Seneca
SCHOOL OF INFORMATION AND
COMMUNICATIONS TECHNOLOGY

BTI425

Web Programming for Apps and Services

Security topics introduction

For our purposes in this course, the *security topics* will introduce you to the security-related goals, terminology, and techniques for a web app.

> *In a separate document, we will introduce infrastructure-based security topics, such as public key infrastructure and transport layer security.*
>
> *Those infrastructure-based topics are outside the scope of this document.*

The overall goal is to protect a web app's usage and its data, making the app available for only its intended users.

> *A "web app" is an app that is created with web technologies.*
>
> *It is assumed that this kind of app relies on HTTP, and is typically written in JavaScript (supported and complemented by related technologies, including HTML, CSS, JSON, etc.).*
>
> *And, it's assumed that an app can be designed to be deployed to one or more device platforms (including a browser, a native-code platform, a server, etc.).*
>
> *It is also assumed that a web app is a distributed app, in that it could be composed of separate parts that run on two or more devices and communicate over a network.*

The security topics coverage in this course will focus on the parts that we implement in software ourselves. We will have little or no treatment of other security topics, including physical and network security. (It is assumed that we will deploy our app on a host that supports transport layer security, aka “https”.)

We will start by defining and studying a *security system*.

What is a *security system*?

Any system is an interdependent group of items that work together to implement a goal or purpose.

As a result, you can think of a *security system* as programmed items that work together to protect an app’s usage and its data.

A typical security system does the following:

1. Identity managment
2. Authentication
3. Authorization

Each task will be explained below. Each can be embedded in an existing app, or used as a distinct service. It is also possible (and typical) to bundle two tasks together.

1. Identity management

Identity management is the process of defining, storing, and managing user accounts for an app.

Each user of an app must be uniquely identified. A “user account” implements this requirement.

The user account is an object (which consists mostly of data, with little or no behaviour) that represents a human *user* of an app. A user account includes identification and description properties, including user name, a shared secret (i.e. a password), email address, etc. It usually also includes some other properties (e.g. family name, given name, etc.), as well as metadata (e.g. active/locked status, user account creation date, etc.).

All user accounts are saved in a persistent store (i.e. a database). One or more user accounts are typically designated as “user account managers”, which can manage *all* user accounts (e.g. active a user account, delete a user account, etc.)

Typical *identity management* tasks include the ability to register for a new user account, and enable a user to edit some properties of their user account (e.g. name, email address, etc.). For user account managers, typical tasks also include the ability to view/inspect user accounts, search for a user account, and so on.

Sometimes, identity management is a part of an app (embedded, in other words). Alternatively, it is often done as a separate app or service. In that situation, you can configure your app to "trust" the external identity management app for this aspect of a security system.

> *This is how "Sign in with your Microsoft account" works.*

As a second-year student, should you write your own identity management app? NO. You must use an existing, widely-used, feature-laden, and robust app, which is created and maintained by people who have security system expertise. (However, as an academic exercise, it is often useful to create a working prototype or minimal app. We may do part of that task in this course.)

> *Another term for the identity management app is "identity authority".*
>
> *In other words, the app is designated and recognized as the authority for the task of defining, storing, and managing identity (user account) information.*

2. Authentication

Authentication is the process of presenting and validating credentials.

There are two process workflows. One is followed when a user has not yet authenticated, and the other is followed when a user has been recently authenticated, and requests another resource.

First process workflow: "Not yet authenticated"

If a user has not been authenticated by the security system, then the user must present their credentials. For our purposes, this is typically done in two different ways:

1. User interface (i.e. a login page in an app)

2. Programmatic interface (i.e. a resource/endpoint in an app)

For both ways, the user typically presents a *username* and a *password* as their credentials.

After the security system validates the credentials, the security system issues (delivers, returns) a package of data to the user:

1. If the user authenticated through a browser (with a "login" user interface), then the package is a cookie (or more specifically, an *authentication cookie*)

2. If the user authenticated through an HTTP client/requestor (programmatic interface), then the package is an access token (often just referred to as a "token")

The cookie or token includes information about the issuer, descriptive information about the user, and information about the cookie or token lifetime. It does *not* include secret information (such as a password).

> *"Descriptive information about the user" is known as claims.*
> *More about claims will be covered soon.*

> *What is the format or content of the data in the cookie or token?*
> *For a cookie, there is a defined standard.*
> *For a token, there is no single standard. However, there are a few widely-used approaches, which we will cover later.*

A browser saves or stores the cookie in a secure manner. This is done automatically as a browser feature, and the user or programmer does not need to do any extra work to save the cookie, and then use the cookie again in the future.

In contrast, when using an HTTP client/requestor (e.g. an Angular app, or a React app), the programmer *must* do extra work to save or store the token in the app (in memory and/or persisted) in a secured manner.

The other process workflow: "Has been recently authenticated"

If a user was recently authenticated, and has a cookie or token that has not yet expired, then the user can present the cookie or token as their credentials.

When using a browser, assume the recently-autheticated user requests a different resource in the same app. The browser automatically fetches the cookie from its storage area, and includes it in the header of the request.

When using an HTTP client/requestor, *the programmer* must fetch the token from the app's storage area, and include it in the header of the request.

> *Angular apps can work a bit differently, and help enable this for the entire app.*
> *You'll learn how do do this soon.*

In both situations, the listening app will notice the cookie or token in the request headers. It will then inspect and validate its contents.

- If valid, the app will allow the request to continue. Typically, the app will create a "security principal" that represents the user for the lifetime of the request, and attach it to the request object (context).

- If not valid, the request cannot continue. A browser user will be sent an HTTP 401 response, and likely redirected to a login page. An HTTP client/requestor will be sent an HTTP 401 response, and likely a message inside an object.

The programmed code that implements authentication is modular:

- The “not yet authenticated” code is always a part of the identity management app.

- The “has been recently authenticated” code is also always part of the identity management app. *In addition*, it can be part of *your* app. (We add code that enables your app to “trust” the identity management app, and to decode an access token.)

As a second-year student, should you write your own authentication app? NO. As above, you must use an existing, widely-used, feature-laden, and robust app, which is created and maintained by people who have security system expertise.

3. Authorization

As you have learned, after a successful authentication, the user will present a cookie or token as credentials with every request for a resource.

However, is the user *allowed* to successfully use that resource? Well, that depends.

Authorization is the process to determine whether a request for a resource can be successfully completed.

How does that process work? What and where is the programmed code? The answer is that the code is (for our purposes) *always part of the framework* (or library or whatever word that we use to describe the add-on JavaScript-based components).

> *In an Angular app, the CanActivate route guard implements the base functionality. We also typically write our own custom “guards”.*

What is used to determine whether a request can succeed? The answer is *claims*. In other words, we use the *descriptive information* that is encoded into an access token.

To begin your understanding of how this works, consider the simplest scenario first: Assume that an access token was inspected (i.e. decoded), and validated. For our purposes *validated* means that 1) the token was issued by the trusted identity authority, and 2) the token has not expired (i.e. its lifetime is still valid).

At the end of this inspect-and-validate scenario, assume that the answer is “yes”. We can therefore consider that the request “is authenticated”. (FYI, this is often coded as a boolean *isAuthenticated* in the framework’s authorization code/component.)

So, to recap, the simplest scenario inspects the “lifetime” claim (often coded as an expiration date-and-time), and returns a yes/no answer.

Claims, more information…

There is a somewhat-official and fancy definition for claim, which is:

> *“a statement that one subject makes about itself or another subject”.*

A “statement” is descriptive information. A “subject” is a participant in the lifetime of an app, and it could be a human, a corporate body (i.e. organization), or a programmable object (e.g. a security system).

We like a much better definition for our purposes, which is:

> *“descriptive information about the user”.*

What kind of information?

- Purely *descriptive* information, such as family name, birth date, or hair colour
- Information that describes the user in the context of the security system (sometimes known as security system metadata), such as the token issuer’s identity, and the expiry date-and-time
- Information that describes the user in the context of the app, such as organizational unit(s), and job/position title
- Information used for access control (sometimes known as roles and permissions), such as broad-based role(s), fine-grained permission(s), and task or activity indicators

How can we use claims? Now that you have an idea about the information that can be conveyed, we can have an almost-limitless way of thinking about security in our app. For example, we can use claims for:

- user-customized content or app usage flow
- high-level role or identity grouping/membership
- app-specific identity grouping or membership
- access control (to data and/or behaviour)

Where are claims defined and stored?

For our purposes, the identity authority is responsible for this task. Each user account will have a *claims collection*.

The decision about what claims to define and use is completely determined by the programmer and sponsor of the app. Obviously, there must be a user interface for managing claims; this topic is beyond the scope of this course.

> *It is also possible and common to distribute this task among the identity authority and your app.*
> *We see this in apps that allow/enable login via Google or Facebook; identity management and "login" are handled by an external provider, and your app attaches locally-defined claims to the cookie or token.*

Claims issuance

After a successful authentication, the identity authority creates a cookie or token. That package of data includes the claims. As a result, any of the claims can be used in any way by any code on the execution path.

Further study

The now-cancelled Assignment 3 was to add a simplified security system to the work done for Assignment 2.

In the future, for your own learning only (and not for marks!), you can attempt to use the guidance and techniques in the how-to documents do do that.