are ways of extending the HTTP protocol to work with state-specific information.

Why do we need Cookies?

HTTP is a **stateless** protocol. As a consequence, session information does not propagate between requests.

Having a consistent user-experience requires information persistence.

Cookies (also Sessions & JWT) are ways of extending the HTTP protocol to work with state-specific information.

All three inject-state specific information through response headers, saving data temporarily using different mechanisms.

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function.

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function. Additionally, we can supply the arguments:

Attribute	**Data Type**	**Use-case**
`maxAge`	Integer	Time to expiration

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function. Additionally, we can supply the arguments:

Attribute	**Data Type**	**Use-case**
`maxAge`	Integer	Time to expiration
`Expires`	Date	Date of expiration

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function. Additionally, we can supply the arguments:

Attribute	**Data Type**	**Use-case**
`maxAge`	Integer	Time to expiration
`Expires`	Date	Date of expiration
`httpOnly`	boolean	Visibility to client-side JS

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function. Additionally, we can supply the arguments:

Attribute	**Data Type**	**Use-case**
`maxAge`	Integer	Time to expiration
`Expires`	Date	Date of expiration
`httpOnly`	boolean	Visibility to client-side JS
`secure`	boolean	Present over HTTPS-only

Cookies

Stores information **client-side**. Added by including `set-cookie` as part of the response header.

In Express, you can pass key value pairs to the `res.cookie` function. Additionally, we can supply the arguments:

Attribute	Data Type	Use-case
`maxAge`	Integer	Time to expiration
`Expires`	Date	Date of expiration
`httpOnly`	boolean	Visibility to client-side JS
`secure`	boolean	Present over HTTPS-only

To read cookies, we define `cookie-parser` middleware at the start of each route, and then reference `res.cookies.<var>`.

Sessions

Cookies have a significant vulnerability: they are visible to the client.

Sessions

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

Sessions

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

This exposes sensitive information; both to the client and malicious actors.

Sessions

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

This exposes sensitive information; both to the client and malicious actors.

Sessions solve this! The relevant information is stored **server-side**, with an identifier cookie that enables the client-server association.

Sessions

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

This exposes sensitive information; both to the client and malicious actors.

Sessions solve this! The relevant information is stored **server-side**, with an identifier cookie that enables the client-server association.

Important: Session persistence is limited to the runtime by default; we need to use a DB (ex. MySQL, NoSQL, MongoDB) to enable persistence.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

This token is broken into three components:

1. The **header**, containing the encryption algorithm and type.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

This token is broken into three components:

1. The **header**, containing the encryption algorithm and type.
2. The **payload**, containing the actual data.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

This token is broken into three components:

1. The **header**, containing the encryption algorithm and type.
2. The **payload**, containing the actual data.
3. The **signature**, that verifies the authenticity of the data.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

This token is broken into three components:

1. The **header**, containing the encryption algorithm and type.
2. The **payload**, containing the actual data.
3. The **signature**, that verifies the authenticity of the data.

Combined, these enable storing information securely in the client-side.

JSON Web Tokens

JSON Web Tokens are a way to store encrypted information within client-side browser storage.

This token is broken into three components:

1. The **header**, containing the encryption algorithm and type.
2. The **payload**, containing the actual data.
3. The **signature**, that verifies the authenticity of the data.

Combined, these enable storing information securely in the client-side.

However: These are vulnerable to CSRF attacks (discuss during security modules), that allow malicious actor to steal identities using JWT.

Let's Integrate Authentication!

If you can view this screen, I am making a mistake.

Outline

Thank you!

Have an awesome rest of your day!

Slides: `https://www.cs.purdue.edu/homes/jsetpal/slides/auth,cookies,sessions.pdf`

If anything's incorrect or unclear, please ping: `jsetpal@purdue.edu`
I'll patch it ASAP.