Development Principles of Secure Microservices

Oleksandr Tereshchenko[1] and Natalia Trintina[2]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," 37 Peremohy ave., Kyiv, 03056, Ukraine*
[2] *State University of Telecommunications, 7 Solomianska str., Kyiv, 03110, Ukraine*

Abstract

The subject matter of this article is to build secure applications using a microservices architecture. The goal is to provide developers with useful guidance on already recognized threats in microservices applications and how to detect, mitigate or prevent them. We also aim to identify potential gaps in security research in microservices architecture. The tasks to be solved are: identifying the most pressing threats for microservices and microservices architecture in general; selection of the set of approaches and security mechanisms that are used to detect, mitigate and prevent these threats. The methods used are: literature analysis, microservices application testing. In this article, we conduct a systematic scoping study to identify threats to microservices architecture and how to minimize them. We have highlighted the most critical threats and proposed solutions and methodologies for dealing with them from selected research studies. The following results were obtained. Methods for protecting data during transition between microservices are proposed, possible measures to ensure the security of data at rest in the database are considered. Some of the most vulnerable and complex processes in a microservices application, namely authentication and authorization, have been analyzed. Several approaches to authentication and authorization are proposed, and the security weaknesses and advantages of each of them are highlighted. In particular, approaches based on single-sign-on technology are considered. All approaches to the development of microservices applications, which were presented in this article, were tested during development of the microservices application and their positive impact on the security of the application was proved. In addition, we discussed how the software development practices, adopted by the development team, can affect the security of microservices applications and how to organize the process of development of a secure microservices application. Unfortunately, a review of scientific papers demonstrates that very little attention is paid to this topic. The scientific novelty of the results obtained is as follows: the most serious security problems encountered in the microservices architecture were highlighted, and clear steps were given on how to develop a secure microservices application. The relevance of the work lies in the fact that very little attention is paid to this topic, while the number of possible vulnerabilities increases as microservices spread.

Keywords

Microservices architecture, monolithic architecture, authentication, authorization, client certificates, single-sign-on gateway.

CPITS-II-2021: Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2021, Kyiv, Ukraine
EMAIL: alexandr.tereschenko2014@gmail.com (O. Tereshchenko); trintina2015@gmail.com (N. Trintina)
ORCID: 0000-0003-0536-2708 (O. Tereshchenko); 0000-0001-6827-4030 (N. Trintina)

1. Introduction

Microservices is a new architectural style that has become widespread in our time. Microservices offer many advantages over older monolithic architectures. Big business is already moving to microservices and this trend will continue, because the monolithic architecture cannot keep up with the requirements and challenges of modern software, which is now quite large and complex [1].

From a technical point of view, microservices are a specialization of the approach to the implementation of SOA (Service-Oriented Architecture), which is used to build flexible systems with independent deployment [2]. Compared to alternative software methodologies, microservices are actually a very cost-effective solution due to the limitations of older and more traditional software development methods, such as monolithic architecture. Monolithic architecture cannot mimic microservices without significant infrastructure costs and information unreliability. Despite the widespread use and rapid development, scalability has become the main condition for the success of the current widespread use of microservices technologies [2].

Data security is an important aspect of software development today. Ignoring this aspect will undoubtedly lead to serious technological and legal problems, as some countries have strict legislation to combat information leakage. An example of such legislation is the general personal data protection regulation in the European Union.

With massive hacking attacks on companies such as Netflix and Amazon, which implemented microservices architecture in their applications, security has become an urgent need. Several research papers have noted the need to study the security of microservices applications [3, 4, 12]. However, security threats are diverse and growing fast. Security proposals also increase and range from the protection of individual microservices to the protection of infrastructure as a whole.

In this article, we conduct a systematic review to identify the main threats to the security of systems based on microservices architecture. We analyze existing research on threats and solutions that facilitate the security of microservices applications. The objectives of this study can be summarized as follows:

- identification of the most severe threats to microservices and microservices architecture in general;
- selection of the set of approaches and security mechanisms used to detect, mitigate and prevent these threats.

2. Monolithic Architecture

Monolithic architecture is a traditional unified model for software development. The monolith in this context means that the application is compiled into a single fragment. The monolithic program is autonomous. In this case, the components of the program are interconnected and interdependent, rather than weakly coupled, as is the case with modular programs.

As you can see in Figure 1, although this is a fairly simple model, the limitations are obvious and do not stop at scalability. They go far beyond the most important modern aspect, namely security.

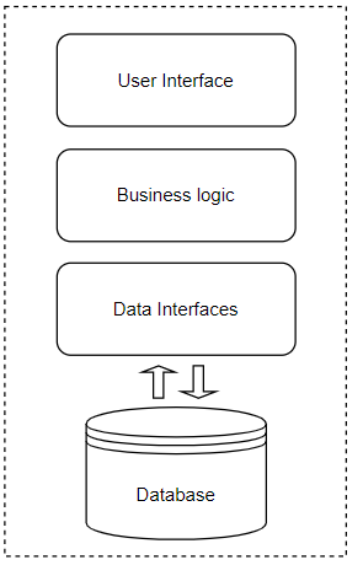

Figure 2: Monolithic architecture

When using a monolithic architecture, we have to compile the entire assembly with minor changes to the code, which significantly slows down the deployment and delivery of updates. When using microservices, you can change only one service and compile it, which is a more attractive solution. Moreover, there is a risk factor that today's large vertical scaling programs can have serious crashes or errors that compromise the operation of the entire application. In microservices, this risk levels, as the architecture supports horizontal scaling. It is important to keep in mind the previous points, but they are somewhat secondary to the possibility of stealing limited data due to their targeted and centralized topology or budget problems related to the fact that monolithic development usually uses one specific technology or supplier.

3. Microservices Architecture

In contrast to the monolithic architecture, which is a single unit, microservices architecture divides the application into a collection of smaller independent units [5]. These blocks implement each business process as a separate service. Therefore, all services have their own business logic and database. This contributes to security by making it possible to use an unlimited number of programming languages, APIs, and scaling resources in both directions when resources are needed or not. See Figure 2.

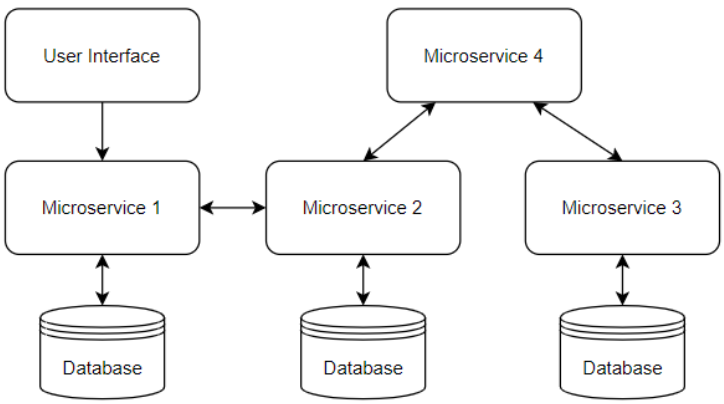

Figure 2: Microservices architecture

It is important to remember that microservices are not a trouble-free technology, methodology or architecture, so there are issues that need to be overcome. This architecture requires more development

control. There is a need for a larger development management team, and its size depends on the number of services involved, vendors, programming languages, databases used during the development process, as well as the high complexity of testing. Dividing into several development teams is common for this type of applications.

We need to realize that these are not all the advantages and challenges of microservices architecture, so it should be borne in mind that for the successful use of this architecture we require a well-established interaction between developers and architects [5-6].

4. Properties of Microservices

Microservices have very important characteristics, and some of these characteristics must be considered in the development of secure services. So let's look at these characteristics in more details.

4.1. Scaling of Microservices

The fast scaling process has become a feature of microservices that has made them so popular in software development. Microservices can independently scale on the X axis, for example, by duplicating services, or scale on the Z axis, which is also called partitioning, which is the allocation of different data of the same type on different nodes [4]. Figure 3 shows the "Scale Cube":

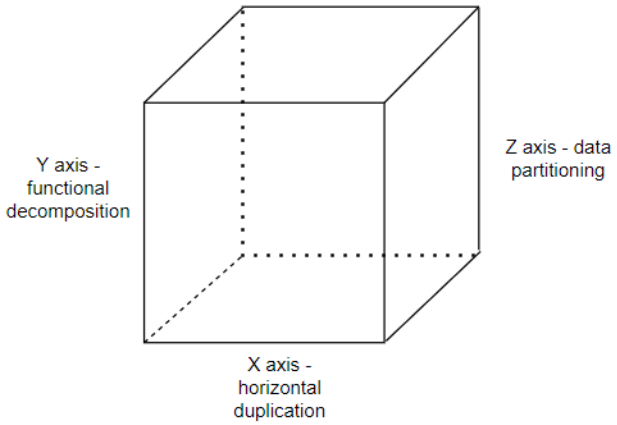

Figure 3: “Scale Cube”

As an example of scaling along the Z axis, we can give an example of the division of large databases into smaller parts, which can speed up and facilitate the management of the application [6]. In turn, this process is the opposite of monolithic applications, which usually have a single database.

4.2. Possibility of Independent Updating

When updating a monolithic application, you need to recompile the entire program. In the case of microservices, each service can be deployed independently of other services. Any change in the service can be easily made by the developer without the consent of other development teams. Microservices architecture ideally promotes the application of continuous integration and continuous delivery [7].

4.3. Simplicity of Maintenance

If you follow the approach of object-oriented design, the microservice code will be limited to one entity, so it is easier to understand than the code in a monolithic architecture. Working with smaller code bases increases the speed of development and allows you to have a real idea of the side effects of the code that programmers write [7].

4.4. Multilingualism and Heterogeneity

Programmers have the right to choose which programming languages are most suitable for their service, freely implementing innovations within the service [7]. Developers can quickly rewrite the microservice code using the latest technologies and tools.

4.5. Failures and Isolated Resources

Any problem in the monolithic architecture, such as an error during connecting to the database, will lead to reduced performance or even a complete failure of the program. However, in the case of microservices architecture, these problems apply only to this service. Microservices isolate failures and limit the extent to which a failure can affect the program. In the case of well-designed microservices, failures are isolated in one service and do not extend to the rest of the system, thus not causing inconvenience to the end user.

5. Security Issues of Microservices Applications

Microservice protection must overcome a huge number of trust and security issues. These are not the new problems. They were inherited from close relatives of the microservices architecture, such as SOA and distributed computing.

The main problems here will always be programming habits and the human factor, and they are becoming more acute, as microservices usually turn a simple application into a large area for a hacker attack.

As mentioned above, microservices completely trust each other, so compromising and effectively operating one of them can lead to complete compromising of others [7]. Microservices applications have many entry points and their protection is a priority. Therefore, the question arises: what are the risks and how can they be eliminated at an early stage of development or how to minimize their consequences?

5.1. Security Issues in the Clouds

The use of microservices implies that these programs extend to multiple services or platforms located in public clouds, such as PaaS, SaaS or IaaS, so security is a problem due to the complexity of development, monitoring, debugging and auditing of the entire application in a foreign environment. This is due to the fact that the level of transparency is standard for these types of services and is determined by providers [8]. Therefore, developers can only blindly believe in the security of the platform or service.

If an attacker compromises a particular service by exploiting a vulnerability in the public microservices, he or she can take control of the virtual machine in which the microservices run. As a result, some microservices may not be credible [9]. If a subdomain of a particular application is compromised, an attacker can access cookies and obtain confidential information. That is why it is important to pay a lot of attention to choosing a cloud service provider.

5.2. Web Applications Vulnerabilities

Microservices are the evolution of standard computer platforms. A simple conclusion follows from this: what was previously a vulnerability remains the same vulnerability in the microservices application [9]. To increase awareness of the most common vulnerabilities, developers should be aware of the ten most critical vulnerabilities in an open source Web application security project, the OWASP Top Ten list and the OWASP Security Testing Framework [10].

All of these vulnerabilities are listed below:

- Injections.
- Cross-site scripts (XSS).
- Problems with the authentication system and storage of sessions.

- Problems with access control.
- Security Misconfiguration.
- Insecurity of sensitive data.
- Dangerous direct links to objects.
- Cross-site request forgery (CSRF).
- Usage of the components with known vulnerabilities.
- Problems with journaling and monitoring.

One aspect that deserves a more detailed description is SSRF (Server Side Request Forgery). This aspect needs attention despite the apparently low risk [10].

In a typical SSRF attack, attackers can force the server to connect only to internal services in the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially stealing sensitive data, such as authorization credentials (Fig. 4) [11].

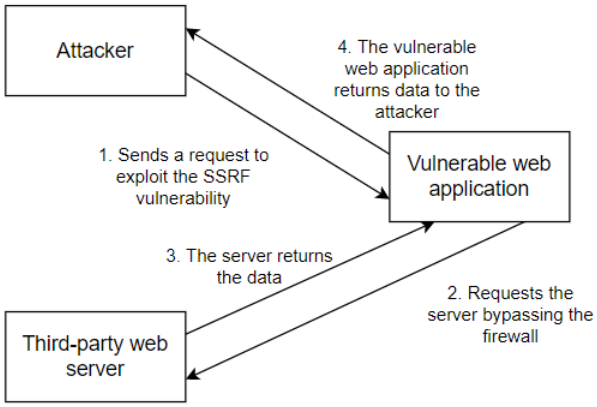

Figure 4: Server Side Request Forgery

In microservices, inter-server requests are usually secure if deep-layer firewall defense or network segregation is used. If these aspects are not met, you may end up with a server-side request forgery situation in which an attacker can abuse trust between servers, bypass IP whitelists, authentication services, and read resources that are not available to the public. Moreover, attacker can interact with API or receive sensitive information, such as the IP address of a web server [11].

5.3. Authentication and Authorization

When we deal with people, authentication and authorization play a big role. Authentication is the process of confirming that a user is the one with whom he positions himself [12]. In turn, authorization is a mechanism that classifies and allows a person to perform some actions in the system.

In the case of monolithic systems, the application itself is responsible for authentication and authorization. Web frameworks such as Angular, Spring, Django, come with a ready-made user management system, which facilitates the implementation of this task in monolithic systems. When it comes to distributed systems, due to its complexity we have to look for better schemes. You should avoid the option where the user has to log in to each system separately, using different passwords and usernames. The way out of this situation may be to create and use a unique identifier that allows authentication only once [12].

The peculiarity of interservice communication in microservices architecture is that microservices completely trust each other. The one of many possible ways to securely authenticate and authorize services is to use single-sign-on gateways, as they avoid the use of libraries that rely on a single code base, while helping to reduce the number of duplicate codes.

5.3.1. Usage of Single-Sign-on Technology in Microservices Architecture

A common approach to authentication and authorization is to use any of the single-sign-on solutions. Relevant capabilities in this area are provided by the open authentication and authorization data exchange standard SAML and the open decentralized authentication system standard OpenID Connect.

SAML is a standard based on the SOAP protocol, but it is considered quite complex, despite the availability of a large number of libraries and tools. OpenID Connect is a standard that emerged as a specific implementation of OAuth 2.0 and is based on the methods of managing single-sign-on technology adopted by Google and a number of other companies. It uses simple REST calls [12].

A directory service can be any tool such as Active Directory or Lightweight Directory Access Protocol (LDAP). These systems allow you to store information about the role of users in the system.

SAML and OpenID are ideal for authentication and authorization of users in the system, but have also become widespread for authentication between services. It is extremely important to promote services and mechanisms that force users, administrators and developers to use complex, unique passwords [13].

5.3.2. Usage of the Single-Sign-on Gateway

Instead of duplicating the logon control logic on each service, you can pass this work to a separate gateway that will act as a proxy server and be located between the services and the external environment. Thus, it is possible to localize and centralize the logic of user redirection and execution of confirmation in one place [13].

But in this case, there is a problem which related to a single point of failure.

5.3.3. Usage of the HTTP(S) Basic Authentication Standard

With HTTP Basic Authentication, clients have the ability to send a username and password in a standard HTTP header. After the server processes the data, the server may grant a permission to access the service to the user. But there is a problem with the lack of security for these headers. Any intermediate instance can view the information in the header and see the data. Therefore, HTTP Basic Authentication should be used with the HTTPS protocol.

By using HTTPS, the client is guaranteed that the server, which the client is communicating with, is the same server that the client wants to connect to. The HTTPS protocol protects the request from "man-in-the-middle" attacks, making it almost impossible to manipulate its payload [13].

It can be problematic to manage multiple machines, because the server needs to manage its own SSL certificates. Tools for automated management of the certification process have not reached sufficient perfection yet. Try to completely avoid using self-signed certificates. Another disadvantage is that traffic passing through SSL cannot be cached by reverse proxy servers such as Squid or Varnish [14].

5.3.4. Client Certificates

As an alternative to the above, and despite management difficulties, the approach to client identification may be the use of the capabilities provided by the descendant of the SSL protocol, Transport Layer Security (TLS), which helps to generate client certificates. Each client has an X.509 certificate, which is used to establish communication between the client and the server. The server verifies the authenticity of the client certificate, thus providing strong guarantees of client reliability.

5.3.5. Usage of HMAC over HTTP

Excessive HTTPS traffic can cause additional load on servers. In addition, there are difficulties with caching such traffic. An alternative way to sign a request is to use a hash-based message authentication code (HMAC) [14].

When using HMAC, the request body is hashed with a private key and the received hash is sent with the request. The server then uses its own copy of the private key and the request body to recreate the hash. If the hashes coincide, the request is accepted. This eliminates the possibility that the request was forged. An additional advantage of this method is that the traffic is much easier to cache and the cost of generating hashes can be much lower than the processing of HTTPS traffic.

5.3.6. Usage of API Keys

API keys are used by all open APIs of services such as Facebook, Google, Twitter, Google and AWS. API keys allow the service to identify the caller and restrict what they can do.

5.4. Security of Data at Rest in the Database

Protecting data that is not used and stored in the data stores is a fundamental necessity. Many hackers' attacks choose the application's data at rest as their ultimate target, and this is made possible by the vulnerability of their locations. Therefore, no matter how carefully the deep-layer defense has been implemented, the placement of data in encrypted form should be guaranteed. Regardless of the programming language used, you need to use the implementations of encryption algorithms that are constantly being reviewed, improved and have earned a good reputation. The encryption of data, for example, by using the AES-128 or AES-256 algorithm, should only be when the program does not use it, and decrypt it during data processing [12,13,15,16].

5.5. Deep-Layer Defense

As a deep-layer defense in a microservices application, it is appropriate to use one or more firewalls. Typically, firewalls are quite simple and aim to restrict access to certain types of traffic to certain ports. The firewall is probably the last line of defense when other lines miss the threat, so the architecture of the microservices application should place the firewall in front of the main service levels [12].

Sometimes it is advisable to use more than one firewall. An example is when you need to secure a host using IPTables by setting up valid inputs and outputs. In turn, the main firewall will be located around the perimeter, which will control public access. The ability to segment networks based on team affiliation can also come in handy when building deep-layer defense.

5.6. Tracking and Logging

Tracking and logging systems have already become a security classic. They allow you to quickly detect system failures, which is an important aspect of quality of the customer service. In addition, the analysis of logs provides administrators and developers with information about the operation of the system and its weaknesses. In turn, this allows you to detect security vulnerabilities in the microservices application. QoS technology and its solutions can be useful for monitoring the interaction between services.

5.7. Human Factor

Everything described earlier in this article was related to the technical side of security. But the human factor, which in most cases is the cause of security problems, cannot be ignored. Developers should be taught to build security according to specific standards, such as ITIL, TOGAF, BSIMM or COBIT.

6. Conclusion

The purpose of this article was to describe the vulnerabilities and security requirements of the microservices architecture and to analyze possible ways to build secure microservices applications. It is important to remember that the study and implementation of security concepts is an important step in building reliable information systems today. Unfortunately, when it comes to security, we can responsibly say that there is no system that is completely safe and secure. Much is left out because of the enormous possibilities of microservices architecture, and much of what has been discussed in this article is just the tip of the iceberg, because of the enormous possibilities of microservices architecture, so this topic needs further study.

7. References

[1] N. Anil, J. Parente, M. Wenzel, Microservices architecture, 2018. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture.

[2] S. Fowler, Scalability and Performance, 2019. URL: https://www.oreilly.com/library/view/production-ready-microservices/9781491965962/ch04.html.

[3] P. Nkomo, M. Coetzee, Software Development Activities for Secure Microservices, Computational Science and Its Applications (2019) 573-585. doi:10.1109/CLEI47609.2019.235060.

[4] S. Sultan, I. Ahmad, T. Dimitriou, Container Security: Issues, Challenges, and the Road Ahead, IEEE Access Volume 7 (2019) 52976-52996. doi:10.1109/ACCESS.2019.2911732.

[5] T. Rosner, The Age of Microservices, 2018. URL: https://www.trinimbus.com/blog/the-age-of-microservices-amazon-ecs-service-discovery.

[6] N. Dragoni, S. Giallorenzo, Microservices: yesterday, today, and tomorrow, Springer, Berlin, 2017, pp. 195-216. doi:10.1007/978-3-319-67425-4_12.

[7] C. Richardson, Microservices patterns, Manning, 2019, pp. 317-331. doi:10.1002/spe.2608.

[8] R. Chen, S. Li, and Z. Li, From Monolith to Microservices: A Dataflow-Driven Approach, 24th Asia-Pacific Software Engineering Conference, Nanjing, China, 2017, pp. 466-475. doi:10.1109/APSEC.2017.53.

[9] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015, pp. 213-224. doi:10.5555/2904388.

[10] D. Wichers, J. Williams, Owasp top-10, 2017. URL: https://wiki.owasp.org/images/1/17/OWASP_Top-10_2013--AppSec_EU_2013_-_Dave_Wichers.pdf.

[11] N. Team, What is the Server-Side Request Forgery Vulnerability & How to Prevent It, 2019. URL: https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf.

[12] H. Abdelhakim, S. Yahiouche, Securing microservices and microservice architectures: A systematic mapping study, 2021. doi:10.1016/j.cosrev.2021.100415.

[13] P. Johnson, Microservices Architecture: Security Strategies and Best Practices, 2021. URL: https://www.whitesourcesoftware.com/resources/blog/microservices-architecture.

[14] A. Pereira-Vale, G. Márquez, H. Astudillo, Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping, XLV Latin American Computing Conference, Panama, 2019. doi:10.1109/CLEI47609.2019.235060.

[15] I. Bogachuk, V. Sokolov, V. Buriachok, Monitoring subsystem for wireless systems based on miniature spectrum analyzers, in: 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (2018). doi:10.1109/infocommst.2018.8632151.

[16] Kipchuk, F., et al. Investigation of Availability of Wireless Access Points based on Embedded

Systems. 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), 2019. https://doi.org/10.1109/picst47496.2019.9061551