Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 181 (2021) 1225–1236

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2020

Security in Microservices Architectures

Nuno Mateus-Coelho[a,*], Manuela Cruz-Cunha[b], Luis Gonzaga Ferreira[b]

[a]*ISLA – Polytechnic Institute of Management and Technology, Vila Nova de Gaia, 4400, Portugal*
[b]*IPCA – Polytechnic of Cávado and Ave, Barcelos, 4750, Portugal*

Abstract

A Microservice is a small or even micro independent process that communicates, acts, and returns via messages through lightweight mechanisms like Thrift, HTTP or RESTAPI. Microservices Architecture is amateur evolution of the Monolithic Architecture. Observing it in afunctional way, it is correct to claim that it breaks down complex applications into a simpler abstraction. As this research demonstrates, Microservices Architecture is intrinsically connected as a symbiosis with container-based deployment, because these containers have no need for embedded operating systems and calls are made for OS resources, via an application programming interface. It is safe to claim that this technology is currently the focus of modern developers nowadays. Semantically speaking, Microservices functionally deconstruct larger applications into smaller, discrete services, and containers are viewed as a natural compute platform for this architecture [1]. A single service is and can be represented by multiple containers in a Microservices cluster, each single service is designed to provide a specific set of functions while services act to makeup the entire application. It's common in large application the decomposition into multiple arms of more than twenty services, although less can be commonly found as well [1]. The main objective of Microservices Architecture is to disassemble the core components of a given type of application [2]. This study could be done in multiple ways, all of them different because practically everybody has their specific way of looking at Microservices, but one aspect is the same cross mentalities, Security. So, the focus of this research is to expose the main security aspects of this specific architecture, in this cost-effective era.

Peer-review under responsibility of the scientific committee of the CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2020

* Corresponding author.
E-mail address: nunomrcoelho@engenheiros.pt

1877-0509
Peer-review under responsibility of the scientific committee of the CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2020
10.1016/j.procs.2021.01.320

Keywords: microservices; security; REST; API; web service; monolithic; architecture; hackers

1. Introduction

Most people think that they have never seen Microservices before. It is a cutting-edge technology and surely a thing from the future. Please analyze figure No.1 and rethink this small statement. One of the best characteristics of the latest generation is the fact of a unique phenomenon, a generation raised with easy and fast access to information, eager for knowledge and with consumers who are increasingly demanding for excellency and transparent software products, delivered in an incredibly shorter time, with more and better construction, presentation, features and higher quality than the previously used. [3] For this reason, the number of software products made available in recent years has grown considerably, available everywhere through the advent of smartphone, mobile resources and now with the IoT. With this increase, there is a need for greater demand regarding the quality of applications development and their architecture design. As it's possible to realize when looking to the airplane and all the independent components controlled individually and still able to connect each other and to the control tower to be a part of a system, the concept is something that already existed, even in computer sciences e.g., networks, and was catapulted to software development.

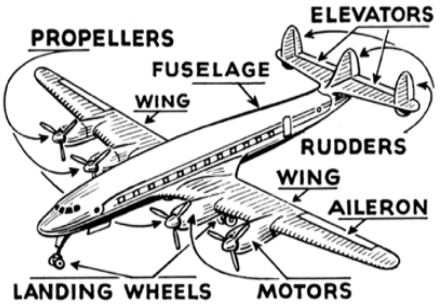

Figure 1 – Airplane components (http://mariorosary.com)

Current methods of approach to software development life cycle changed significantly over the last decades and particularly, over the last years. There is a need to ensure security and adaptation to the market demand for solid and robust information technologies, increasingly demanding for process quality standards within the extremely complex development cycle. As a result of the previous stated fact, universities, research labs and companies intend to ensure an adequate response to stakeholders' requirements, with high levels of availability, usability, reliability, resilience and support of the software products or services they produce, while streamlining the processes and performance of the software engineers and their development teams.

Microservices exists in software development because someone in some business department found a risk on their products and place the correct question. - What if in an application developed in Monolithic Architecture known as common 3 Tier, fails? What would have happened, or what part of the business would be impaired? How can it be overcome so parts of the services provided by the development can still function even if parts of the development are not working?

Technically, Microservices is a specialization of an implementation approach for SOA-Service Oriented Architectures used to build flexible, independently deployable software systems, and despite there is no confirmed owner, inventor, or standard to specify it, there is already a common settlement of whom started to use the term and started to aggregate the ground rules that allow to mature the specification today [4]. Looking back in time and analyzing the present it is easy to comprehend why this technology is widely used and a profitable business to vendors like Amazon and Microsoft. This is due to the advantageous genesis represented by its multi-cellular kind working and evolving within the SDLC in fully independent way. Comparatively with other types and alternatives to software methodologies, Microservices are in fact a success case due to the limitations of the elder and more traditional methods e.g., Monolithic Architecture, hence these cannot mimic Microservices without heavy infrastructural costs and

information insecurities. Despite massive use and years of maturation, the scalability came first as a success condition to the current spreading of Microservices architectures and technologies.

1.1. Monolithic Architecture

Currently the main server-side development programming languages vastly used in academic or professional ambience are Java, C#, C/C++, Ruby and Python, due to the capability of providing linear and concrete abstractions that in fact reduces complexity of programs breaking them down into singular modules. This is the core objective, ability, and purpose, designed for the creation of unique executable artefacts, commonly known as monoliths, and their modularization abstractions rely on the sharing of resources of the same machine [5].

As it's possible to observe in figure No. 2, despite being a simple model of use, the limitations are obvious and do not stop at scalability, they go far beyond and cross the most important nowadays aspect, security. The Microservices approach is a first realization of SOA, that followed the introduction of DevOps and is becoming more popular for building continuously deployed systems. One of the most famous transition from Monolithic to Microservices was conducted by Netflix in 2010 when they started to use the AWS Amazon to host their application and services in more than 100 grained services, instead of common a web-app (.war). Today other companies like eBay use it. [6, 27].

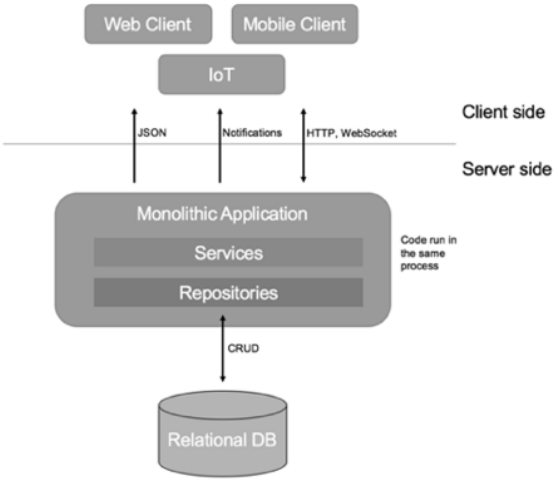

Figure 2 – Monolithic Architecture

The success of Microservices lays where Monolithic fails, e.g., problematic issues like having the need to make a build of an entire development just to alter a small constraint or check, or the risk factor that today, large vertical applications might have major fails or bugs that compromises the entire application purpose. A monolithic development is a full stack or a vertical alignment of an object, thus requiring more resources, but in fairness, for small and simple point applications, it is more than viable solution, it is the correct one. The previous points are important to consider but somewhat became of small importance when compared to the possibility of getting the data confined to them stolen, due to its centered typology or the budgetary issues surrounding the fact that Monolithic development normally lay in one specific technology or vendor.

1.2. Microservices Architecture

In Microservices, as it is possible to observe in figure No.3, the applications are no longer dependent on itself, they have the ability to possess and process more than one interface and can spread across multiple infrastructures instead of a vertical development or stack. This facilitates and promotes security making possible the use of an unlimited number of programming languages, API, use of serverless techniques and scale up or down resources, when services are required or not. Also terminate some problematic issues with classpath dependencies in large applications, where

everything depends on everybody in a dependency's nightmare scenario, quite familiar to most developers. There is a great similarity with Serverless Architecture and is now a teaching subject in schools, and is complex, as it is simple.

It's important to retain that this is not an issues free technology or methodology or architecture, hence there are great challenges to overcome. This architecture requires a larger capacity of control as developers are no longer looking to a building but to a city with multiple governance and municipalities that must work together to function, despite all its constraints. There is a need to have a larger control unit team and the size depends on the amount services, vendors involved, coding languages, databases spread across the development and also, great testing complexity because it is common for this type of project to be spread throughout multiple teams [30].

Above all the things that can be concluded about Microservices, this is just a small aspect of the advantages and challenges of Microservices Architecture over the Monolithic Architecture. It is paramount to keep in mind that not everything is linear regarding this architecture, and realize that human interaction is mandatory to use the architecture, and where is human hand programming, there is always complications due to limited scope of action and condition.

2. Microservices Properties

Microservices are composed of very important characteristics for software implementation and some of these characteristics must be considered in the process of developing services that use this architecture, to be known in the following points.

2.1. Scaling Microservices

Some of these elements are what make the use of this method a principal option for developers and the main one is the fast scalability process[1]. This process enables Microservice to scale independently through the scale of the X axis e.g., increase of CPU or memory, also called vertical scale, or scaleup or the Z axis scale or *sharding*, called horizontal scaling or scale out, as shown in figure No.3, from Martin Abbott and Michael Fisher:

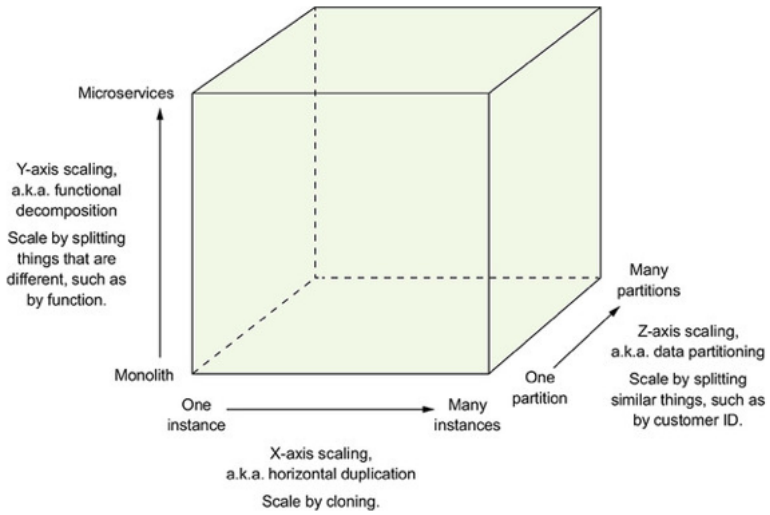

Figure 3- Sharding monolith to microservices basic example

As an example of scaling, the partitioning of a databases, separating large databases into smaller parts, enabling a faster and more easily manageable application. This is the opposite of monolithic applications, which can have far more distinct requirements, however, usually have a single database [8, 30, 31].

2.2. Independent Update Feature

Each service can be deployed independently from other services. Any change in the availability of a service can be easily done by a developer without the need for coordination with other development teams or members. It is also a facilitator for the implementation of continuous integration and continuous delivery [31].

2.3. Simple Maintenance

The code for a microservice is restricted to one capacity, so it is easier to understand than in monolithic architecture. IDE's can easily load small amounts of code, making the build lighter. Working with smaller code bases increases the speed of development and allows to have a real idea of code side effects that programmers are changing or developing [36].

2.4. Heterogeneity and Multilingualism

Programmers are free to choose which programming languages are most suitable for their service, being free to innovate within the service's limits. This allows the rewrite of the service using the latest technologies and enables the freedom to choose the technology, tools, and structure [35].

2.5. Failure and Isolated Resources

A poorly built service in a monolithic architecture, with problems such as lack of memory or an unconnected database connection, will generate a performance breakdown or even complete failure of the application, however in a Microservices architecture scenario, only that service is affected. Microservices isolate failures and limit how much an application can be affected by a failure. With well-designed microservices, failures are isolated in one service and do not spread to the rest of the system, thus not demonstrating weakness to the end user [35].

2.6. Improved Communication Between Teams

A microservice is usually built by a Full-Stack team. All members related to a domain work together in a single team, which significantly improves communication between team members. It becomes advantageous because they share the same final goals, offer cadence and, perhaps most importantly, the service as the product of the team [31].

3. Security Exposed

"With great power comes great responsibility". This might not be a familiar phrase to most people but is surely one to most developers that use new technologies. An architecture is something that allows a given subject e.g., serverless, to evolve and mature in a secure and trustworthy path. This permits to spread the acquired knowledge base of the given subject in a way that includes all the mandatory aspects to an effective and solid evolution. Securing Microservices has an immense number of trust and security challenges to overcome. This are old topics and were inherited from close relatives like SOA and distributed computation [37].

The main topic will be always programming habits and human failure, and it increases as Microservices normally transform a simple app into a large surface attack area under many people respectively responsibilities. Looking at this previous example, systems administrators, database administrators, cloud solution providers and API gateway middleware spread over network [9], all must communicate and act like an orchestra but playing each instrument over a streaming service without fails.

It is important to keep in mind and important aspect, Microservices are often designed to trust on their pears and compromising and effectively exploiting one, can lead to a full advantageous disclosure of the others [32]. Hackers know how systems work and ultimately know users' habits, and as it's possible to observe in figure No. 4, there are many points of entry in a common Microservices project, so securing them is of the up most importance. Considering the scope in this research, the question that is probably in the reader mind is: - What are the risks and how can they be addressed in an early phase or minimized after an attack?

To address the previous question, this research is an output of the Altran 5G Lab, where the points of this research where implemented and tested in a controlled environment and presented as solid countermeasures or robust recommendations.

3.1. Risks

Using Microservices implies that given applications spread across several services or platforms, many of them in public cloud as PaaS, SaaS or IaaS so, securing them is a challenge due to the complexity of the developments, the hardness of monitoring, debugging and auditing of the full application in foreign environments, and will be more challenging due to the fact of a layer of transparency common to these types of services that must be hidden because it's the vendors business. In a cloud environment, using third party solutions completely out of reach in terms of security, denying developers the control or the access to the purchased solution, leaving them with the product without the secrets that composes it, is asking them to take a leap of faith, security-wise speaking. They can use these services to host their Microservices like database, API gateway at their own risk and they make sure that this is well transmitted in their EULA- End User License Agreement. Taking advantage of this panorama, hackers take benefit of the lack of abstraction to attack applications using this kind of architecture spread across multiple solutions, and cloud vendor still lack mechanisms to control or support clients (applications developers and owners), keeping track of their distributed system with all the components that trust each other.

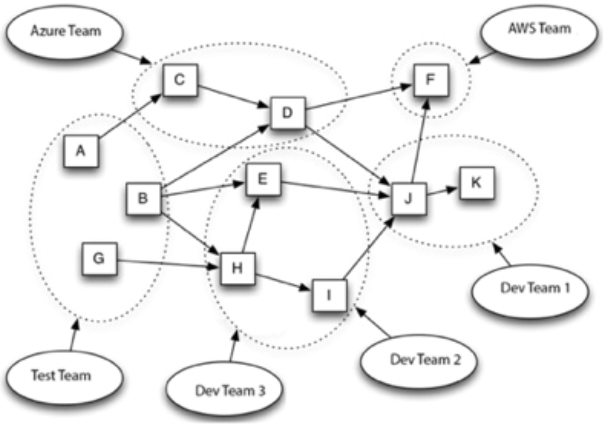

Figure 4 - Microservices area example

If and attacker compromises an individual service by exploiting a vulnerability in a public facing Microservices and escalate privilege on the VM that the Microservices runs in, catastrophe may occur with a simple abuse of privilege. As a result, individual Microservices may not be trustworthy [10].

It's important to question our own judgement when choosing a partner for the use of cloud services. Questions regarding the way they keep security trusted and how they do it, it is important, no matter the size or quality of the vendor. Size doesn't matter and everybody is a target when security is the subject. If sub-domain of a given app is compromised, and from that domain an attacker serves any content in the context of the webapp domain, if all users' cookies can be accessed from any sub-domain, an attacker controlling a sub-domain can tamper with authentication and ultimately the data [9]. Secure development is something that requires allot of effort and dedication from a development team, nowadays, the top key to have in mind while developing using Microservices are observable in the following examples [11].

3.1.1. Password Complexity

It is vital to promote services and mechanisms that enforce users, administrators, and developers to use complex, unique and time stamped passwords. There are no maximum requirements here, everything must be highly complex and integrated with extreme density using available software engineering procedures, infrastructure applications like

Active Directory, OpenID, or LDAP services. All characters of a possible password must be securely checked to avoid any use of previous strings, names, account names, sequence numbers or letters. Enforce the use of upper-case characters from A to Z with diacritic marks, Greek, and Cyrillic; non-alphanumeric characters or any Unicode that is categorized as an alphabetical character but is not upper-case or lower-case.

3.1.2. Authentication

Sometimes authentication is wrongly not considered a top priority while developing and the result can be serious. These days' apps cross each other using API or another federated authentication method to facilitate user interaction. User prefer transparency and these kinds of methods are all they need [10, 29]. There is a tendency to overlook such issues at heavy cost and recently, with a financial heavy cost, due to the fact that most apps possess mechanisms to process purchases. It's important to lock accounts after a very short number of fail attempts of login and verify if third party software integrated or at use via API does the same. For critical services, e.g., health, banking and military, authentication must be assigned and secret instead of user-defined public data [13]. It is important to remove ways that allow to enumerate user accounts and password when prompted for reset and use the multi-factor authentication services in separated Microservices.

3.1.3. Web Security Flaws

Microservices are not more than an escalation and an evolution of standard computer platforms and services by them provided, so what used to be a single point of vulnerability now is a spread one, but still the same vulnerability. The key point to retain in this risk is that security needs to be acknowledge and well discussed by architects, developers and project managers, and the subject of discussions must be the OWASP TOP 10 issues distributed scenarios [35].

3.1.4. People and specific processes

As the reader of this document can comprehend by now know, nothing is bullet proof and human effect surrounds everything in a godly aura, affecting and infecting in a positive and negative ways systems. There is a mandatory need to ensure that the development teams "build security in" recurring to specific frameworks and standards like TOGAF, ITIL, COBIT or BSIMM [38]. Responsible managers must design practical training teams capable of transmitting information instead of dumb boring hours do outdated information and resources. Promote a training catalogue that includes explicit activities of security principals and secure software development so that designers, developers, testers and operational resources can learn complex and basic principles like access control. The key here is the use of PDCA - Plan, Do, Check, Act and the objective is a strong internal Microservices operational and semantic security policy, consisted in [39]:

- Business Continuity Planning
- System Access Control
- System Development and Maintenance
- Physical and Environmental Security
- Compliance–Personal Security
- Organization Security–Systems & Network Secure Management
- Asset Classification and Service Control

3.2. Attacks

It is pointless to have in this work a full explanation of what kind of attacks exists and how they hurt Microservices applications directly in the core. There is a list with the TOP 10 Most Critical Web Application Security Risks by OWASP and in Microservices, they all apply. In the following list are the release candidates for 2019:

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS
- A4 Broken Access Control
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Insufficient Attack Protection

- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Under protected API's

One aspect is worthy of a closer investigation is SSRF. When approaching security, it must be done in the broad sense of its aspects despite an apparently low risk. Microservices spread a given app data source through services in a network of app-server requests, reducing the necessity of heavy using a given database or service hosted in a server e.g., a system deployed over an infrastructure or IaaS. Commonly these database or services are spread into multiple small data requests, or the app processes several databases where it stores and retrieve information, maintaining redundancy or fast services, as tables are stored in several engines of multiple types.

Despite applications in Microservices spread around the information into small pieces of data, this doesn't mean that SSRF is not a risk hence applications can trigger inter-server requests to fetch resources that can translate to software at its completion, exploiting the trust link between server and services. Please observe figure No. 5.

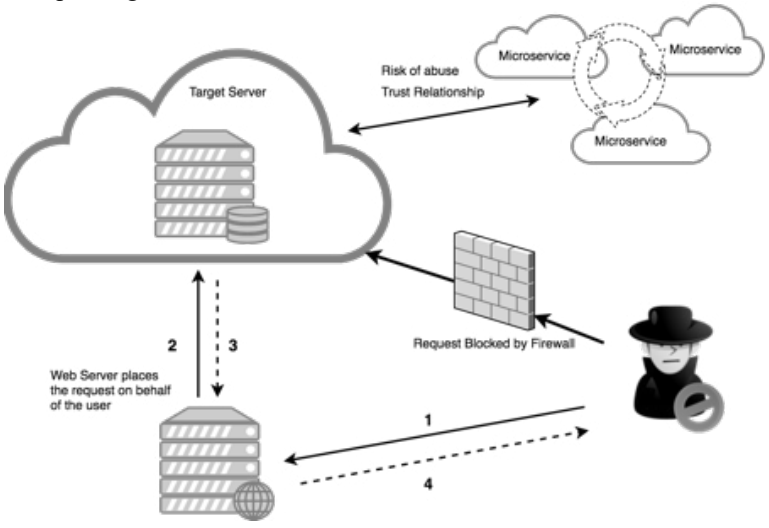

Figure 5 - Typical SSRF Abuse

In Microservices inter-server requests are commonly safe if the following aspects of this research are applied, e.g., using in depth defense with firewall or network segregation. When these aspects are not complied, the result might end up as a Server Side Request Forgery where an attacker can use the smallest advantage and abuse the trust relationship between servers; bypass IP whitelisting, bypass host-based authentication services, read resources which are not accessible to the public, such as trace.axd in ASP.NET or metadata, APIs in an cloud environment, scan the network to which the server and servers are connected to, interact with API's or retrieve sensitive information such as the IP address of a web server behind a reverse proxy [14].

These points are the most common in typical app's/web app's and rarely observed compromising a Microservices applications hence Microservices architecture is meant to prevent exactly an entire app compromise. Nevertheless, it's a risk worth following due to aggregated efforts from attackers' groups.

3.3. Top Countermeasures

The objective is clear, use minimal privileges as an objective set of rules to establish trust boundaries, identify, minimize, and harden the spread surface attack area, control and reduce the most possible scope and access, force layer protection defense system that works in depth, force architects and developers to control their levels of thrust and excess of it. Data needs to be protected not only at the logic point but also in transit from point A to point B. These previous topics are the main poits and broadly recommended by many experts like Bruce Schneider [15], Sam Newman [16]. The following examples are based on Michael Hofmann [17], IBM Redbooks [18], James Lewis [19] and Martin Fowler [20] research findings to tackle the above issues:

3.3.1. Authentication and Authorization

The Authentication process is the part of the process that confirms that a given subject is who he/she claims to be, and the Authorization consist in the mechanism that classifies and allows a given person the actions that he/she can perform in each system. This is less complicated in a monolithic application because it is common for the application itself to handle authentication and authorization. Spring, PHP, UserSpice, Angular, Django, Python and other languages and frameworks allows to set user management in simple secure steps [32].

In Microservices it is required to think of more advanced schemes. It is inconvenient that everyone might have to login separately for different systems, under a various usernames and passwords for each, and having complexity here by forcing a broker to do this job. Sometimes abstractions are powerful tools. The objective is to promote and deploy systems with a single identity, able to promote authentication in a trustworthy way. It is important to retain that, Microservices fully trust each other as definition. Among the many possible ways to have a strong Authentication and Authorization, is the use of Single Sign On gateways because these can avoid the use of libraries that, despite they help to reduce duplicated code, rely in shared one. Gateways can act as a proxy managing handshaking with identity providers just by sitting in the middle of internal and external services. The objective is centralized behaviors for redirecting the user and perform the handshake in only one place.

3.3.2. Service-to-service Authentication and Authorization

In Microservices, services must communicate with each other in an implicit way and are exposed to a man-in-the-middle attack. It is advisable to use HTTPS instead of HTTP basic authentication not just due to the fact of encrypting user and password information. Using HTTPS guarantees that a given client is communicating with whom he wants to, providing additional protection against people eavesdropping on the traffic between client-server or messing with the payload.

This approach is secure but complex and risky nowadays. Servers need to manage SSL certificates which is complicated in a multi-machine scenario, and there is a need to handle risky and complex issuing processes and some certificates are difficult to revoke e.g., self-signed certificates. Reverse proxying cached traffic through SSL is not possible and mandating to do it so if needed, requires the use of the server or the client or load balancers, to load the cache. A solid solution and to be correct, and appropriated solution in Microservices are the use of single-sign-on implementations. SAML and OpenID is perfect for Authentication and Authorization of someone's on a system but it's also great for service-to-service authentication as well.

As an alternative to the above and despite the operational challenges with management, Client Certificates can confirm identities over TLS. These can be used to establish a connection allowing the server to verify client authenticity thus providing a strong authentication assurance. It is important to keep in mind the constraints observable above.

Many examples can be found in the web also suggesting the use of third-party solutions like HMAC over HTTP. HMAC consist in hash-based messaging code to sign the request. Requests are hashed with a private key, and the resulting hash is sent along with the request. The server then uses its own copy of the private key and the request to re-create the hash. If it matches, it allows the request. API is core to Microservices as it enhances communication and promotes services and intercommunication between them. When developing apps or webapps nowadays, is fundamental to include services provided from other entities like Google or Facebook, and the way to identify which service is making a call, identifying, or limiting the range of action of a user recurring to or via API Keys. When it comes to using API keys to handle microservice-to-microservice approach, some systems use a single API key that is shared and use an approach similar to HMAC [33, 34].

A more common approach is to use a public and private key pair. Typically, keys can be managed centrally, just as people identities are managed centrally. The gateway model is very popular in this ground.

What makes this technology well received among developers is the fact that API keys are focused on ease of use as it aims to simplicity, robustness and strangeness when allied with powerful methods. Compared to handling a SAML- Security Assertion Markup language handshake, API key–based authentication is vastly more effective, much simpler, and more unambiguous. Finally, it's of the up most criticism to store this information in a well-protected environment recurring to separate security appliances to encrypt and decrypt data. SQL Server includes built-in support for encryption, so it is a reliable support to securely keep information. Alternatively, the use of vaults services can store, access, and retrieve keys.

3.3.3. Securing Data at Rest

Despite being a different kind of key, this one is also a fundamental key-factor, protect data that is not at use. Now that the environment is secure, that the network is secure and segregated in depth, and hopefully attackers cannot reach information, protecting laying-around-data is important.

Many breaches take place in protected environments, and information is attainable just because it is reachable at a given poorly secured point, as opposite areas of the system that well-guarded and consist in a costly attack target. Despite using defense in depth, its necessary to assure that data laying-around is contained in an encrypted way. Steps should be promoted to avoid using personal and not well-known encryption algorithms because this lacks assertiveness and the most important factor, massive criticism that enables the find and fix of errors [34]. Great and massive encryption algorithms have regular patches, many security teams pen-testing them and regular advisory tools. The use of examples AES-256, and encrypted data, should only be encrypted when it's visible e.g., backups, and decrypt it when data is being handled.

3.3.4. Defense in Depth

Microservices acts in layer or cells, so designing a system that can act like an onion thus providing layer of security is essential.

With DPI and Firewall, defense in depth is probably a last line of defense when the others are lingering or failing, so, the architecture of a given app should consider firewall over main layers of service controlling every port and service passing through it, recurring to deep packet inspection which is a combining technology of intrusion detection system and intrusion prevention systems with a stateful firewall. If the plan is to have multiple vendors serving hosted Microservices, multiple local and perimeter firewall and MPLS techniques should be applied.

3.3.5. Network and Subnet

The advantage of using Microservices is the security capability of network segregation application. Spread services across different networks or subnets and control them with firewall services or a simple IPTABLES, define their connections and active ports, functional IP addresses or MAC addresses, a fundamental task in 2020 and beyond.

3.3.6. Tracking

Tracking systems is a classic in security. Keeping track of what does what and done by whom is essential nowadays, and to do so recurring to logs (encrypted) provides knowledge of exploitation, use of the system, weak point and above all a record to study vulnerabilities. A solid alternative is the implementations o QoS solutions to control de environment and all aspects of intercommunication between peers. This type of systems and techniques enables the implementation of smart proceedings to recover from a problem, hence most provide alarm for activities.

4. Conclusion

In information security, may it be regarding software or infrastructure, there is a classic expression "Nothing is totally secure and protected". The objective of this research was to create a single point of contact regarding vulnerabilities and security requirements of the Microservices Architectures as a natural successor of Monolithic Architecture and methods. To do so, all aspects pointed by this work were tested in laboratory environment to assess effectiveness. Searching journals and articles available in databases, an uncomfortable reality emerges, the lack of direct overall research regarding information security in Microservices, and many results regarding application of the methodology avoid or lack the security point of view. Now, with this work, a research is made available regarding the most pressing points about securing this technological architecture and its use [36].

Most authors point out that there is no assurance that all will go well when observed the topics that this research presents. There are many things left aside due to the already immense possibilities and realities regarding this technology, and many of the present and included in the research, are the tip of the iceberg when concerning securing Microservices products and services. Studying and implementing security concepts and their constraints is nowadays an important critical point understanding people intention and how they work and interact with information systems. Human-related aspects will always be present and ignoring them, deploying application centered in excluding this reality will be, without no doubt, the path to failure [25, 32], the path to severe technological problems and legal problems hence, some countries have heavy juridical laws and legislation passed to combat information leakage, e.g.,

the GDPR set available in Europe. It's a scaring task to claim that Microservices are the future or that they are the top of the line, security wide speaking. It is even harder to say that they are not the future of application development. One thing is guarantee, despite innovative, scalable, wide available, Microservices are a severe headache for developers due to its architectural spreading nature and when in use, the mandatory security requirements to such possibility are complex to implement [26, 29, 36]. The goal is to prevent, detect, respond, and recover when necessary, might that be in a complex subject that has at its core asynchronous communication, isolation, automaticity, single responsibility, or in an exclusive state.

References

[1] Roberts, M., Udernani, R., Newman, S., Sharif, A., Baird, A., Buliani, S., Nagrani, V., Nair, A., Sun, Y., Nanda, S., Jaeger, T., Walker, D., Nadareishvili, I., Schneier, B., Dinh, K., Rajagopalan, R., Johnston, P., Pata, M., Pance, M., … Fowler, M. (2016). Rethinking Application Security With Microservices Architectures. In IEEE (Ed.), Software Architecture (WICSA), 2014 IEEE/IFIP Conference (Vol. 1, pp. 50–57). O'Reilly Media. https://doi.org/10.1109/CloudCom.2015.93

[2] L. Chen and M. A. Babar, "Towards an Evidence-Based Understanding of Emergence of Architecture through Continuous Refactoring in Agile Software Development," 2014 IEEE/IFIP Conference on Software Architecture, Sydney, NSW, 2014, pp. 195-204, doi: 10.1109/WICSA.2014.45.

[3] Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution. Proceedings of the Conference on The Future of Software Engineering - ICSE '00. doi:10.1145/336512.336534

[4] Puripunpinyo, H., & Samadzadeh, M. (2017). Effect of optimizing Java deployment artifacts on AWS Lambda. 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). doi:10.1109/infcomw.2017.8116416

[5] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. Present and Ulterior Software Engineering, 195-216. doi:10.1007/978-3-319-67425-4_12

[6] Puripunpinyo, H., & Samadzadeh, M. (2017). Effect of optimizing Java deployment artifacts on AWS Lambda. 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). doi:10.1109/infcomw.2017.8116416

[7] Fowler, S. (n.d.). Production-Ready Microservice. O'Reilly. Retrieved October 20, 2020, from https://www.oreilly.com/library/view/production-ready-microservices/9781491965962/ch04.html

[8] M. L. Abbott and M. T. Fisher, The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise. Addison-Wesley, 2015.

[9] Sun, Y., Nanda, S., & Jaeger, T. (2015). Security-as-a-Service for Microservices-Based Cloud Applications. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). doi:10.1109/cloudcom.2015.93

[10] Bonér, J. (2026, April 1). Bla bla microservices bla bla. Retrieved October 20, 2020, from http://jonasboner.com/resources/bla-bla-microservices-bla-bla.pdf

[11] I. Nadareishvili. Microservices shift complexity to where it belongs. Retrieved from https://www.oreilly.com/ideas/microservices-shift-complexity-to-where-it-belongs

[12] Cluley, G. Mystery surrounds iCloud hack as naked celebrity photos leak. Retrieved from https://www.intego.com/mac-security-blog/mystery-surrounds-icloud-hack-as-naked-celebrity-photos-leak/

[13] OWASP. Authentication cheat sheet. Retrieved from https://www.owasp.org/index.php/AuthenticationCheatSheet#AuthenticationGeneralGuidelines

[14] Team, N. (2019, July 17). What is the Server-Side Request Forgery Vulnerability & How to Prevent It? Retrieved October 20, 2020, from https://www.netsparker.com/blog/web-security/server-side-request-forgery-vulnerability-ssrf/

[15] Schneider, B. Schneider on security. Retrieved from https://www.schneier.com/blog/archives/2015/09/thesecurityri4.html

[16] NEWMAN, S. (2018). BUILDING MICROSERVICES: Designing fine-grained systems. Retrieved October 20, 2020, from https://www.amazon.com/Building-Microservices-Designing-Fine-Grained-Systems/dp/1491950358

[17] Hofmann, M. (2017, March 13). Microservices Best Practices for Java. Retrieved October 20, 2020, from https://books.apple.com/us/book/microservices-best-practices-for-java/id1194724476

[18] Daya, S. (2015). Microservices from theory to practice: Creating applications in IBM Bluemix using the microservices approach. Retrieved October 20, 2020, from https://www.amazon.com/Microservices-Theory-Practice-Creating-Applications/dp/0738440817

[19] Lewis, J. (2014). Episode 213: James Lewis on Microservices. Retrieved October 20, 2020, from https://www.se-radio.net/2014/10/episode-213-james-lewis-on-microservices/

[20] Nishanil. (2018). Microservices architecture. Retrieved October 20, 2020, from https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture

[21] Synopsys. (2015). The Heartbleed Bug. Retrieved October 20, 2020, from https://heartbleed.com/

[22] N. M. Coelho, B. Fonseca, and A. Castro. Paranoid operating system methodology for anonymous & secure web browsing, doctoral project. [Online]. Available: http://dx.doi.org/10.18803/capsi.v17.127-143

[23] N. M. Coelho, M. Peixoto and M. M. Cruz-Cunha. Prototype of a paranoid mobile operating system distribution. 2019 7th International Symposium on Digital Forensics and Security (ISDFS), Barcelos, Portugal, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757551.

[24] Sharif, A. (2017). Global Cybersecurity Leader. Retrieved October 20, 2020, from https://www.aporeto.com/accelerating-business-devops-and-microservices-part-ii-running-safer
[25] The Age of Microservices - Amazon ECS Service Discovery. (2018, June 28). Retrieved October 20, 2020, from https://www.trinimbus.com/blog/the-age-of-microservices-amazon-ecs-service-discovery/
[26] Sahni, V. (n.d.). Best Practices for Building a Microservice Architecture. Retrieved October 20, 2020, from https://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture
[27] Vučković, J. (2019). You Are Not Netflix. Microservices, 333-346. doi:10.1007/978-3-030-31646-4_13
[28] Hassan, S., Ali, N., & Bahsoon, R. (2017). Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. 2017 IEEE International Conference on Software Architecture (ICSA). doi:10.1109/icsa.2017.32
[29] Mateus-Coelho, N., Fonseca, B., & Castro, A. (1970, January 01). POSMASWEB: Paranoid Operating System Methodology for Anonymous and Secure Web Browsing. Retrieved October 20, 2020, from https://www.igi-global.com/chapter/posmasweb/261743
[30] Guaman, D., Yaguachi, L., Samanta, C. C., Danilo, J. H., & Soto, F. (2018). Performance evaluation in the migration process from a monolithic application to microservices. 2018 13th Iberian Conference on Information Systems and Technologies (CISTI). doi:10.23919/cisti.2018.8399148
[31] Carvalho, L., Garcia, A., Assuncao, W. K., Mello, R. D., & Lima, M. J. (2019). Analysis of the Criteria Adopted in Industry to Extract Microservices. 2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP). doi:10.1109/cesser-ip.2019.00012
[32] Yarygina, T., & Bagge, A. H. (2018). Overcoming Security Challenges in Microservice Architectures. 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). doi:10.1109/sose.2018.00011
[33] Fetzer, C. (2016). Building Critical Applications Using Microservices. IEEE Security & Privacy, 14(6), 86-89. doi:10.1109/msp.2016.129
[34] Torkura, K. A., Sukmana, M. I., Kayem, A. V., Cheng, F., & Meinel, C. (2018). A Cyber Risk Based Moving Target Defense Mechanism for Microservice Architectures. 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). doi:10.1109/bdcloud.2018.00137
[35] Sun, Y., Nanda, S., & Jaeger, T. (2015). Security-as-a-Service for Microservices-Based Cloud Applications. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). doi:10.1109/cloudcom.2015.93
[36] Vijaya, A., & Venkataraman, N. (2018). Modernizing Legacy Systems. International Journal of Web Portals, 10(2), 50-60. doi:10.4018/ijwp.2018070104
[37] Raj, V., & Ravichandra, S. (2018). Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services. 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). doi:10.1109/rteict42901.2018.9012140
[38] Goel, A. (2010). The Philosophy of Software Architecture. International Journal of Web Portals, 2(4), 28-39. doi:10.4018/jwp.2010100103
[39] Clohesy, B., Frye, A., & Redpath, R. (2009). Conceptual Business Service. International Journal of Web Portals, 1(3), 56-77. doi:10.4018/jwp.2009070104