This part of the thesis will present requirements analysis for a microservices-based architecture that relies on content trust to verify the behavior of its different components. The discussion will be split into two parts:

1. Microservices and micro frontends
2. Content trust between microservices

The reason for such division is that, microservices and micro frontends are architecture. Hence they have their own set of rules and requirements when it comes to building an application based on this new architecture.

On the other hand, content trust is one of the security concerns, and with just like any application, security concerns will be addressed throughout the development of the proposed Blog. Yet the focus here is on the trust among different microservices, especially those coming from different sources/developers. Thus, this chapter will allocate a subsection in the requirements analysis to discuss the requirements analysis of the content trust of the application.

**Content trust between microservices**

This section will discuss the requirements for a content trust system that will be implemented to help microservices verify the behavior of other microservices. The following pages will try to give more details about how and what a system of content trust should be.

Microservices themselves need to have clear rules about how to communicate with other microservices especially ones coming from different developers. This communication must allow microservices to make a judgment of whether to trust the other microservices or not.

On the other sides, there are no requirements for the user, the one who is going to use the overall application. Because, content trust is something to be handled between microservices, or a protocol and set of rules to be read and understood by the microservices themselves. And so the user has no role in how the content trust should be and what it should include.

**Microservices requirements**

A system based on microservices architecture consists of different small pieces of code. Each small piece is an application that can be deployed independently. It can also be updated and modified while no need for any modifications of the other small apps because such an update took place for one or more microservices. Such architecture, in theory, makes the system loosely coupled. Thus different system parts and components are easy to change, update, modify or even replace. As long as the interface of the new introduced microservices respects the old interface or keeps using the same communications protocols, then the system will continue to function.

Just like any other system, applications based on the microservices architecture are vulnerable to security threats. This vulnerability comes from the traditional security challenges that face any other system, but also from the nature of a microservices-based application.

When developing microservice-based applications, security concerns should be addressed. Each microservice should be able to verify that the other microservice that’s trying to connect with it is what it claims it is. Then once this established each microservice should be able to move on and do other checks.

The reason for this verification is because microservices are written in a way makes them easily deployed and modified. The whole idea of the microservices architecture is built on creating a very flexible system. Such system will continue to run when one or more of its microservices are not functioning. It will also continue to run when developers decide to make changes. Such changes could be that one or more microservices need update, some alternations or even a complete replacement.

Here at this point, when introducing new microservices to the system, the older microservices should be able to have a good security system. Such system will allow older microservices to check the identity as well as the behavior of the newly added microservices.

This mentioned scenario is the normal case. It is ok and even expected that developers of a system will make changes once the system is deployed. Especially in the testing phases when developers are running beta versions of their newly developed application. In this stage developers could continue to make changes. Hence microservices are not supposed to exchange data blindly with other microservices.

The fear of hacking into the system is another situation that comes also from the nature of microservices. Such hacking could happen by replacing one microservice with another. The newly added microservice could successfully have a similar interface, allowing the other microservices to exchange data with it. In such case if unwanted party get to access and replace one or more microservices, they could also be able to access sensitive information exchanged with the new implanted microservices. Where those added microservices could send any information exchanged with them outside the application and thus putting the security of the clients at risk.

Content trust has more dynamic nature than other types of trust such as identity trust. While when performing an identity check the outcome could be one of two. Either the identity has been proven, or the identity failed to prove itself. But with content trust several characteristics, measures and aspects should be taken into account to come to a decision of trust or distrust.

Those measures differ in how hard it is to obtain the needed information. It all depends on the kind of information is being retrieved. In order for microservices to be able to check the content trust of one another each microservice should be able check the following:

* The identity of a microservice
* The type of service a microservice provide
* How sensitive the service provided by the microservice
* How many other microservices interact and use the services of a certain microservice
* The evaluation of content trust of a certain microservice by other microservices
* Age of the microservice in concern in relation to other microservices
* The relation of the microservice in concern with other trusted microservices
* The interface and the nature of the connection with the microservice

**The identity of a microservice**

When an application starts using third-party microservices then there should be a need for verifying the identity of microservices.

Verifications of other microservices

A system must be in place to help microservices verify each other and make sure at first that each microservice is what it claims. Verification is not an essential step for content trust but if verification was available. Then before proceeding to the contextual check, each microservice will have the opportunity to declare itself to others.

If a verification mechanism is implemented as a part of the content trust system, then the evaluation process will be handled in two steps. The system at first must allow for microservices to verify themselves to other microservices. If the verification process was successful the microservices move on to the next step. In case the verification step failed then the involved microservices can’t move further in the process and any planned communication will not take place. Moreover such scenario could actually lead to further steps. The system could include a notification mechanism. Those notification will signal the ID of the involved microservices and which one failed in authorizing itself. Hence developers can take notes of what happened and make sure that all the microservices of the system are secure.

Microservices can always proceed to the content trust check without going through any identity verifications. But having an identity verification system implemented will help to make the system more secure. Hence a discussion of the different possible verification methods is presented.

Verifications of microservices can be implemented in different ways, the following methods will be discussed:

* One-to-one verification
* Centralized verification

One-to-one verification

When one-to-one verification takes place, each microservice will have the opportunity to exchange identity information with another microservice. This exchange of identity information should happen only once. The very first time two microservices are trying to connect with each other.

When a microservice is sending a request to another microservice, if both microservices never connected before then a protocol of identification should be followed. Such protocol could allow each microservice to check the identity of the other microservice and make sure that it is exchanging data with an authorized microservice.

When developers are implementing different microservice, a discovery record could be developed to keep track of the added microservices to the system. This way each microservice can search this list and know what microservices are available to connect with. Inside this index of microservices, each microservice can have its identifying information. Hence when two microservices are about to connect for the first time. Each one will send a copy of its public identity information to the other. The microservice on the receiving side, will check the received information against the ones available in the record. Information in the record is put by the developers themselves and is secured against hacking or any attacks.

Developers could also store the hash of identity data for each microservice in the record. Once two microservices wants to connect and exchange data. They will send each other their identity information. Each microservice will receive identity information and then calculate the hash for the received data. Once the hash is calculated it can be compared against the hash that was generated by the developers. If the hash values match then the microservice is what it claims to be. Because each microservice will not know how the hash was calculated. If someone was trying to put unauthorized microservice in the system, they will not be able have a microservice with identity information that will result in the same hash data as the ones available in the index of microservices.

Of course such approach doesn’t guarantee security of the system in all cases. And extra attention must be given to the index of microservices to keep safe from any attacks.

Centralized verification

Such approach can take many forms. One way of doing a centralized verification is by having a central microservice responsible for the verifications of different microservices. This approach is simple, the authentication microservice will have a table that contains certain secure information about each microservice. Once a microservice wants to connect with another microservice, it should first contact the authorization microservice.

The idea is similar to a log in process. Each microservice will send its secure data to the central microservice. Once received by the central microservice, those data will be checked. The authorization microservice will check this data against the data that it has already stored b the developer in its database. If there’s a match then, the requesting microservice will be informed. The next step would be that the requesting microservice will inform the central microservice of which microservice it wants to connect with. Once this information is received, the central microservice will send a token for both microservices that want to connect with each other.

Once the token is received both microservices can exchange this token and start their communication. This process is only needed the first time any two microservices want to connect together. Next time each microservice can start exchanging information directly with the other microservice.

As discussed above, verification can be skipped before content trust check is performed, but having verification of identity as a first step will help microservices to communicate with each other after establishing a basic security step.

This verification could actually play a role in the content trust and the decision of trusting the behavior of one microservice or not. In this case if a verification system is implemented and the verification step is passed successfully. Then an extra value could be added to the microservices which successfully authorize themselves to other microservices. Of course such consideration depends on the final system that’s implemented for the content trust.

**The type of service a microservice provide**

Microservices should be able to check the service that’s provided by any microservices trying to connect with them for the first time. The more sensitive the provided service is the more important such check. There can be different ways that allow one microservice to check the service of the requesting microservice. For example, there could be a record that lists the services provided by each microservice with the identity of each microservice. Once a request has been made, the requested microservice can check the request information of the requestor microservice against this record. It will then be able to verify that the service of the requestor microservice is what it claims to be.

Another method could be done by running a simple test with the requestor microservice using dummy data to first check that it’ll run as it claims to be. This might need some form of artificial intelligence. It will enable each microservice to truly verify if the requestor behavior is as it is supposed to be or not.

**How sensitive the service provided by the microservice**

The services provided by the different microservice will vary in nature, some will offer simple service, while others will offer sensitive services. For example, some services could offer routing to help the user navigate from one page to another. On the other hand, another microservice will offer a log in service to the user. Such service has a higher sensitivity than the other.

**How many other microservices interact and use the services of a certain microservice**

Each microservice should be able to know if there are other microservices that used the service of the requestor microservice or replied positively to its requests. Also each microservice should be able to know if the requests of the requestor microservices have been denied or not. And which microservices denied the requests of the requestor microservice. In case of failing or negative replies to the requestor microservice, then it’ll also be helpful to know which microservices responded negatively to the requestor. And what kind of relationship exist between the requested microservice and those other microservices that responded negatively to the requestor.

**The evaluation of content trust of a certain microservice by other microservices**

When a microservice receives a request from another microservice, it should be able to see what evaluation the requestor microservice received from other microservices. This evaluation will be the degree of trust each microservice gave to the requestor. By seeing what other microservice thinks of the behavior of the requestor microservice it’ll then help the requested microservice to increase its own positive or negative evaluation of the requestor.

**Age of the microservice in concern in relation to other microservices**

Microservice in the system might have different operation age, this difference come from the nature of the architecture of microservices itself. Microservices are added gradually to the system. Hence some will be added in the early stages while other will be added at a later stages. Moreover, some microservices will be replaced by new microservices. And some new microservices will also be added to fulfill new requirements or fix a newly discovered bug. When making requests to other microservices, the requested microservice should be able to check the age of the requestor. The older the requestor the ore trust the requested microservice could have about the requestor. Of course, there should be a very trusted method of checking the age of each microservice by the other microservices.

**The relation of the microservice in concern with other trusted microservices**

The relationships between microservices could play an important role when deciding which microservices to trust or not. A transitive system could be in place to help transit the trust of a microservice from one to another. For example, three microservices A, B and C. the content trust relationship between them is as follows: A trust B, and B trusts C. the two microservices A and C have never interacted before. When C makes a first request to A. the microservice A should be able to check the microservices that it already trusts and their trust decision regarding C. Since A already trusts B, and B already trusts C, then A could come to a positive conclusion and increase its evaluation of trust to C based on its trust to B which in turn trusts C.

**The interface and the nature of the connection with the microservice**

Microservices have interfaces that help them interact with each other and exchange information, this can be in a form of an API calls over a protocol. Such interface could be helpful when deciding to trust a microservice or not. When microservices have similar methods for API calls and use the same data type and containers for exchanging data such as JSON or XML then this could be considered a positive point when deciding to trust the behavior or a microservice or not. The reason for this is because if the microservice was installed by the same developers then it is more likely that they will try to unite the way microservices interact with each other inside their system.

When microservices have a system that helps them in identifying other microservices and also helps them in making a decision about the behavior of other microservices. Then the system will exhibit an intelligent behavior that will help in detecting and eliminating security threats. In order to obtain a decision about the behavior of other microservices, each microservice must rely on different set of factors and guidelines. These factors and norms will help microservices come to a decision of whether they should trust another microservice making a request and trying to exchange information with them.

As can be seen, content trust requirements of microservices are a combination or policy-based trust and reputation-based trust. Microservices have to make a decision of trust the content and behavior or other microservices based on a combination trust systems.

**Content trust requirements of others**

The other involved parties in a microservices-based applications are developers and users. Developers are the person or the group of people who are creating the application. From their point of view when trying to handle content trust between microservices, there can be two cases:

* All microservices are developed in-house
* Some microservices are developed by a third-part

When having all the microservices as an internal product, something developed by the same company then trusting the behavior of different microservice is not as important. The reason for this is that when developers develop a microservice they’ll be able to trust its behavior directly. They will be sure that no hidden intentions are implemented or any harmful behavior intentionally in place. Furthermore, some concerns should be taken into consideration if in the future some microservices will be introduced from a third-party.

On the other hand when some microservices are developed by a third-party, then developers must make sure that microservices of both sides will be able to communicate with each other to verify their behavior and build their trust. Developers should make sure that any third-party microservices should be able to provide the requested information. This information such as identity verification, age of operation, the type of service provided, and so on. Those information will help microservices to make a decision of whether they should trust the behavior of a certain microservice or not. Failing to do so while having a content trust system implemented could create many problems.

On the other hand users of the system don’t have any content trust requirements. The reason for this is because content trust is something related to the interaction between microservices themselves. It is all internal behavior.

**Literature review**

This part of the thesis tries to present a literature review of trust and the different mechanisms used to help secure the behavior of a software. The interaction between the user and the application as well as the interaction among different applications.

First, a definition of trust is illustrated, to give the reader a broader understanding of trust and its use in the literature. The next step is presenting the different techniques of trust as used by researchers and software developers.

Definition of trust.

The word trust has been a subject of many studies, many researchers tried to form a definition of trust and what it means. The reason for this is because trust plays an important part of people’s life and is involved in many scientific fields such as philosophy, psychology, economy and recently in computer science. In his famous PhD thesis Marsh [9] mentions that many efforts have been spent trying to discuss trust and putting a definition to it, especially in the second half of the last century. His research was an attempt to create a model that can offer a mathematical way to measure trust.

Dr. Harrison et al [6] concludes that there’s no one agreed upon definition of trust, “little consensus has formed on what trust means”. In his research, he agrees with Marsh [9] that many discussions have been written about trust. Furthermore he adds that different definition of trust are used in the literature,” significantly diverse definitions of trust continue to be used in the interdisciplinary research literature”. In his attempts to giving a definition for the word trust, Dr. Harrison et al [6] first tries to show the importance of trust. He mentions that “trust leads to cooperation”. He first, presents what some other researches have provided as a definition or an understanding of trust. Then Dr. Harrison et al [6] distinguish between six types of trust “we define six related types of trust”. Those six types are also used by other researchers when attempting to define trust [x]

The first type of trust defined by Dr. Harrison et al [6] is the Trusting Intention. This type of trust means that one is able to depend on others. Dr. Harrison et al [6] argue that this type of trust is a “situation-specific”. On the contrary to this definition Gabarro, J. J. [10] thinks that this type of trust is not a situation specific. Dr. Harrison et al [6] present the difference in such understanding as “This makes the other person the object of trust, rather than the person in one situation”

The second type of trust is the “trusting behavior” according to Dr. Harrison et al [6]. The definition for trusting behavior is also given by Lewis & Weigert [11] “is the extent to which one person voluntarily depends on another person in a specific situation with a feeling of relative security, even though negative consequences are possible.“ Dr. Harrison et al [6] go further and tries to decompose trusting behavior into different subconstruct forms. Namely are: “cooperation, information sharing, informal agreements, decreasing controls, accepting influence, granting autonomy, and transacting business. “ Anna Macko & Marcin Malawski [12] studied the trusting behavior in their work, named “Belief in others’ trustworthiness and trusting behavior”. They show that many factors plays a role in the trusting behavior, and it is not just about “maximizing the profit”.

Dr. Harrison et al [6] puts “trusting Beliefs” as the third type of trust. The given explanation is “the extent to which one believes (and feels confident in believing) that the other person is trustworthy in the situation” other researches have also studied trusting beliefs, for example Dr. McKnighta et al [13] explain that trusting beliefs are “perceptions of the competence, benevolence, and integrity”. In their explanation they give an example of a vendor-consumer relationship. “Willingness to depend (that is, a decision to make oneself vulnerable to the vendor)”.

Moving on to the fourth type of trust that was distinguished by Dr. Harrison et al [6], this type is called system trust. It is explained as “means the extent to which one believes that proper impersonal structures are in place to enable one to anticipate a successful future endeavor”. Robin Pennington et al [14] give an example of system trust, the use ecommerce system as such example. They concluded that system trust has an impact on the intentions of customers to decide whether to buy or not, “system trust plays an important role in the nomological network by directly affecting trust in vendors and indirectly affecting attitudes and intentions to purchase.”

Dr. Harrison et al [6] provide dispositional trust as the fifth type of trust, explaining “, if one believes that others are generally trustworthy (Belief-in-People), then one will have Trusting Beliefs (which in turn lead to Trusting Intention).” Dispositional trust is also noted by Sharon G. [15], in his research about situational uncertainty, he concludes that dispositional trust can foresee trusting phenomenon.

Lastly, the sixth type of trust according to Dr. Harrison et al [6] is the situational decision to trust. Dr. Harrison et al [6] explain it as “the extent to which one intends to depend on a non-specific other party in a given situation. “. Although it is recognized as a different type of trust, but it doesn’t exhibit much difference from the first type of trust which is Trusting Intention.

Dean Leith [8] in his paper about the concept of trust defines trust as “Trust as a ‘leap of faith’ or willingness to be vulnerable”. He argues that trust is a tool learnt at as earl age as infancy. Where people use this tool to approach uncertain situations “trust is learned in infancy and enables the individual to deal with the unknowable in the social context”. In his explanation for the term ‘leap of faith’ he presents this term as an important part of the trust where it “involves the trustor experiencing a lack of expertise in a particular area of their life and acknowledging that the expertise they require to address this lack is held by another individual or system. They either consciously“

On the other hand dean Leith [8] presents trust from a different view, he looks at the trust as a “social capital” he argues about the role trust plays for individuals in society and hence the role each individual plays in the society.

Lastly he presents trust as a part in the “power-knowledge” theory where knowledge leads to power and trust is an important component to acquire knowledge.

As can be seen that there’s no one definition of trust in the literature and many researchers have come to conclude different means and concepts of trust. Some have given examples from the real world such as Robin Pennington et al [14] where he talks about trusting a system. The same concept of trust is agreed up by Dr. Harrison at al [6] where he gives an example of trust in a system of doing a purchase via the credit card. Where both the buyer and the seller trust the system. In case the system rejects the credit card of the buyer, both parties will not have less trust of the system but the seller will only be suspecting the buyer and most likely never the system.