Building Secure Microservice Architectures

Julian Hanhart

Table of Contents

Management Summary . . . 1
Objectives . . . 1
Approach . . . 2
Microservices: Definition . . . 2
    Example Architecture . . . 3
Perimeter Security and Defence in Depth . . . 4
    Perimeter Security . . . 4
    Defence in Depth . . . 6
    Decision Making Support . . . 8
Authentication & Authorisation . . . 9
    Service-to-Service Authentication . . . 9
    Authorization and its Propagation . . . 10
Secret Management . . . 11
Immutable Infrastructure . . . 11
Conclusion . . . 12
References . . . 13

The Microservice Architecture Pattern has recently grown in popularity. But since there are not that many authoritative texts on the security aspects of such architectures, this document shall attempt to compile and discuss the best practices on building secure Microservice Architectures.

Management Summary

"Microservices" as a architecture pattern are quickly gaining traction in software development circles. The idea behind the Microservice architecture pattern is to separate business logic into distinctive domains and to build individual, light-weight services focusing on specific business domains. These so called "Microservices" are only loosely coupled so that they can be developed and operated individually.

Most resources on Microservice architectures concentrate on technical and operational challenges. The implications on security of such an approach are often omitted from the discussion, unfortunately. Therefore, this document aims to summarize existing literature and provide best practices on the security aspects of Microservice architectures.

One major consideration when designing a Microservice based architecture should be whether it is sufficient to only secure the system's boundaries against threats or if the communication within the system and the individual components needs to be protected as well. If the system does not need to handle and store sensitive data and would not necessarily be a valuable target for potential attackers, securing the system perimeter might be sufficient. A layered approach to security would always be preferable, but depending on the intended purpose and the resource limitations involved, concentrating on the system boundaries may well be good enough.

However, if the system could be considered a high-value target for attackers, either because it processes and stores sensitive data or is mission-critical, the security measures in place should anticipate malevolent actors within the system's boundaries. Most of these measures would be very similar to those used in regular network environments, but in Microservice based systems, there is the possibility to identify the services handling sensitive data and to focus the security measures on these services specifically.

Other considerations should be how to authenticate and authorize users and clients to the system, as well as between individual Microservices. How to manage and distribute shared secrets like encryption keys and certificates in the system. And how to secure the automated build and deployment processes employed by such architectures.

Microservice architectures provide us with a lot of flexibility in how we want to solve problems, but this also means that we need to consider the security implications of these decisions and to incorporate appropriate security measures to mitigate them.

Objectives

Microservices have proven to be quite a popular architecture pattern for building elastic service landscapes. There are quite a lot of resources to help with the design and implementation of microservice architectures, but the security aspects of building such an architecture are often

omitted. Therefore, we want to discuss the best practices on how to build secure microservice architectures in this document.

With this document, we aim to provide a list of security relevant aspects that should be considered when designing a microservice based architecture and present possible solutions for them.

Approach

To identify the architectural aspects that are relevant to the security of microservice based systems, we first researched the best practices others have already proposed. One source that was mentioned frequently was the "Security" chapter of Sam Newman’s book "Building Microservices" [Newman]. Therefore, we decided to use his book and the keynote talk Mr. Newman held at the μCon microservice conference [MuCon] as our primary resources.

Although the basic concepts used in microservices architectures have been around for quite some time, microservices as an architecture pattern are a rather recent development [MF-MS]. Therefore, certain people might us the term "microservices" slightly differently than others. Because of that, we would like to present a definition and a sample architecture we would like to use for the purpose of this document first.

Second, we would like to discuss some differences in the basic approaches to security in microservice architectures. Especially, whether to solely rely on Perimeter Security or if and when a Defence in Depth approach might be more advisable.

At last, we want to look at some security challenges and their implications on microservice architectures specifically. Wherever possible, we shall also propose solutions (either concepts or products) to those challenges.

Microservices: Definition

A broadly accepted definition of what "Microservices" are was given by James Lewis and Martin Fowler [MF-MS]:

> In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
>
> — James Lewis & Martin Fowler, https://martinfowler.com

For our purpose, we would like to adapt this definition and define "Microservices" as follows:

- Microservices are independent services that implement business logic for a specific, limited domain
- Individual microservices manage their own data and restrict the data they handle to their specific business case
- Client requests to the system are delegated to the designated microservices by an API Gateway, which also aggregates the corresponding responses
- Microservices can be individually:
  - Developed and maintained
  - Tested and deployed
  - Scaled horizontally
- Testing and deployment of the microservices shall be done automatically

For our example architecture we would like to use an event-driven microservice architecture, meaning that:

- Synchronization of the data between the individual microservices in done by events that are propagated using a message queue

Example Architecture

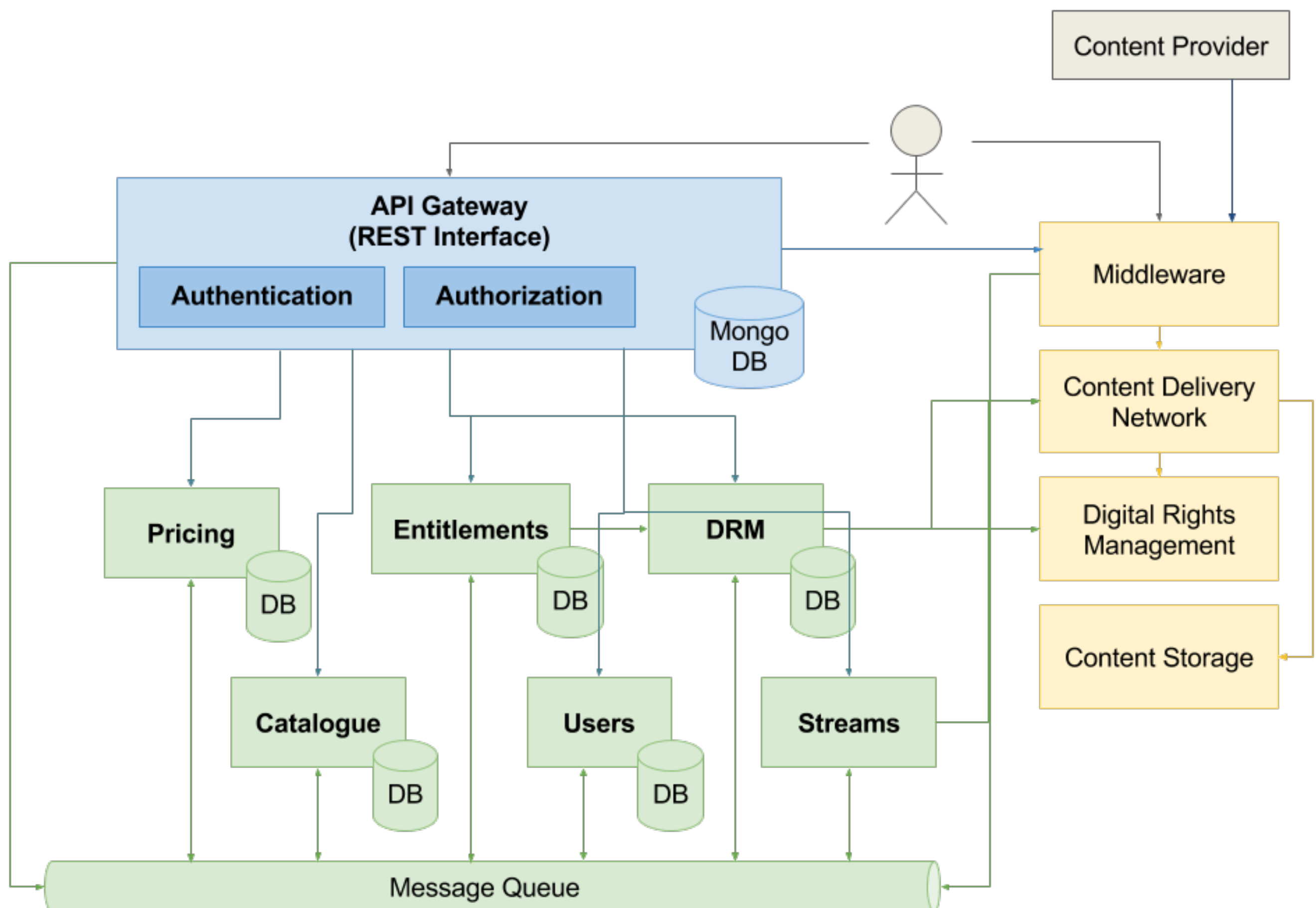

Figure 1. Example of a Event-based Microservice Architecture

We shall use a video streaming platform in the vein of *Netflix* as our example for such an event-

driven microservice architecture. Users can request video content, the prices for that content and what content they are already entitled to from the API Gateway, which retrieves the corresponding data from a catalogue service, pricing service and entitlement service respectively and merges the data to a client response. They can also get streaming URLs, for which the entitlements are retrieved and checked, the DRM system is prepared through a DRM connector service and the URLs are assembled by a stream service. There are also some clients (set-top boxes) that can access the middleware directly, circumventing the API Gateway. The services are populated with data by events the middleware system publishes to the message queue when content is received from an external content provider.

Perimeter Security and Defence in Depth

One of the first questions, that should be discussed when considering the security aspects of a microservice architecture, is what scope the security measures should have in the end. Is it sufficient to only secure the systems perimeter or do the individual microservices and the communication between them also need to be secured?

Perimeter Security

The first solution that comes to mind when thinking about how to secure a microservice-based system is probably to handle the security aspects at the network perimeter of the system. If only trustworthy actors can operate inside the network and the network is sufficiently secured against intruders, then one should be able to trust all actors inside the network's perimeter implicitly. This approach is of course not exclusive to microservice-based architectures and the same limitations as with other architecture types apply.

For one, there is of course the question of how the network perimeter is actually secured. The first thing one should ensure is that the network segmentation is properly implemented and that the firewalls to external networks are properly configured. Only clients from within the system's network should be able to directly access individual microservices. Only those components that must be accessible from outside the network should be exposed to other networks. In our example, the API Gateway should be the only part of the microservice architecture that is exposed to the internet. Outside access to third party systems like the middleware should be limited to specific networks or hosts.

Other than network segregation, one would also need to secure the outside-facing components against unauthorized access. There, one would need to consider which measures could be taken to secure these components:

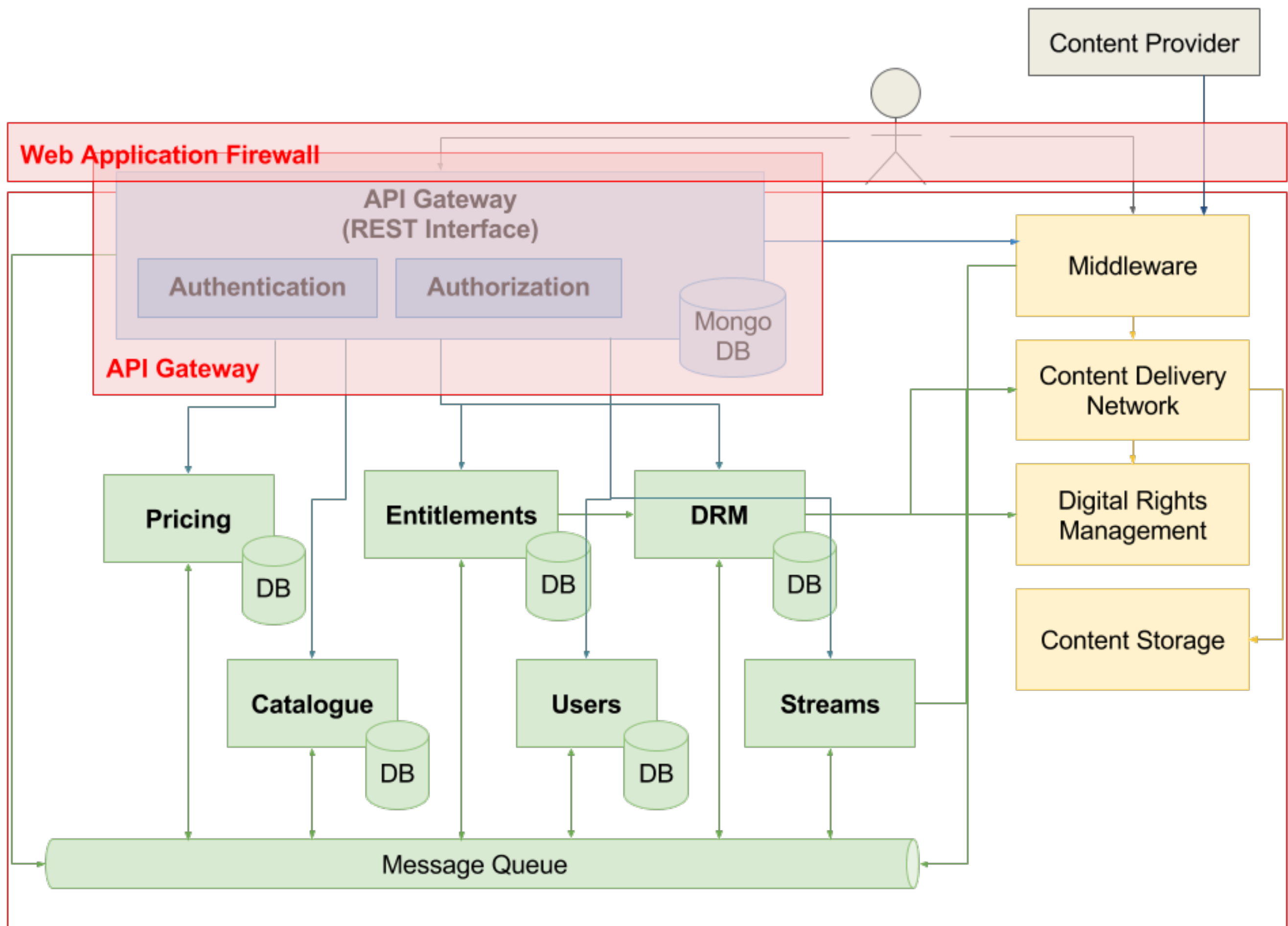

Figure 2. Perimeter Security Measures

Securing the API Gateway

One approach would be to harden the API Gateway. Depending on the underlying technology, this could be a question of adjusting the configuration of the existing application. It may also involve hardening the host operation system or the application platform the API Gateway runs on top of. In the end, the objective should be to minimize the attack surface presented to a potential attacker. But one problem with this approach is that only calls accessing the API Gateway would be covered. Third party systems would need to be secured separately.

Web Application Firewall

An additional measure that could be taken would be the introduction of a Web Application Firewall (WAF). A WAF could inspect all incoming requests for attempts to exploit common or specific web application vulnerabilities (such as SQL or Cross-Site Scripting injection attempts) and prevent suspicious requests from being sent to the API Gateway. The scope of the WAF could also be expanded to cover third party systems if they provide web services to external clients.

Third Party Systems

To secure third party systems one might well be dependant on the support of the system's vendor, if it can not be sufficiently covered with a WAF (or an XML firewall, if they expose SOAP interfaces). If the vendor does not provide enough security measures, one could adjust the network segmentation and firewall rules to be much more restrictive as compensation.

Application to the Example Architecture

Considering our example architecture, limiting security measures to the API Gateway would leave the system vulnerable at the middleware level. Attackers could attack either one of the middleware's external interfaces, either the one for direct client access or the one for the content providers. Introducing a WAF could secure these attack vectors and reduce the attack surface. This way, we could also work around vendor restrictions to securing the third party systems.

Defence in Depth

Even with the aforementioned measures implemented, relying solely on perimeter security remains problematic. If an attack manages to circumvent these measures, they will have unchecked access to the whole system. Since all actors within the network are considered trustworthy, an attacker within the perimeter will be trusted and allowed access to any component.

Therefore, a more deliberate approach should be applied if possible. To keep the perimeter security from being a single point of failure, a layered approach to security should be implemented. Multiple security measures should complement each other and components inside the network should be appropriately secured. This approach is called "Defence in Depth" [BS-DD] and should certainly be considered when designing the security measures of a microservice architecture.

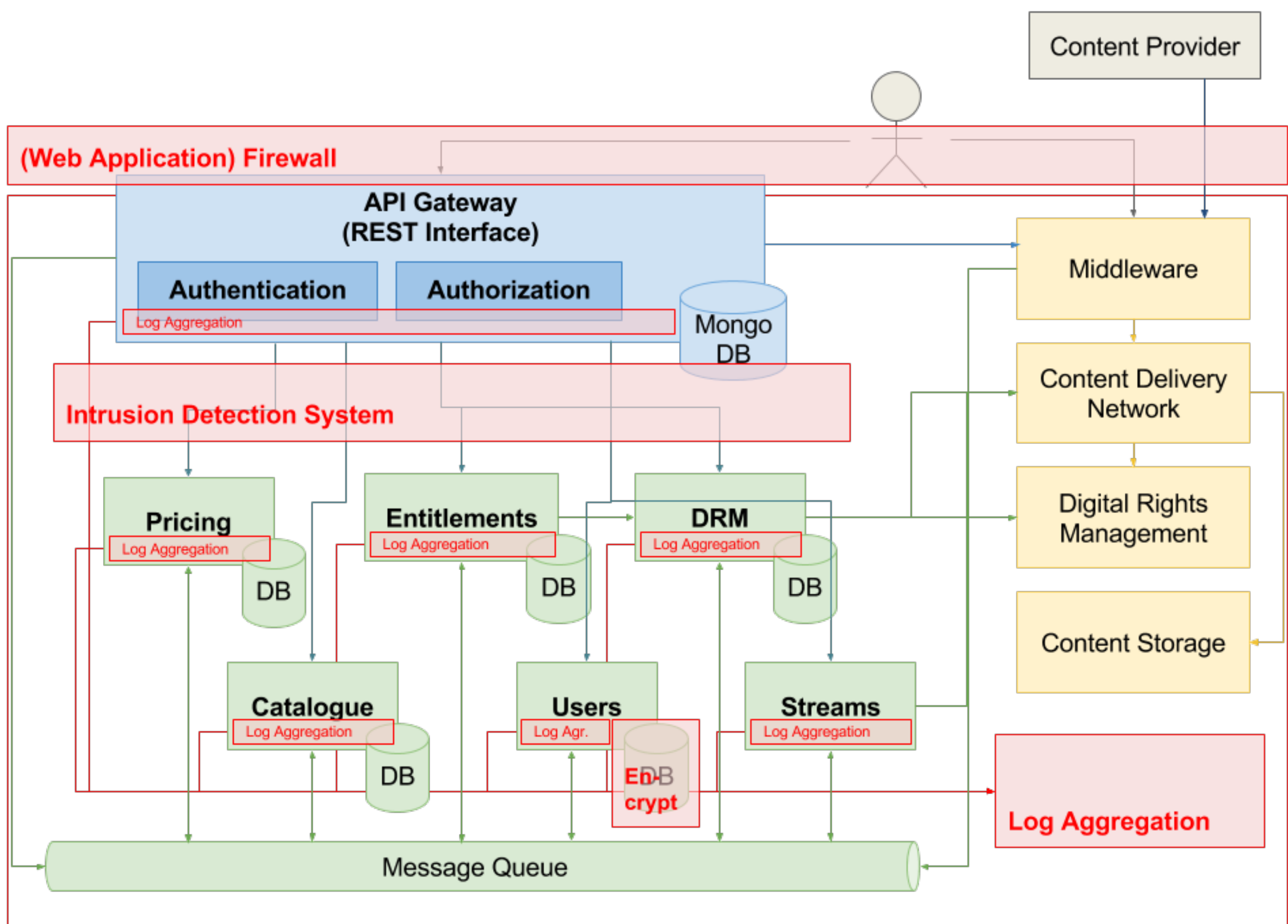

Figure 3. Security Measures to achieve Defence In Depth

Most security layers that should be considered in a microservice architecture are very similar to the ones used to secure regular network infrastructures. Therefore, we will not discuss them in too

much detail here and concentrate on how they can help us in a microservice environment.

Firewall / Web Application Firewall

The first layer of security when considering Defence in Depth should likely remain a WAF or at least a regular firewall. The ability to detect common attack patterns before they reach the API Gateway will still be very helpful and can aid in guarding against automated attack attempts and vulnerability scans. Firewalls are especially important for third party systems for which the integration of additional security layers can not be realized without the support of the vendor.

Transport Security

A very basic security measure on the microservice layer would also be to require the use of secure connections (HTTPS/SSL) for all communication between microservices. The performance penalty that the use of SSL once implied can mostly be disregarded nowadays [ITFY]. Therefore, there are few reasons not to implement Transport Layer Security, at the very least to secure the connection and to verify its server-side. One issue would be the need to generate certificates for all microservices (preferably in an automated way). Here, projects like *Let's Encrypt* could be evaluated to facilitate automatic certificate generation. To further improve the level of trust in the system, the use of client certificates could also be considered. But there, the certificate management is an even greater obstacle. The client certificates would need to be distributed and the revocation and reissue of certificates would need to be handled.

Intrusion Detection / Prevention System

An Intrusion Detection and/or Prevention System (IDS/IPS) could be used to monitor the network traffic between the API Gateway and the microservices, the microservices and the external systems and in-between the microservices themselves. Since the interfaces between those services should be very well defined, the definition of the rules that detect irregular behaviour might actually be easier than in a normal network environment. An IDS could also help to detect unwanted dependencies within the architecture if it were to be configured to send out alerts if sensitive services are accessed from components that do not have explicit permission to do so.

Log File Aggregation

The implementation of Log File Aggregation and Correlation should be considered for any microservice architecture with certain complexity, even without the security benefits in mind. Log Aggregation will help with understanding and retracing runtime behaviour and request paths over multiple services and is a very important tool to debug a microservice architecture. From a security standpoint it will also be indispensable to reconstruct what happened during a security incident and to follow an attackers actions after the fact. It should therefore be part of a Defence in Depth approach. It may also foil an attackers attempts to hide their traces by deleting or altering the log files of the services they compromised.

Encryption of sensitive Data (Data at Rest)

On the database layer, one should consider encrypting the data stores of services that keep sensitive data. Sensitive data is usually one of the main targets of attackers, either for exfiltration or to hold as ransom, and encrypting it while at rest would make such attacks much more difficult. It would not foil a ransomware attack encrypting the whole data store, were a strong backup policy would be much more helpful. But the encryption of sensitive data would also make these backups much less valuable as a target for attackers. In a microservice environment, the separation of data into service specific stores would enable us to be very selective in what we consider to encrypt and

lower the barrier of entry considerably.

Network Segregatation

With a modern, software-defined networking [ONF-SDN] based approach to network segregation, one could prevent unauthorized components from accessing microservices that manage sensitive data on a network basis. Only those components that are given permission explicitly would even be allowed to connect to the sensitive services. This approach would make it much more difficult for an attacker to access sensitive data, since they would have to identify and compromise a privileged component before they could even attempt to breach the intended target.

Application to the Example Architecture

In regard to our example architecture, enforcing HTTPS connections between all microservices and introducing an IDS to monitor those connections [1: Encrypting all connections would of course limit the usefulness of the IDS, unless we break up the SSL connection on the IDS for analysis] in addition to the existing WAF, would enable us to enhance our defencive capabilities significantly without too much effort. Hopefully, we would already employ Log Aggregation, otherwise it would be the right time to introduce it. If we only consider the user data as sensitive, we could also encrypt the database of the user service and even consider limiting network access to the user service to components that actually require access to the user data.

Decision Making Support

When deciding what security measures should be implemented for any given system, considering the resources of the implementing organization and team, there are a few tools and processes that can support the decision making process.

Identifying Sensitive Data

To identify the components that should be considered high-value targets and therefore need additional protection, one should first identify what information handled or stored by the system would be sensitive to the owner and/or could be valuable to an attacker. Microservices that store or process sensitive data should warrant additional protection measures and should be prioritized when implementing security measures.

Threat Modeling

Thread Modeling is the process of decomposing an application into its components, determining and ranking threats to the application and its components and determining countermeasures and mitigation measures for those threats [OW-TM]. This process can help with identifying weaknesses and low-hanging fruits in the application's architecture, but also with finding high-value attack vectors within the application. There are different approaches to Threat Modeling and accompanying tools for them, including these:

- Attack Trees as proposed by Bruce Schneier [BS-AT]
- Microsoft: Thread Modeling [MS-TM] (also part of their Security Development Lifecycle [MS-SDL])

Application to the Example Architecture

In the case of our example architecture, we decided to only consider the user data as sensitive. All other data would not be very useful to an attacker or is already encrypted by the third party systems (i.e. video assets). Therefore, we were able to decide that only the user service requires special protection measures.

Authentication & Authorisation

One other major aspect of any security setup is the question of how to solve the authentication and authorization of users in the system.

As with the question of whether to solely secure the network's perimeter or whether to use a Defence in Depth approach, we can also decide to simply authenticate users at the API Gateway and to let the gateway handle the access control to the individual microservices and their functions. However, this approach has similar drawbacks as Perimeter Security. Once an attacker has breached the network, they would be free to do as they please, since the microservices would not check the authorization of requests from within the network. Such an approach would also require all other external interfaces to have their own implement of authorization and authentication, making maintenance and debugging much more difficult.

To exclusively rely on the API Gateway for authentication and authorization can be a viable approach if the architecture's complexity is limited and no sensitive data is handled by the system (or if it can be individually secured). But one should at least consider whether to forward authentication tokens to the microservice, so that they could implement more advanced verification measures individually if needed.

Service-to-Service Authentication

For a more fine-grained solution to authentication, one should also consider to authenticating the individual microservices to each other. Since most of the implementation approaches to service-to-service authentication require some sort of shared secret, we will discuss Secret Management in a separate paragraph. These authentication methods may also be viable solutions to authenticate clients to the API Gateway and a common approach to client and service-to-service authentication might be advantageous.

HTTP Basic Authentication

The most basic approach would certainly be to submit a username and password pair over HTTP Basic Authentication. But this would also make the use of Transport Layer Security for service-to-service communication strongly advisable. Otherwise, the authentication data is sent over an open channel and could be visible to an attacker.

Single Sign-On

Another common approach would be to integrate the microservices into an (existing) Single-Sign On (SSO) solution. Incorporation SSO into the service-to-service communication would also help with the authorization of requests. There are a host of open and proprietary SSO solutions and we only want to introduce a few widely used ones:

- Security Assertion Markup Language (SAML): An SOAP-based standard used by many products, but can be fairly complex to implement [Newman]
- OpenID Connect: An open, REST-based solution, build on top of OAuth 2.0
- Directory Services: Existing directory services like Active Directory or OpenLDAP could also be utilized

Client Certificates

As already discusses in Transport Security, the use of client certificates for service-to-service connections would enable a very strong authentication, but would entail a lot of effort to get the certificate management right.

Hash-based Message Authentication Codes

To ensure requests were sent by a known service and were not altered during transport, one could implement Hash-based Message Authentication Codes (HMAC) [IT-HMAC]. Were the messages sent from one service to another and a secret key known to both services would be run through a cryptographic hashing function and the resulting hash would accompany the message (i.e. as a header). The receiving service can then reproduce the process with the received message and check if the result matches the given HMAC, thus verifying both the integrity and authentication of the message.

API Keys

Another rather simple approach to authenticating services to each others would be the use of API keys. There are various implementation approaches to API keys. They can be implemented as simple shared secrets, as keys for a HMAC as discussed above or as public and private key pairs.

Authorization and its Propagation

When discussing authorization within a microservice architecture, we should both consider service-to-service authorization and the propagation of the external clients authorization principal. Usually, the same tools can be used to solve both problems.

One potential stumbling block most approaches have in common is the Confused Deputy Problem [Hardy]. Where an actor tricks a service to make requests to another service they should not have access to. The second service would only see that the request is coming from the first service and assume it to be authorized to execute the request. Therefore, it should be investigated if it is possible to pass the original caller’s authorization principal down to downstream services if such a scenario is viable.

Single Sign-On Tokens

Most Single Sign-On solutions allow the propagation of roles. Therefore, the integration of microservices into the SSO solution discussed above could be a good approach to authorization within a microservice architecture.

OAuth

The OAuth 2.0 protocol was specifically designed as an open standard to allow secure authorization in an HTTP environment and is therefore well suited for REST-based microservices. It is widely used and there is very good support in variety of frameworks (i.e. Spring Security) because of that.

JSON Web Token

JSON Web Token (JWT) is a JSON-based open standard for access tokens. The tokens are signed by the issuing server's key to verify their legitimacy. Tooling support might not be as widespread as with OAuth, but due to their compact design, they could be a good basis for a custom authentication implementation.

Secret Management

A common problem to a lot of the presented solutions is secret management. In a distributed environment where lots of instances and services need to share secrets, such as keys and certificates, with other services, the question arises where those secrets should be stored. Hard-coding keys or storing them in the application configuration would only scale up to a certain point and would not be very friendly to an automated build process. Therefore, some kind of centralize secret management is probably advisable in a microservice environment.

Public Key Infrastructure

If Transport Layer Security should be used for the service-to-service and especially if the use of client certificates is wanted, some form of a Public Key Infrastructure (PKI) would be very helpful. Manually generating, distributing, revoking and reissuing certificates will probably get burdensome very soon.

Secret Management Tools

There are some tools and service specializing in secret management in a distributed environment.

- Amazon's AWS Key Management Service (KMS): Provides key management as a service in Amazon's AWS cloud environment
- HashiCorp's Vault: A standalone service for the management of secrets (tokens, passwords, certificates, API keys etc), with a variety of secret and authentication backends

Immutable Infrastructure

Since one of the main advantages of microservice architectures is the ability to automatically build and deploy individual services, the concepts of Immutable Infrastructure and Infrastructure As Code are also gaining traction [MF-IC]. The ability to describe the configuration, setup and dependencies of infrastructure components like servers, virtual machines, containers and networks in text form and to build the components from that description greatly enhances the automated deployment and orchestration processes for microservices. Without the need for manual involvement, setting up infrastructure components becomes repeatable and less prone to human error. But since the idea is to rebuild components from source instead of adapting and patching the rolled out components, there are some security aspects that need to be kept in mind when employing an Immutable Infrastructure approach.

Vulnerabiliy Management

If we rely on base images that were provided by third parties, we need to be aware that they can contain vulnerabilities. In fact, a 2015 study found that over 30% of images published on Docker Hub contained components with critical vulnerabilities [BY-DV]. Therefore, we should be very

careful when selecting images to base our own containers on and verify their update frequency (a tool that could help detecting vulnerable images is CoreOS' Clair). Furthermore, once they are built, containers should no be left deployed indefinitely. One should consider to automatically replace containers after a certain runtime with new versions built from the latest base image. Of course, vulnerabilities are not limited to the containers. The host operation system should also be updated automatically (or rebuild if something like Vagrant is used to build virtual machines from source).

Secret Management

As already discussed in Secret Management, secrets should not be hard-coded or configured by hand. Distribution during the build process could be a viable option, but should be well though out. The usage of a secret management service should at least be considered.

Tracking of Deployed Services

To monitor what components are rolled out throughout the system and to facilitate the automatic replacement of outdated ones, it should be tracked which services are deployed. To profit from the performance benefits of microservices and to achieve an elastic system that can scale on demand, the usage of tools like Service Registries, Service Discovery systems and Client-side Load Balancing is strongly advisable anyway [MI-SD]. But they can also help with monitoring the versions of base images and the age of containers, as well as with the seamless replacement of outdated containers.

Conclusion

When contemplating the security aspects of a microservice based architecture, one should focus on identifying the sensitive and vulnerable parts of the system and to implement appropriate security measures accordingly. A major advantage of the microservice pattern, that also enables us to use a targeted approach to securing the system, is that the separation into individual services is done by business domain. Therefore, sensitive data should be contained to the services that need to handle this information and restrictive security measures can be applied to those services while a more light-weight approach can be chosen for less sensitive services.

While relying exclusively on Perimeter Security can be an option for systems that do not need to handle sensitive information, Defense in Depth should be the objective for most architectures. Avoiding a single point of failure and layering security measures is always advisable.

If fine-grained access control should be achieved, designing sensible authentication and authorization mechanisms is paramount. There are a variety of authentication and authorization options for most common frameworks that should be carefully evaluated. Implementing a custom solution should therefore be the last resort.

Since modern orchestration tools rely predominantly on container solutions, potential implications of containerization to the system’s security should be considered while designing the architecture. Containers should be scanned for vulnerabilities and monitored once deployed.

References

Books & Talks

- [Newman] Sam Newman. *Building Microservices*. O'Reilly. ISBN 978-1-491-95035-7.
- [MuCon] Sam Newman. *Keynote: Security and Microservices* (https://skillsmatter.com/skillscasts/8413-keynote-security-and-microservices), µCon 2016. 01.11.2016.

Articles & Documents

- [MF-MS] James Lewis, Martin Fowler. *Microservices* (https://martinfowler.com/articles/microservices.html), Martin Fowler. 25.03.2014.
- [BS-DD] Bruce Schneier. *Security in the Cloud* (https://www.schneier.com/blog/archives/2006/02/security_in_the.html), Schneier on Security. 15.02.2006.
- [ONF-SDN] ONF. *Software-Defined Networking* (https://www.opennetworking.org/sdn-resources/sdn-definition), Open Networking Foundation. March 2017.
- [ITFY] Ilya Grigorik. *TLS has exactly one performance problem: it is not used widely enough.* (https://istlsfastyet.com/), Is TLS Fast Yet?. March 2017.
- [OW-TM] OWASP. *Application Threat Modeling* (https://www.owasp.org/index.php/Application_Threat_Modeling), Open Web Application Security Project. March 2017.
- [BS-AT] Bruce Schneier. *Attack Trees* (https://www.schneier.com/academic/archives/1999/12/attack_trees.html), Dr. Dobb's Journal. December 1999.
- [MS-SDL] Microsoft Security Development Lifecycle. *SDL Process: Design* (http://www.microsoft.com/en-us/SDL/process/design.aspx?Accordionitem1=True), Microsoft. March 2017.
- [MS-TM] J.D. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla, A. Murukan. *Improving Web Application Security: Threats and Countermeasures* (https://msdn.microsoft.com/en-us/library/ff648644.aspx), Microsoft. June 2003.
- [IT-HMAC] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication* (http://www.ietf.org/rfc/rfc2104.txt), Internet Engineering Task Force. February 1997
- [Hardy] Norm Hardy. *The Confused Deputy (or why capabilities might have been invented)* (http://www.cap-lore.com/CapTheory/ConfusedDeputy.html), Operating Systems Reviews. 1988.
- [MF-IC] Martin Fowler. *Infrastructure As Code* (https://martinfowler.com/bliki/InfrastructureAsCode.html), Martin Fowler. 01.03.2016.
- [BY-DV] J. Gummaraju, T. Desikan, Y. Turner. *Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities* (https://banyanops.com/blog/analyzing-docker-hub/), Banyan. 29.05.20015.
- [MI-SD] Chris Richardson. *Pattern: Client-side service discovery* (http://microservices.io/patterns/client-side-discovery.html), Microservices.io. March 2017.