Authentication and Authorization

As mentioned previously, computer programs are used to work with data, regardless of its form. If the data is valuable, then improper handling of it or unauthorized access to it can significantly reduce or destroy its value. Therefore, data must be protected. There is a separate section in this book about protecting the system as a whole. And in this section, we will focus on determining who can do what with the data.

It is also necessary to mention rare cases when a certain program works with public data according to its own special algorithms. Here, it is the program (or rather the algorithms) that is the know-how that must be protected. These situations are beyond the scope of this book.

Almost all industrial systems have a concept of distribution of roles and powers.

RBAC and ABAC

Let's talk about two approaches to managing user powers. Often, these two approaches are mixed and complement each other.

RBAC

RBAC stands for "Role-based access control". This means that all users are divided into functional roles, like those played in the theater. Imagine that the actor playing Hamlet falls ill, and another one comes on to replace him. This actor will play the same play, read the exact text, and perform the same actions on stage. That is, two people have the same functional role as Hamlet.

There can be an unlimited number of roles in the system, but in practice, there are no more than a dozen. Each role is assigned to certain permissions. For example, you can read specific data and rewrite it; you can read other data, but you cannot delete it. Only specific roles open parts of the visual interface of the system. For example, the administration panel.

One user can combine several roles. Imagine a sales department. The manager works with clients and registers new deals, that is, adds records to the database. He can modify existing data, such as the status of a deal or make a payment. However, he cannot delete deals. The senior manager, who oversees this employee, holds the same role as his subordinate. In addition, he has an additional role with broader powers. For example, transfer a deal to the archive, or change bank details. And only the administrator can physically delete records - a senior role with great responsibility.

ABAC

ABAC stands for "attribute-based access control". This approach is characterized by assigning powers to a user based on the attributes of this user. For example, a person in the position of "senior manager" can add, edit, and archive data on the deal. Unlike RBAC described above, powers are not assigned to a specific person but are issued to all employees in the position of "senior manager". Any employee transferred to this position automatically acquires new powers and loses the old ones for their previous position in the company.

Attributes can be understood as a wide range of characteristics necessary for the company's business processes. For example, the manager from the example above can enter data into the system only when he is on shift. That is, the local date and time are considered. Or the powers to enter aggregate data are granted to him after the twentieth of each month and are withdrawn on the first of the following month. In other words, he can only create and edit a report at the end of the month, and he won't be able to correct anything retroactively.

General recommendations on permission

Please remember that in modern systems with API, users are not necessarily people. Your system may be accessed by an NPA – non-personal account from another system. This account must also have permissions (roles or attributes).

There is a general principle of granting a minimum number of permissions in the system. Under no circumstances should ordinary users have the same rights as the administration.

Do not forget to separate accounts used for system development and maintenance from user accounts. The system administrator, i.e., business data, rarely changes the database structure and seldom executes direct SQL queries. The encapsulation principle applies here too: system administrators should not have access to its internals.

Regularly review the list of users and assigned permissions. It happens that an employee has moved to another position or left the company, but they still have active permissions in information systems. This can lead to undesirable consequences. The same applies to NPA: the permissions of external systems should be checked carefully.

In practice, RBAC systems are easier to maintain than ABAC. Especially if you use user grouping. Adding a new user to the Administrators group automatically assigns them new roles.

Now we need to talk about the practical implementation of the above. Firstly, we need to determine which user is accessing the system and then decide what permissions they have.

Authentication

"Authentication" is responsible for establishing the user's identity. We emphasize that here we are talking not only about human users, but also about NPA that came from other systems. Often, the mechanisms for determining the authenticity of their identity are similar or identical.

Identification is a task that humanity has been solving for millennia. And here the analogy with the evolution of a sword and shield is quite appropriate. For each new defense, after some time, a means is found to help overcome this defense.

The organization decides what security measures it considers sufficient. In some industries, such as banking and medicine, it is necessary to consider the requirements of the law. One way or another, protection comes down to the following principles, which can be implemented separately or mixed: Something you Know, Something you Have, Something you Are. Chronologically, these principles were introduced in this order.

* Something you Know. The user's identity is established based on the information they know. Historically, this is a login and a password. The login identifies the user, and the password confirms that this is who he claims to be. The PIN code also belongs to this type. The disadvantages of this type include the user forgetting the details and the risk of them being lured out or stolen by third parties, especially if the user writes them down somewhere. The combination of login and password can be too simple, making it vulnerable to analysis and hacking.
* Something you Have. We are referring to a specific physical device that verifies the owner's identity. In the non-computer world, this is a passport or similar document. Nowadays, smartphones are used, where there is a program that generates unique keys. They are synchronized with the response keys in the system of interest. The key changes every half a minute. The user enters the key along with the login and password — a similar principle to a confirmation SMS. The user is required to confirm the operation in the system by entering the code from the SMS. And finally, there are physical devices for the USB port. While the key is inserted into the port, the system is open for work. The disadvantages are obvious: the item can be stolen from the owner.
* Something you Are. We are talking about the user's biometrics. The simplest example is Face ID and similar checks used by modern smartphones. In addition, there are physical keys (from the example above) with a built-in fingerprint scanner. The disadvantages of this method include the ease of overcoming the barrier. Some systems recognize a photograph as the face of a living person. A fingerprint transferred to a suitable surface can sometimes fool the program.

The most secure system that the author of this chapter had to work with looked like this. Upon entering the building, you must pass through a passport check and obtain a temporary pass, which grants access to the elevator on a specific floor. In the room with the attendant, there is a computer terminal connected only to the internal network. Electronic devices are not allowed into the room. The attendant runs a program on his local computer that generates a password and then prints it. The password from the piece of paper must be entered on a secure computer along with the login and work with the system. After half an hour, leave the room for a new password. The duty officer does not enter the room with a secure computer and transmits the password on paper. That is, the duty officer cannot compromise the system, and the password cannot be intercepted electronically.

A combination of methods is often used in industry. Something you Know and Something you Have are the easiest to organize. Google Mail, Git Hub, and AWS Console work on this principle.

Authorization

After the user's identity is established, the system determines the available permissions. The main strategies of RBAC and ABAC were discussed above.

In the past, when desktop applications were in use, it was enough to check the user's identity and permissions once. After checking, the user saw the program menu and did the work. However, even now, it is enough to log in to the operating system on a computer or smartphone and continue working without additional questions. If you leave the computer for a long time, it will be locked. If you turn off the smartphone screen, it will also be locked. And to turn it on, you will need to re-establish your identity.

With web applications, things are not so simple. HTTP is a stateless protocol. It does not save information about previous requests. Therefore, for each request, you need to establish the user's identity. In other words, re-enter the login and password, the key from the physical device, and so on. Imagine how cumbersome this process would be.

There is a token mechanism for this, which we will consider below.

JWT

JWT stands for "JSON Web Token". This mechanism allows you to avoid cumbersome user authentication for each request. It works as follows:

* The user logs in using login and password from the client application (browser, but not only).
* If the user's identity is confirmed, the application creates a new JWT and sends it to the client.
* Each subsequent request from the client to the server is supplemented with this JWT.
* When the JWT expires, the server issues a new one or invites the user to re-authenticate.

How does JWT work? It consists of three parts, each of which is JSON. JWT parts are converted to Base64 String and separated by dots:

* Header. This contains the token type (always "JWT") and encryption type.
* Payload. This part can contain any useful information (except secret, of course). For example, the date and time of expiration of this token, the username, and a list of their roles. Although there are no official restrictions on the length of JWT, it is recommended to keep it small. This is definitely not the place to send a sales list to a client.
* Signature. This section is composed of a header, payload, and a secret. The secret is a cryptographic key stored on the server (or on a third-party service that is a provider of JWT).

Access Token and Refresh Token

We have considered the basic scenario when only an Access Token is used. It is attached to each request to the server. When it expires, you need to authenticate again.

For greater convenience, an advanced scheme with two tokens is used: Access Token and Refresh Token. The difference is that the Access Token has a short lifespan, for example, 10 minutes. But the Refresh Token, on the contrary, lives a long time: a day, a week, or even a month. When the Access Token becomes invalid, you need a Refresh Token to get a new Access Token.

Of course, at some point, the Refresh Token will also expire, and then you will have to authenticate again. However, much less often than when using a single Access Token.

It is easy to imagine that the theft of a Refresh Token poses a much greater threat, because its lifespan is much longer. And it needs to be stored more carefully, for example, in Http-only and Secured cookies. Such cookies cannot be changed on the client using JavaScript and can only be passed inside the HTTPS request. It helps prevent Cross-Site Scripting (XSS) attacks.

OAuth2

If we are talking about a public service, then we must keep in mind that the user most likely will not want to remember many credentials for each visited site. Especially if he follows security requirements and comes up with a new password each time. For these purposes, the authentication process can be outsourced to a third-party service.

There is a wide range of such services on the market. Let's mention the following: Google OAuth, Auth0, Azure AD, and Keycloak. The most popular of them is Google.

Working with a third-party service comes down to the following scheme:

* User Authorization. The client redirects the user to the authorization server to log in and grant permission.
* Authorization Grant. After user approval, the authorization server sends an authorization code back to the client.
* Access Token Request. The client exchanges the authorization code for an access token by authenticating itself with the authorization server.
* Access Resource. The client uses the access token to request resources from the resource server.

SSO

Single Sign-On (SSO) is a convenient mechanism that allows you to authenticate a user once and then transfer the fact of identity to several applications at once. This scheme is convenient to use in corporations. The user logs into the operating system and then gets access to corporate applications. Of course, when it comes to authorization, the powers of this user will be different, depending on the specific system.

Custom Authentication

Sometimes you need to do authentication & authorization yourself. In this case, you need to use reliable (!) libraries for the selected programming language. We will show this using an example for Express.js.

Note the two directories: token-auth-backend and token-auth-frontend. The first of them contains a simple application that simulates the authentication process, and the second one contains a client application with a login and password entry form.

Let's consider the token-auth-backend application in detail. Execution starts in the app.js file. A separate layer, represented by middleware, is responsible for authentication: "login" handler and authentication helper. All incoming requests go through this layer. In real applications, some of the functionality may be open and not require authentication. To simplify things, we only request a login, do not check its presence in the database, and do not register a new user. Let's imagine that these actions have been performed, and we have made sure that such a user exists.

Now we want to issue him a JWT. This happens in authHelper.js, "createToken" function. Note that the work of creating the token is taken care of by the "jsonwebtoken" library. Always check the reputation of the library before using it! Always use the latest version of the library! When the token is created, it is attached to the response and sent back to the client. The client stores the token in the localStorage of its browser.

The next time the client contacts the server and requests the "hello" endpoint, the program will call the authHelper.js function, "checkAuthentication". In this function, we again resort to the help of the "jsonwebtoken" library and thus check the correctness of the passed JWT. If it is confirmed, then we execute the requested function, in this case, "hello".