Security in Backend Systems

In the design of modern computer systems, considerable attention is paid to security issues. This is because the number of threats has increased significantly over the past two decades.

At the beginning of the century, desktop applications were in use. Data was usually stored on the user's computer and transferred via external media. As a rule, valuable data can be stolen by one of the company's staff members. To do this, an employee needed to copy the data to their media and remove it from the building. Accordingly, security measures were more likely to be based on preventing people from accessing the room where the computer was located, as well as the basic login credentials in the operating system and the computer program.

The matter became more complicated when local networks, and especially the Internet, emerged. The intruder did not need to penetrate the protected area or steal an employee's laptop. Although, in fairness, even today, there are critical systems that require access to be on the company's territory and using a special computer. For example, computer systems of critical infrastructure, such as those involved in city power supply management.

With the advent of network technologies, it was necessary to protect networks and access points. When mobile devices, including laptops and smartphones, appeared on the scene, it became even more complicated. A small device is more likely to be lost or stolen. In addition, new work scenarios have emerged, where users are outside the office and work with data storage remotely, without needing to come to the office and synchronize.

And finally, cloud technologies have changed the principle of organizing computer programs. They have fragmented into small services, replacing monolithic applications. These parts of the whole are executed on someone's computers that are outside the company's control. Moreover, these computers can be located in another country. This means that legal issues about the transfer of data abroad inevitably arise.

Thanks to artificial intelligence, electronic assistants based on large language models (LLM) are appearing. To incorporate specific business details into the general model, the model is expanded through Retrieval Augmented Generation (RAG). It follows that it is necessary to provide the LLM with the company's data, which may include sensitive information. Information security in this area is still in the process of becoming established.

In this chapter, we will examine the general principles of creating secure programs that can be applied to any application technology.

**Assets and their value**

Most applications work with data in one way or another. In a computer system, the most valuable thing is the data, not the program itself. Of course, there are exceptions if the program utilizes proprietary algorithms, but such programs are relatively few in comparison to their total number.

Loss of data can cause critical damage to the company's reputation. Users will stop trusting this company's software or stop buying its services. The company can go bankrupt. Here are just a couple of examples:

* Equifax was forced to pay $700 million after the theft of users' personal data. The theft was made possible due to flaws in the web application's design.
* Cryptocurrency exchange Mt. Gox lost 850 thousand bitcoins after being hacked. The exchange ceased to exist, resulting in millions of dollars in losses for investors.

It follows that data needs to be protected. What kind of data and how? The decision depends on the nature of the data. Specific categories of data are protected by law. For example, passport data, medical records, and financial information. Some data cannot be placed abroad. There is a procedure for operations that can be performed with personal data, and this procedure is also regulated by law. The laws will be discussed below.

Let's denote valuable data with the word "asset". Each asset has a "value", that is, a measure of its importance. First of all, you need to understand the data and determine which of them are assets and what their value is.

Forming protection costs money. The cost includes the work of specialists, licenses for software products, and infrastructure, among other expenses. The asset value must be commensurate with the budget allocated for its protection. The discrepancy can occur in either direction. For example, weak protection for valuable data. Examples are given above. And vice versa, sophisticated protection for data with the value of a sandwich. Both options are extremes that should be avoided.

Here are characteristics of an asset:

* Holds its value.
* Produces value.
* Provides access to value.

There are three principles of information security:

* Confidentiality. This means that only those with the appropriate authority can access the data.
* Integrity. Ensures that data is accurate, complete, and unaltered except by authorized parties. Only authorized personnel are permitted to modify the data.
* Availability. Ensures that information and resources are accessible to authorized users when needed.

Together, they are abbreviated as CIA.

The goal of a developer and architect is to arrange storage and data access in a way that ensures the above principles are observed.

**Vulnerabilities**

A vulnerability is a weakness in a system that an attacker can exploit to breach CIA and ultimately steal data, disrupt a company's operations, or cause more serious consequences in the event of an attack on critical infrastructure facilities.

Any software has vulnerabilities. Some of them are known and can be fixed with patches. Others remain hidden for a certain period and can be exploited by an attacker who discovers them before others. Such vulnerabilities are called 0-day. This means that the authors of the problematic software had zero days to fix the situation.

Additionally, developers should remember that even if their code is perfect from a security perspective, it relies on other libraries and operating system components, which can contain critical vulnerabilities. It is not uncommon for a publicly used library to have a vulnerability built in. The authors of such a library hope that their product will eventually become a building block of an important system, and they will gain access through it. This is called a backdoor.

**Decrease the attractiveness of assets by design**

At the system design stage, it is worthwhile considering which assets should be stored in the system and why they should be stored there. The general practice is to store as little sensitive data as possible. Then the attractiveness for intruders will be less.

Let's demonstrate using the example of a fictitious application "Dog Walker". This is a web application designed for dog owners seeking a way to stay engaged while walking their dogs. Through this application, the user can find a walking buddy. The program has the following requirements:

* The search is tied to the place of residence of the dog owner. People are most likely to walk near their home or in a nearby park.
* You can filter walking buddy. For example, by age, gender, and favorite topics of conversation.

Of course, there may be many more requirements; however, for the purposes of this demonstration, we will focus on these for now.

A straightforward solution is to force users to enter their full name, date of birth, and home address. This is personal data protected by law. We will have to implement their protection properly, which will increase the costs of our startup. And will users want to share this data? Will this approach scare them away?

However, we can altogether refuse to store personal data by simply reversing the application filters. Why do we need the full name of the dog owner if we can store the dog's nickname and its portrait? This data is not personal, but it gives an idea of who you will be walking with. The date of birth can be replaced with an age range, for example, 18-25, 26-35, etc. Instead of a home address, you can indicate the desired meeting place. They are not tied to a person in any way and, therefore, are also not personal data.

Thus, we completely removed any mention of personal data, reduced the attractiveness of the assets, and, at the same time, preserved the application's functionality.

**STRIDE framework**

There are many approaches to identifying possible ways an attacker can harm an application. Knowing what data is stored, how it is stored, and how the application is structured, you can identify potential attack vectors. The STRIDE framework can help with this. The acronym stands for:

* Spoofing. This refers to the act of masquerading as another user or device to gain unauthorized access to systems, data, or communications. For example, an attacker might use stolen credentials to impersonate a legitimate user.
* Tampering: This involves the unauthorized modification of data or code. Attackers may alter files, databases, or software to gain unauthorized access or disrupt services. For instance, an attacker might change the parameters of a transaction in a database.
* Repudiation: This occurs when a user denies having acted without the ability to prove otherwise. For example, if a user disables logging mechanisms, they can later claim that they did not perform a specific transaction, which can complicate accountability and auditing.
* Information Disclosure: This refers to the exposure of sensitive information to unauthorized parties. This can occur through various means, including inadequate access controls, software vulnerabilities, or social engineering attacks.
* Denial of Service (DoS): This threat involves making a system or resource unavailable to its intended users. This can be achieved through overwhelming a server with requests or exploiting vulnerabilities to crash a system.
* Elevation of Privilege: This occurs when an attacker gains higher access rights than intended. For example, a user with limited permissions might exploit a vulnerability to gain administrative access to a system.

These provisions can be divided into the following groups:

* The attacker pretends to be someone he is not (Spoofing, Elevation of Privilege). These vulnerabilities are associated with imperfect authorization and authentication mechanisms. As we will see later, they account for the majority of successful attacks.
* Tampering with data (Information Disclosure). An attacker can modify data or disclose it to unauthorized third parties. Unauthorized changes to data are challenging to track, even if the audit subsystem is connected. From my experience, this is the most unpleasant problem. Imagine a vast database of real estate transactions. Say, a land registry. In this database, the boundaries of plots have been updated, reflecting the history of real estate transfers from one owner to another. Moreover, en masse, including to confuse the tracks. You will need to review paper documents methodically and check them against the electronic database.
* Disruption of service (Repudiation, Denial of Service). These attack vectors are auxiliary because an attack is often a complex operation involving multiple subsystems. It is not without reason that an attacker attempts to compromise the logging and audit systems to evade monitoring systems. In turn, a DDoS attack (i.e., overloading a web service) can have different goals. From a primitive system shutdown that can damage a company to a more complex scenario. For example, if a third trusted service is involved in checking incoming requests, then disabling this service reduces the overall security of the system.

**Security principles for your application**

You should approach the protection of computer programs using common sense and an understanding of the paths from which trouble can come. We discussed the paths above. It is essential to remember that computer systems are created by people for people. Both can have different qualifications, work cultures, and varying attitudes towards their responsibilities. It is worth mentioning that most often, users are far from the IT world. They work for a company and help it conduct business: bake bread, manage tram traffic, and perform accounting operations. Anything that does not concern their business is of little interest to them and is perceived as a burden. And this is normal! The work of an IT system architect, in collaboration with developers and security specialists, is precisely designing a system that does not burden the user with unnecessary responsibilities but rather helps in daily work. It is worth remembering and constantly applying the following simple principles:

* Least Privilege. Determine a distribution of roles for system users. Do not give them more authority than required. The user should not have access to system nodes and data that are not necessary for work.
* Do not store credentials in repositories. Do not store debug data, passwords, or test users in repositories. Some time ago, GitHub restricted on this, but nevertheless, do not post sensitive data.
* Defense in Depth. Think about multi-level protection, similar to an onion. If an attacker picks the key to one door, another door will appear on his way, unlike the first. And so on. For example, data is only available from a local network, which must be connected to via a VPN. It is installed and configured only on the company's computer. Several credentials are required to log in to the computer, connect to the local network, and access the information system. The user has authority only for his own data.
* Fail safe / Fail Security. Whatever happens to a computer program when it malfunctions, it should not provide access to sensitive information or the details of the system implementation. Do not allow the user to access the system if authentication is compromised. Do not show pieces of SQL with a description of the table and query if it was not possible to change the data. At the same time, remember common sense. If a fire breaks out in the building, the electronic locks on all doors must be opened. Yes, this nullifies security, but it helps save people.
* Do not assume anything about a user (trust but verify). Each access to data or another resource must be authorized. It is not necessary to request a login and password for every action. There are standard token mechanisms that are attached to every request.
* We are humans. Once again, we remind you that users are most often far from IT. They are focused on their work. Therefore, it is essential to find a balance between convenience and security. A sense of proportion comes with experience and wisdom. Believe me, no one will remember a 25-character password that must be changed every three days. Yes, this is an excellent security solution, but rest assured that it will immediately appear on a sticker stuck to the monitor.

**OWASP TOP 10**

There is a long-standing rating of vulnerabilities found in computer systems. The order of elements rarely changes. That is, despite numerous security trainings, developers continue to make the same mistakes. Let's consider it in detail:

* Broken Access Control. This includes all cases of Spoofing and Elevation of Privilege. Incorrectly designed privileges, the ability to directly access website pages without authorization, etc.
* Cryptographic Failures. Lack of data encryption during storage or transmission. Use of outdated encryption protocols. Loss of control over encryption keys (for example, their transmission to the subscriber in an insecure way).
* Injection. Code injection into SQL, XML, and web pages (XSS). For example, SQL Injection becomes available when data from the input field is directly substituted into the SQL query without verification and sanitization. In this case, an attacker can enter a piece of SQL directly into the input field, significantly distorting the original query.
* Insecure Design. Everything about incorrect system design. Lack of logging and auditing, incorrect distribution of user roles, ill-conceived system architecture, etc.
* Security Misconfiguration. Incorrect configuration of individual services that are part of a single information system. I think that over the years, the role of this problem will only grow stronger, thanks to the microservice architecture and the abundance of various third-party services.
* Vulnerable and Outdated Components. We touched on this problem above. For popular components (for example, logging or JSON parsing), known problems can be used by attackers as entry points into the system.
* Identification and Authentication Failures. This is a specific issue of broken access control, in my opinion. It concerns inadequate policies regarding passwords (including encryption), session management, and other related areas.
* Software and Data Integrity Failures. These problems are partly related to Security Misconfiguration, but Software and Data Integrity Failures are much broader. The point is that using third-party services increases the security risk because you can't control the provider of any service (see Supply Chain Attacks). Risks associated with incorrect role distribution, weak passwords, and encryption can occur with each provider; that is, the problem is compounded.
* Security Logging and Monitoring Failures. Some of the problems discussed above are impossible to track without sufficient logging, auditing, and, of course, someone who will read this information and make decisions. Quite often, one or more of these things are simply missing.
* Server-Side Request Forgery. This is a special case of an attack, distinct from the systemic problems we previously discussed. This attack involves a successful attempt to force the server to disclose data that should not normally reveal to the attacker.