

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MİKROSERVİS VE UPA GÜVENLİĞİ İÇİN KARMA YÖNTEM
TASARIMI

MEHMET NAZIM TAT
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE
2021

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MİKROSERVİS VE UPA GÜVENLİĞİ İÇİN
KARMA YÖNTEM TASARIMI

MEHMET NAZIM TAT
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
PROF. DR. İBRAHİM SOĞUKPINAR

GEBZE
2021

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**HYBRID DESIGN FOR MICROSERVICE
AND API SECURITY**

MEHMET NAZIM TAT
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
DEPARTMENT OF COMPUTER ENGINEERING

THESIS SUPERVISOR
PROF. DR. İBRAHİM SOĞUKPINAR

GEBZE

2021



YÜKSEK LİSANS JÜRİ ONAY FORMU

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 25/12/2020 tarih ve 020/60 sayılı kararıyla oluşturulan jüri tarafından 18/01/2021 tarihinde tez savunma sınavı yapılan Mehmet Nazım TAT'ın tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : Prof. Dr. İbrahim SOĞUKPINAR

ÜYE

: Doç. Dr. Mehmet GÖKTÜRK

ÜYE

: Doç. Dr. Ali Gökhan YAVUZ

ONAY

Gebze Teknik Üniversitesi Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

ÖZET

Günümüzde teknolojik hadiselerin, kötücül davranışların, uygulama ve ağ güvenlik zafiyetlerinin farklı boyutları ve türleriyle her geçen gün daha fazla karşılaşılmaktadır. Bu hadiselerin yanı sıra siber saldırı yöntemleri de teknolojinin gelişmesine paralel olarak artmaktadır. Siber güvenlik önlemleri sayesinde teknolojinin içinde bulunan birçok hata ve zafiyet tespit edilerek bunlara çözüm üretilmektedir.

Yazılım dünyasının yekpare (monolith) mimariden mikroservis mimarisine evrildiği bugünlerde mikroservis mimarisinin güvenlik açısından tekrar gözden geçirilmesi elzem hale gelmiştir. Mikroservis mimarisi, servis odaklı yaklaşımı temel alan bir felsefeden doğduğu için, servis odaklı yaklaşımın mevcut zafiyetlerini de barındırmaktadır. Bu bağlamda, mikroservislerin tasarımında yer alan servislerin ve UPA'ların güvenliğinin daha fazla artırılması gerekmektedir.

Bu çalışmada, mikroservis ve UPA'ların güvenliğinin artırılması amacıyla karma yöntem güvenlik tasarımı önerilmiştir. Tezimizin temelini oluşturan bu tasarım; sertifika, bilet ve e-imza tabanlı üç farklı güvenlik yönteminin hibrit bir şekilde, birlikte uygulanmasıyla geliştirilmiştir. Öncelikle servisler arası ağ güvenliğini sağlamak için karşılıklı TLS sertifikası kurulumu yöntemi (mTLS) uygulanmış olup, sonrasında servisler arası yapılan istek içerisine bilet bilgisi de eklenerek servisler arası bilet kontrolü yapılmıştır. Son aşama olarak yapılan istek, bilet bilgisini de içerecek şekilde elektronik imza ile CAdES formatında şifrelenmiştir. Böylece siber saldırılara karşı daha efektif bir koruma ve mimari tasarım oluşturulmuştur. Ayrıca karma yöntem tasarımında uygulanan güvenlik yöntemlerinin mevcutta var olan ortadaki adam saldırısı, siteler arası istek sahtekarlığı ve oturum korsanlığı gibi açıklıklarının ve zafiyetlerinin de önüne geçilmek suretiyle sistemin güvenlik seviyesinin artırılması sağlanmıştır.

Anahtar Kelimeler: Mikroservis, UPA, Siber Güvenlik, Kimlik Doğrulama, TLS, Servis Tabanlı Mimari.

SUMMARY

Today, technological incidents, malicious behavior, application and network security vulnerabilities are encountered more and more with different dimensions and types. In addition to these events, cyber-attack methods are also increasing in parallel with the development of technology. Thanks to cyber security measures, many errors and weaknesses in the technology are detected and solutions are produced.

Nowadays, when the software world has evolved from monolith architecture to microservice architecture, it has become essential to review the microservice architecture in terms of security. Since microservice architecture is born from a philosophy based on a service-oriented architecture, it also contains the current vulnerabilities of the service-oriented approach. Therefore, the security of services and APIs included in the design of microservices should be further increased.

In this study, hybrid method security design is proposed to increase the security of microservices and APIs. This design, which forms the basis of our thesis; It was developed by applying three different security methods based on certificate, token and e-signature in a hybrid manner. First, the mutual TLS certificate setup method (mTLS) was applied to ensure inter-service network security, and then the inter-service token control was carried out by adding token information to the inter-service request. As the last step, the request was encrypted in CAdES format with an electronic signature, including the token information. Thus, a more effective protection and architectural design against cyber-attacks has been created. In addition, the security level of the system has been increased by preventing the vulnerabilities and vulnerabilities of the security methods applied in the hybrid method design such as man-in-the-middle attack, cross-site request forgery and session riding.

Key Words: Microservice, API, Cyber Security, Authentication, TLS, Service Oriented Architecture.

TEŞEKKÜR

Başta, yüksek lisans eğitimimde ve akademik hayatımda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanım Prof. Dr. İbrahim SOĞUKPINAR'a

Bütün çalışmam boyunca gereken kolaylığı sağlayan ve yüksek lisans eğitimimin tamamında kendimi geliştirme konusunda önümü açan başta yöneticilerim olmak üzere tüm Kuveyt Türk ARGE merkezine,

Son olarak tüm bu süreçte maddi ve manevi desteği ile her zaman yanımda olan aileme ve sevgili eşim Nuray TAT'a en içten şükranlarımı sunarım.

İÇİNDEKİLER

	<u>Sayfa</u>
ÖZET	v
SUMMARY	vi
TEŞEKKÜR	vii
İÇİNDEKİLER	viii
SİMGELER ve KISALTMALAR DİZİNİ	ix
ŞEKİLLER DİZİNİ	x
TABLolar DİZİNİ	xi
1. GİRİŞ	1
2. KURAMSAL TEMELLER VE İLGİLİ ÇALIŞMALAR	3
2.1. Mikroservis Mimarisi	3
2.1.1. Mikroservis Tasarımı	5
2.1.2. Mikroservis Güvenlik Katmanları	7
2.2. UPA Kavramı Tanımı	8
2.3. Mikroservis ve UPA İlişkisi	10
2.4. Mikroservis Mimarisinin Avantajları ve Dezavantajları	10
2.5 Mikroservis Güvenlik Yöntemleri ve İlgili Çalışmalar	13
2.5.1. Kullanıcı Adı ve Şifre Tabanlı Kimlik Denetleme	14
2.5.2. Bilet (Token) Tabanlı Kimlik Denetleme	14
2.5.3. Sertifika Tabanlı Kimlik Denetleme	18
2.5.4. Elektronik İmza Tabanlı Kimlik Denetleme	19
3. KARMA YÖNTEM GÜVENLİK TASARIMI	20
3.1 Matematiksel Model	24
4. DENEYSEL SONUÇLAR ve DEĞERLENDİRME	27
4.1. KYGT Karşılaştırmalı Performans Analizi	27
4.2. KYGT ve Yekpare Mimari Performans Karşılaştırması	30
5. SONUÇ VE GELECEK ÇALIŞMALAR	32
KAYNAKLAR	34
ÖZGEÇMİŞ	36

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler ve Açıklamalar

Kısaltmalar

AKD	: Açık Kimlik Denetleme
BOEP	: Basit Obje Erişim Protokolü
CAdES	: CMS Advance Electronic Signature
CRUD	: Create - Read - Update - Delete Operations
ÇOA	: Çoklu Oturum Açma
HTTPS	: Hyper Text Transfer Protocol Secure
JSON	: Javascript Obje Notasyonu
JWB	: Json Web Bileti
KYGT	: Karma Yöntem Güvenlik Tasarımı
mTLS	: Mutual Transport Layer Security
OWASP	: Open Web Application Security Project
OWASP	: Open Web Application Security Project
PAdES	: PDF Advance Electronic Signature
SAML	: Security Assertion Markup Language
SOA	: Service Oriented Architecture
SOA	: Service Oriented Architecture
TCKN	: Türkiye Cumhuriyeti Kimlik Numarası
TDT	: Temsili Durum Protokolü
TLS	: Transport Layer Security
UPA	: Uygulama Programlama Arayüzü
VKN	: Vergi Kimlik Numarası
XAdES	: XML Advance Electronic Signature
XML	: Extended Markup Language

ŞEKİLLER DİZİNİ

<u>Sekil No:</u>	<u>Sayfa</u>
2.1: Yekpare Mimari ve Mikroservis Mimarisi.	4
2.2: Yekpare Mimari.	4
2.3: Mikroservis Mimarisi.	5
2.4: Mikroservis Ayırıştırma Yöntemi.	6
2.5: Mikroservis Ayırıştırma Algoritması.	7
2.6: AKD 2.0 (OAuth2.0) Protokol Akışı.	15
2.7: ÇOA Protokolü.	16
2.8: MTLS Çalışma Sistemi.	18
3.1: Karma Yöntem Güvenlik Tasarımı.	20
3.2: Borç Sorgulama Örnek Akışı.	22
3.3: Tahsilat İşlemi Örnek Akışı.	23
3.4: KYGT Çizgesi.	25
4.1: KYGT Karşılaştırmalı Performans Analizi.	27
4.2: KYGT ve Yekpare Mimari Performans Analizi.	29

TABLÖLAR DİZİNİ

Tablo No:

Sayfa

2.1: Hiyerarşik Mikroservis Güvenlik Katmanları.

8

1. GİRİŞ

Mikroservis tabanlı sistemler, birçok farklı işlev yapan küçük servislerin bağımsız olarak var olması ve bunların aralarında iletişim kurmasıyla meydana gelmiş, servis tabanlı mimari (Service Oriented Architecture) felsefesinden ortaya çıkmış sistemler bütünüdür. Mikroservis mimari yaklaşımının iki temel özelliği vardır. Bunlardan ilki izole olmasıdır. Yani her bir servis kendi içinde bağımsız olarak yaşam döngüsünü sürdürür. Diğer servislerin durması veya bozulması mevcut servisin işleyişini etkilememelidir. İkincisi ise kolay ölçümlenebilmesidir. Bu sayede sistem yükü ve dengesi çok daha kolay yönetilebilir. Bu mimari yaklaşım her ne kadar büyük ve karmaşık sistemler için birçok avantaj sağlasa da güvenlik zafiyeti gibi zorlukları da beraberinde getirmektedir. Son zamanlarda açıklanan “WhiteHat Security” şirketinin tehdit raporuna [20] göre Netflix gibi mikroservis mimarisine sahip şirketlere birçok saldırı yapılmıştır [1]. Bu saldırılar belli başlı servislerden tüm sisteme kadar her bir noktayı kapsayıcı bir şekilde gerçekleştirilmektedir. Bu bağlamda şirketler, çeşitli güvenlik standartları ve mekanizmaları oluşturarak kendilerini bu tip saldırılardan korumayı amaçlamaktadırlar.

Güvenli yazılım geliştirebilmek için tüm zayıflıkların göz önünde bulundurulması ve karşılaşılan problemleri en aza indirebilmek için doğru çözümlerin uygulanması gerekmektedir. Bu konuyla alakalı Açık İnternet Uygulamalarının Güvenliği Projesi (OWASP) isimli oluşum [2] sayesinde periyodik olarak en çok maruz kalınan zafiyet türlerinin sıralamasını aktif olarak takip edilebilmektedir. En son 2017 sürümünde açıklanan dokümana göre zafiyet sıralaması aşağıdaki liste şeklindedir.

- i) Enjeksiyon (Injection)
- ii) Kimlik Denetleme Kırma (Broken Authentication)
- iii) Hassas Data Sömürüsü (Sensitive Data Exposure) zafiyetleri almaktadır.
- iv) XML Sömürüsü (XML External Entities)
- v) Erişim Kontrolü Kırma (Broken Access Control)
- vi) Yanlış Güvenlik Konfigürasyonu (Security Misconfiguration)
- vii) XSS Zayıflığı (Cross-Site Scripting)
- viii) Güvenli Olmayan Serileştirme (Insecure Deserialization)

- ix) Bilinen Zayıflığa Sahip Araçların Kullanımı (Using Components with Known Vulnerabilities)
- x) Yetersiz İz Kaydı ve İzleme (Insufficient Logging&Monitoring)

Yukarıdaki her bir madde, yazılım ve uygulama güvenliği açısından dikkat edilmesi elzem olan kriterleri içermektedir. Bu bilgiler ışığında güvenlik konusunda hangi alanlardaki zafiyetlerin güncel olarak güvenlik kırıncılar tarafından sömürüldüğü bilgisi izlenebilmektedir.

Güvenli mikroservislerin geliştirilmesi için şimdiye kadar ortaya konulan standartlar yeteri kadar belirli ve keşfedilmiş değildir. Örneğin, günümüzde UPA'ların güvenliği için sıkça kullanılan güvenlik yöntemlerinden biri olan AKD2.0 gibi yöntemlerin state parametresini kullanmadıklarında CSRF ataklarına karşı zaafiyetleri tespit edilmiştir [3]. Mikroservis ve UPA'lar için bugüne kadar AKD2.0, Açık Kimlik Oturumu (Single Sign On), Güvenlik Onayı Biçimlendirme Dili (Security Assertion Markup Language), Json Web Bileti (Json Web Token) gibi birçok güvenlik yöntemi ve literatürde bunları temel alarak geliştirilmiş ekstra yöntemler önerilmiştir.

Bu bilgiler ışığında önerilen yaklaşımlar incelenerek, karma yöntem güvenlik tasarımında uygulanacak olan çözümün altyapısı oluşturulmuştur. Bu çalışmanın ikinci bölümünde mikroservis mimarisi ve UPA hakkında yapılan çalışmaların tarihçesi, kavramsal tanımları ve kuramsal temelleri üzerine incelemelerde bulunulmuştur. Üçüncü bölümünde ise literatürde UPA, webservis, mikroservis ve servis tabanlı mimari konusunda önerilen güvenlik yöntemleri incelenmiş olup, bu yöntemlerin avantaj ve dezavantajları tespit edilerek, burada uygun olan yöntemler çözümün bir parçası olarak uygulanmıştır. Dördüncü bölümde ise “Karma Yöntem Güvenlik Tasarımının” detayları ile ilgili örnek bir iş akışı ile çözümün prototip üzerinden teyit edilmesi sağlanmış olup, deneysel sonuçlar paylaşılmıştır. Son bölümde ise elde edilen kazanımlar, ortaya çıkan sonuç ve geleceğe dair potansiyel olarak yapılabilecek çalışmalar önerilmiştir.

2. KURAMSAL TEMELLER VE İLGİLİ ÇALIŞMALAR

Bu bölümde mikroservis mimarisi ve UPA üzerine bazı temel kavramlar ve bu kavramlara ait ayrıntılar açıklanmıştır. Temel olarak ana başlıklar belirlenerek, öncelikle mikroservis kavramının tanımı, tasarımı ve güvenlik katmanları konularında incelemelerde bulunuldu. Bu konuların ardından UPA kavramı tanımı, mikroservis ve UPA ilişkisi, mikroservis mimarisinin avantaj ve dezavantajları konuları açıklamalı olarak birkaç alt başlıkta tartışılmıştır.

2.1. Mikroservis Mimarisi

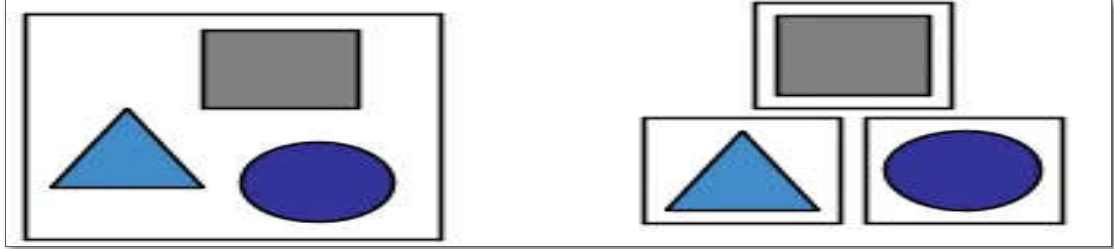
Mikroservis mimarisi temel olarak hizmet/servis tabanlı mimariden (SOA) evrilerek düşünülmüş ve geliştirilmiş, yekpare (monolith) mimariye alternatif olarak yazılım dünyasında son zamanlarda epeyce rağbet görmüş bir yaklaşımdır.

Mikroservisin tarihçesi incelendiğinde ilk olarak bu terimin 2005 yılında bulut bilişim konferansında Dr. Peter Rogers tarafından “Mikro İnternet Hizmetleri” adıyla kullanıldığı görülmektedir [4].

Mayıs 2011’de Venedik yakınlarında düzenlenen bir yazılım mimarı atölyesinde katılımcıların çoğunun ortak olarak gördükleri bu mimari yaklaşımı tanımlamak için “mikroservis” terimi kullanıldı. Mayıs 2012’de, aynı grup en uygun isim olarak “mikro hizmetlere” karar verildi. James Lewis bu fikirlerden bazılarını Mart 2012’de, Fredros’un aynı zamanda Fred George’da olduğu gibi, Kraków’da (Java, Unix Yolu) 33. Derecede bir vaka çalışması olarak sunuldu. Bu yaklaşımı "ince taneli hizmet tabanlı uygulama" olarak nitelendiren Netflix’teki Adrian Cockcroft, bu makalede bahsedilen diğer pek çokları - Joe Walnes, Dan North, Evan Bottcher ve Graham Tackley gibi stilin öncülüğünü yapmıştır [4], [6].

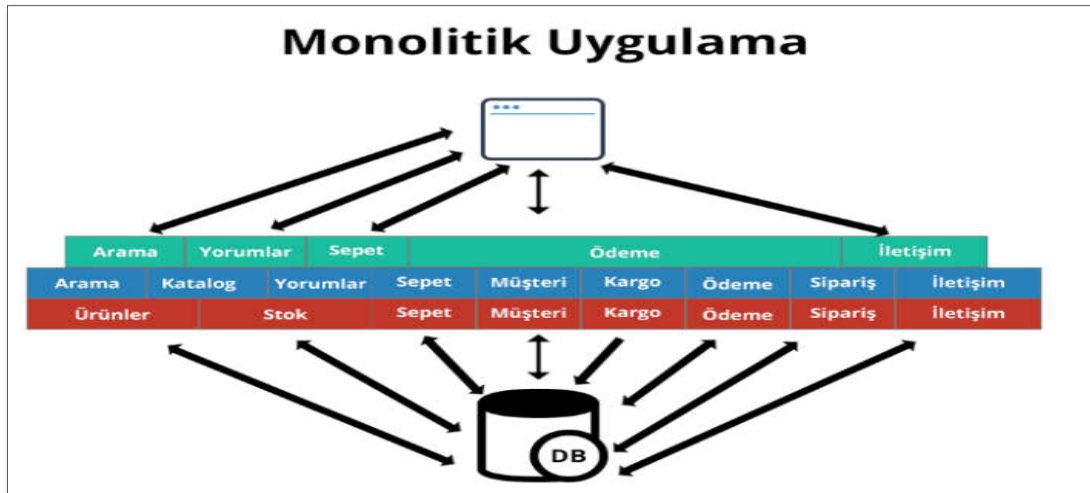
Mikroservis mimarisinin popüler olmasının bir nedeni de hiç kuşkusuz bu mimariye geçiş yapmış olan popüler ve itibar sahibi büyük ölçekli teknoloji firmalarının olmasıdır. Bu firmalardan bazıları Comcast Cable, Uber, Netflix, Amazon, Ebay, Sound Cloud, Karma, Groupon, Hailo, Gilt, Zalando, Keynote, Lending Club, AutoScout24 gibi büyük ölçekli şirketlerdir [7].

Mikroservis mimari yaklaşımının yapılmak istenirse; “Her biri kendine ait görevler yapan ve genellikle bir HTTP kaynak UPA’sı olan ve mümkün olan en az bağımlılık prensibiyle diğer mekanizmalarla iletişim kuran küçük hizmetler paketleri olarak tek bir uygulama geliştirmeye yönelik bir yaklaşım” denilebilir.



Şekil 2.1: Yekpare Mimari ve Mikroservis Mimarisi.

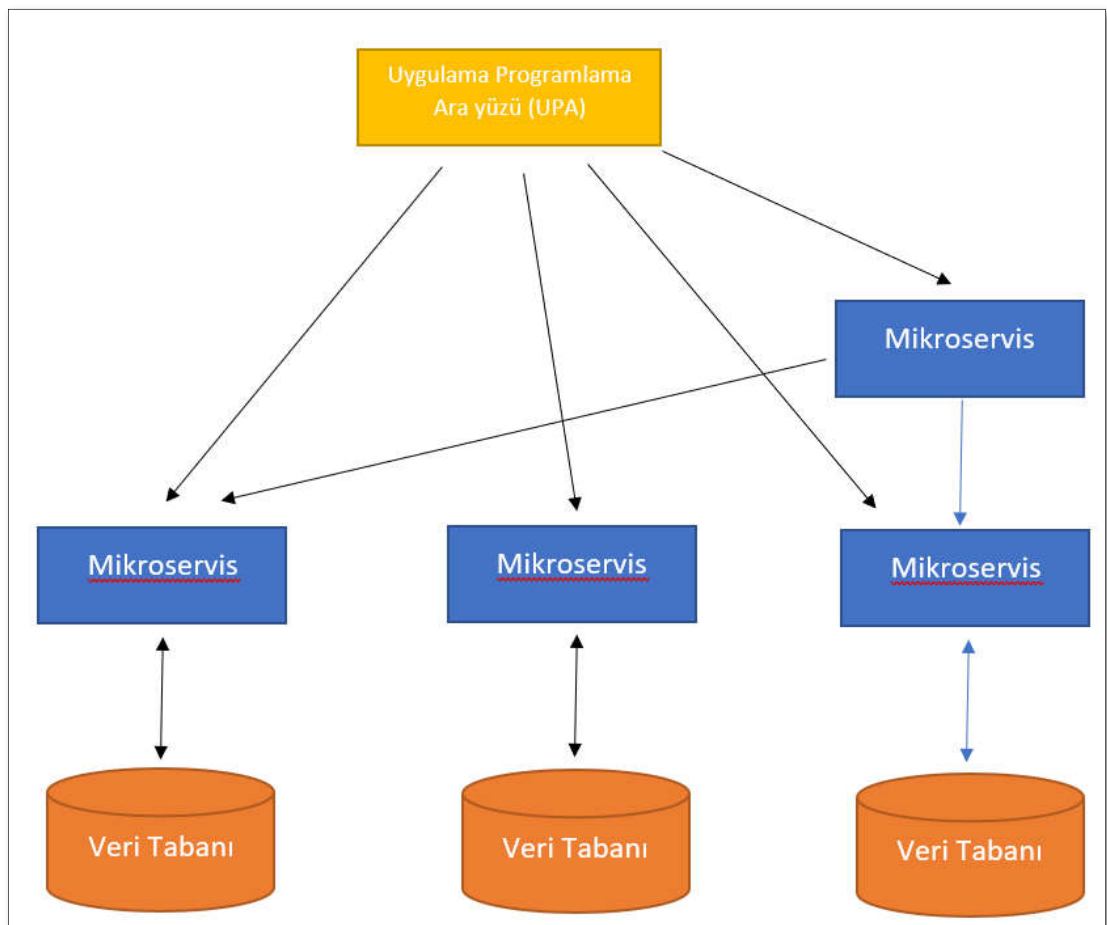
Mikroservis mimarisi, yekpare mimariye alternatif olarak ortaya çıkmış ve yekpare mimari yaklaşım felsefesini değiştirmeyi hedef almış mimari bir yaklaşım olarak literatürde yer almaktadır. Şekil 2.1’de görüldüğü üzere yekpare mimaride sistem, bütün katmanların ve servislerin birbiriyle sıkı sıkıya bağlı olduğu bir yapıdan meydana gelirken; mikroservis mimarisine sahip sistemde, her bir servis kendi içerisinde tamamen bağımsız bir şekilde, diğer servislerin varlığından bağımsız olarak yaşam döngüsünü sürdürerek var olmaktadır.



Şekil 2.2: Yekpare Mimari.

Yekpare mimari detaylı olarak incelendiğinde karşılaşılan yaklaşım, N katmanlı mimarinin birbiriyle sıkıca bağlı ilişki (tightly coupled) içerisinde olduğu modeldir. Şekil 2.2’de görüldüğü gibi yekpare N katmanlı mimari, yazılım dünyasında çoğunlukla üç temel katmandan ($N=3$) oluşur. Bunlar istemci (client), sunucu (server) ve veri tabanı (database) katmanlarıdır. Bu katmanlar sistem tasarımı ve ihtiyaca göre artabilir. ($N>3$)

Mikroservis mimarisinde yekpare mimariden farklı olarak Şekil 2.3’te görüleceği üzere her bir servisin kendi veri tabanına ve bağımsız sisteme sahip olduğu, diğer servisler ile mesaj tabanlı olarak iletişim kurulduğu görülmektedir.

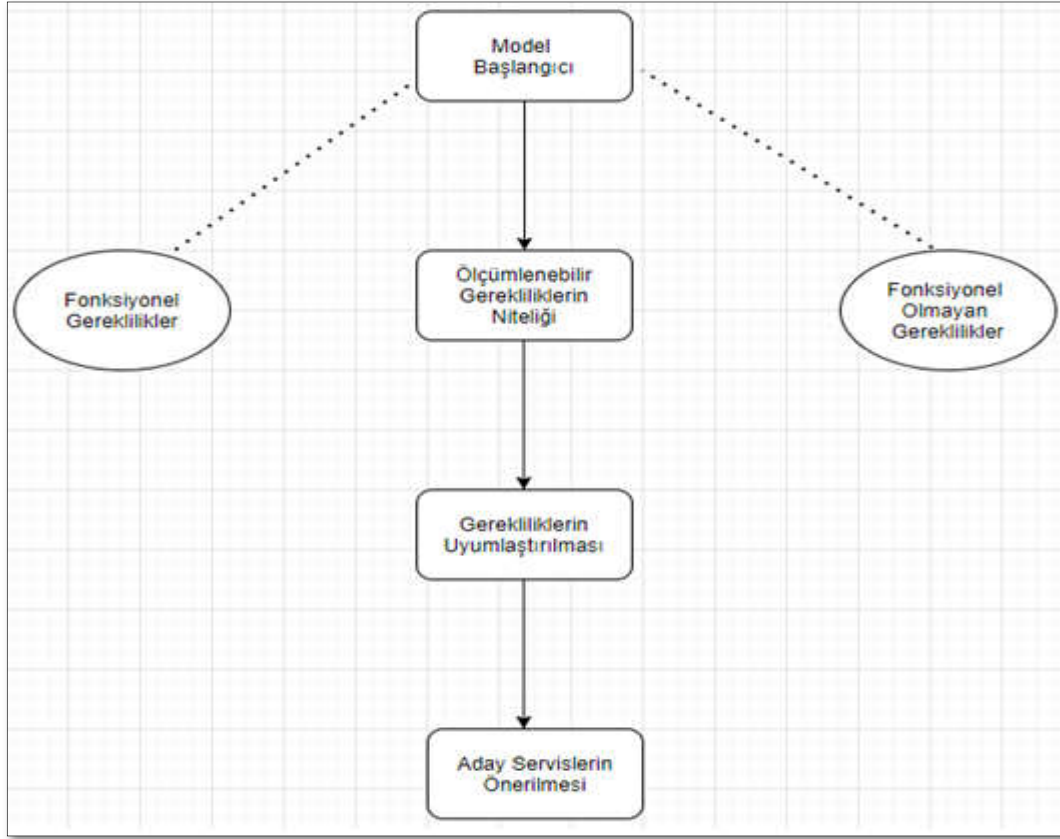


Şekil 2.3: Mikroservis Mimarisi.

2.1.1. Mikroservis Tasarımı

Mikroservis mimarisi, yekpare mimariye alternatif olarak ortaya çıkan ve günümüzde çok fazla ilgi çeken yazılımsal tasarım deseni olarak tanımlanabilir. Bu

durumda tek parça mimariye sahip sistemler, mikroservis mimarisine nasıl geçebilir sorusuna cevap aramaktadırlar. Ahmadvand and Ibrahim bu konuda yapmış oldukları çalışmayla [8] adım adım tek parça mimariden mikroservis mimarisine nasıl geçiş yapılacağı noktasını aydınlatmışlardır.



Şekil 2.4: Mikroservis Ayrıştırma Yöntemi.

Şekil 2.4’te görüldüğü üzere [8] tüm sistem fonksiyonel ve fonksiyonel olmayan gereklilikler olarak iki bölüm olarak kabul edilir. Her bir fonksiyonalitenin diğer fonksiyonlarla olan ilişkisinin hiyerarşisi çıkarılır ve gerekliliklerin bağımlılık analizi yapılır [9]. Her bir bağımlılık için, bağımlılığın sıklığını belirten “bağımlılık ağırlığı” (dependency weight) değeri kullanılır. Örneğin bir alışveriş sitesinde fatura ve alışveriş sepeti arasındaki bağımlılık değeri yüksek iken, fatura ve ödeme metodlarının metadata bilgisi servislerinin bağımlılık değeri düşüktür. Çünkü bu ilişki günde sadece bir kere ihtiyaç duyulacak şekildedir. Son olarak sistemin güvenlik gereksinimleri ve politikaları, fonksiyonel gerekliliklerin karşılanmasını artırmak için kullanılmalıdır.

Örneğin, fatura mikroservisi kişisel verileri içerdiği için erişim kontrolü, yetkilendirme ve güvenli bağlantı gibi özelliklerle korunmalıdır.

```
1:  $ML \leftarrow \text{InitMicroserviceList}$ 
2:  $FR \leftarrow \text{FunctionalRequirement}$ 
3:  $V \leftarrow \text{DependantRequirement}(FR)$ 
4:  $S \leftarrow \text{ScalabilityRequirement}(FR)$ 
5:  $P \leftarrow \text{SecurityPolicy}(FR.Assets, FR.Threats)$ 
6: if Scalability is a requirement of FR then
7:    $MS \leftarrow FR$ 
8:   for all  $v \in V$  do
9:      $v_u \leftarrow$  dependency weight of the relation  $FR - v$ 
10:     $v_p \leftarrow$  security policy of the dependant requirement
11:    if  $\text{Max}(v_u, FR.P, v_p) = \text{High}$  then
12:       $MS \leftarrow \text{merge}(FR, v)$ 
13:    end if
14:  end for
15:   $ML \leftarrow \text{add}(MS)$ 
16: end if.
```

Şekil 2.5: Mikroservis Ayırıştırma Algoritması.

Şekil 2.5'teki algoritma [8] ile yekpare bir sistem, adım adım yukarıdaki aşamalardan geçerek mikroservis mimariye sahip bir sisteme dönüştürülebilmektedir. Burada nihai olarak ulaşılabilecek olan nokta, yekpare sistemin bağımsız parçalara ayrılarak mikroservislere ayrılmış olarak hizmet verilebilmesinin sağlanmasıdır.

2.1.2. Mikroservis Güvenlik Katmanları

Mikroservis güvenliği, literatürde temel olarak iki katman üzerinde yoğun olarak incelenmiştir. Bunlar ağ ve uygulama katmanlarıdır. Ancak Yarygina and Bagge [10] bu konuda “Açık Sistem Arabağlantısı (OSI)” mimarisinden de esinlenerek mikroservis güvenliğini 7 katmanlı olarak incelemiştir ve bu bakış açısı diğer temel incelemeleri de kapsayan çok daha kapsamlı bir yaklaşım oluşturmuştur.

KATMAN	ÖRNEK TEHDİT	ÖRNEK ÇÖZÜM
DONANIM	Donanımın üretim zamanında kasıtlı veya kasıtsız olarak açık kapı bırakılmış veya bir hata yapılmış olabilir.	Donanım güvenlik modülü kullanarak kendi donanımımız tasarlayabiliriz.
SANALLAŞTIRMA	Sanal makine üzerindeki yaygınlaştırmalar, operasyonlar, işletim sistemleri ele geçirilebilir. Örnek saldırılar; kum havuzundan kaçış saldırısı, paylaşılan bellek saldırısı, zayıflık içeren imaj kullanımı.	Yaygınlaştırmalar güçlü bir izolasyon ile yapılmalıdır. Paylaşımlı bellek veya kütüphaneler kullanılmamalıdır. Yazılım ve imaj güncellemeleri zamanında yapılmalıdır.
BULUT	Bulut bilişim sistemi üzerinde sayısız açıklıkta beraberinde getirmektedir. Bulut sistem sağlayıcıları limitsiz kontrol sunmaktadır.	Ters kum havuzu koruması sağlanarak işletim sistemine uzaktan erişimle yapılan saldırıları engelleyebiliriz.
AĞ	Ağ yığınlarına ve protokollerine yapılan klasik saldırılar. Bu saldırılar ağ dinlemesi, kimlik aldatması, oturum hırsızlığı, servis durdurma saldırılarını da içermektedir.	TLS veya JSON gibi onaylanmış ve doğrulanmış protokoller kullanılmalıdır.
UYGULAMA	SQL Enjeksiyonu, XSS, yanlış konfigürasyon gibi OWASP 10 da yer alan tüm saldırılar.	Statik veya dinamik kod analizi, düzenli kod gözden geçirme işlemi, temel yazılım mühendisliği pratikleri yapılıp, uygulanmalıdır.
YÖNETİM	Servisin bulunduğu ağ yapısı değişebilir. Örneğin DNS saldırıları gibi merkezi ataklar gerçekleşebilir.	Ağa kayıt olan cihazlar kontrol edilmelidir.

Tablo 2.1: Hiyerarşik Mikroservis Güvenlik Katmanları.

Tablo 2.1’de belirtilen donanım, sanallaştırma, bulut, iletişim, servis/uygulama ve yönetim katmanları ile alakalı güvenlik çözümleri de ayrı ayrı düşünülmelidir. Çünkü her bir katman ayrı güvenlik ihtiyacına sahip olmakla birlikte, bir önceki veya bir sonraki katmandan doğabilecek güvenlik açığından dolayı tüm sistem istismar edilebilir. Örneğin iletişim (communication) katmanı için TLS veya JSON güvenlik standartları gibi çözümleri uygulamak gerekirken, uygulama seviyesinde SQL Enjeksiyonu önleme gibi çözümler gerekebilir. Bu çözümleri her bir katman için ayrı ayrı planlamak ve uygulamak gereklidir.

2.2. UPA Kavramı Tanımı

UPA (Uygulama Programlama Arayüzü) yani “Application Programming Interface (API)”, bir uygulamanın işlevlerine dışarıdan veya uzaktan erişilip bu işlevlerin kullanılmasını sağlayan arayüzdür. UPA hazırlayan ve sunan sistem sahibi, sunduğu bu özelliğin başka sistemler tarafından kullanılmasını ister ve böylece hem

UPA sahibi hemde UPA'yi kullanan sistemler kazan-kazan felsefesiyle çift taraflı olarak menfaat elde ederler. Örneğin, Google firması arama motoru kullanımı için web sitesi sahiplerine ücretsiz UPA hizmeti sunmaktadır. Böylece Google, kendi arama motorunu tüm dünya üzerinde yaygınlaştırarak kendisine olan bağımlılığı artırmakta ve piyasadaki rakiplerine karşı elini güçlendirmektedir. Ayrıca UPA üzerinden yapılan işlemlerdeki veriler üzerinden algoritmalarını geliştirme fırsatı buluyorken; bu UPA'ları kullanan web sitesi sahipleri ise çok düşük bir kod maliyeti ile Google gibi profesyonel bir arama motorunu kendi web siteleri üzerine entegre ederek, web sitesi kullanıcılarına arama motoru hizmeti sunmuş olmaktadır.

UPA'ların birçok farklı kullanımı mevcuttur. Bunları temel olarak çevrimiçi ve çevrimdışı olarak iki ana gruba ayırabiliriz. Çevrimdışı UPA'ları yazılım geliştiricilerin kullanmış oldukları kütüphaneler olarak örneklendirebiliriz. Yani herhangi bir şekilde kullanımı için ağ bağlantısı gerektirmeyen UPA'lar bu sınıfa girerler. Genelde bu UPA'lar sistemlerde gömülü olarak çalışır ve UPA sahibi tarafından kolay kolay değişikliğe uğramazlar.

Çevrimiçi UPA'ları ise temel olarak ağ üzerinden iletişim kurulan servisler olarak tanımlayabiliriz. Bunlar çoğunlukla bir sunucunun üzerindeki uygulamaya farklı platformlardan yapılan isteklere (request), cevap (response) dönmesine olanak sağlar. Ağ üzerindeki UPA'ların birçoğu “Temsili Durum Transferi (REST)” veya “Basit Obje Erişim Protokolü (SOAP)” mimarisi üzerinde tasarlanır. Bu bağlamda platform bağımsız çalışmayı esas alırlar. BOEP mimarisi GET, POST, PUT, DELETE metodlarının hepsini desteklemektedir. Web UPA çıktıları talebe göre JSON, XML gibi çeşitli çıktıları olabilir [11].

Herhangi bir sisteme ait UPA'ları kullanarak, sistemin kendisine ihtiyaç duymadan sunulan özelliklerini, fonksiyonlarını veya içerikleri edinilebilir. Çoğunlukla erişimler Internet Protokolü (IP) kısıtı, sorgu limiti, hassas veri gibi belirli sınırlandırmalara sahiptir ve tüm bu erişimlerin iz kayıtları (log) tutulur. Ayrıca gerçekleştirmek istenilen işlemler için erişim sağlamak istenilen uygulama tarafından özel olarak sunulan erişim bilgilerinin kullanılması gereklidir.

2.3. Mikroservis ve UPA İlişkisi

UPA ve mikroservisler arasında işleyiş açısından ortak noktalar bulunmaktadır. Mikroservis mimarisini oluşturan her bir servis aslında birer çevrimiçi UPA'dır. Bu servisler dâhili sistem içerisinde çalıştığı gibi harici sistemler tarafından da kullanılabilmesi için dışarı UPA olarak sunulabilirler. Bu bağlamda her bir mikroservis, harici sistemlere açılabilme potansiyeli bulunan bir UPA olarak güvenlik açısından değerlendirilip tasarlanmalıdır.

2.4. Mikroservis Mimarisinin Avantajları ve Dezavantajları

Mikroservis mimarisi, dikey bir şekilde büyüyen yekpare mimarideki sistemlerin beraberinde getirdiği karmaşıklığı, ölçeklemeyi ve yönetim zorluklarını yatay büyüme yaklaşımıyla çözmeye odaklanmaktadır. Geleneksel servis tabanlı mimari (SOA) yerine yenilikçi servis tabanlı mimari yaklaşımı ile aşağıda bahsedeceğimiz avantajları sağlamaktadır:

- i) Teknoloji çeşitliliği
- ii) Esneklik
- iii) Ölçekleme
- iv) Kolay sürüm yaygınlaştırma
- v) Şirket organizasyon kolaylığı
- vi) Çok amaçlı servis tasarlama
- vii) Sistem iyileştirme kolaylığı

Mikroservislerin sağlamış olduğu teknoloji çeşitliliği sayesinde yekpare sistemlerde olduğu gibi belirli teknolojilere bağımlı kalma durumu söz konusu değildir. Örneğin sunucu tarafında yazılım dili C# kullanılarak kodlanmış, veri tabanı olarak MS-SQL¹ kullanılmış bir sistem geliştirilmesi istenildiğinde her ürünün bu teknolojilere bağımlılığı var olacaktır. Böylece bu sistem üzerinde Oracle² veya No-SQL tarzı veri tabanlarını kullanmak istenildiğinde neredeyse tüm sistemi dönüştürerek kullanım sağlanabilecekti. Ancak mikroservis mimarisinde her bir servis farklı programlama dili veya veri tabanı bağımsız olarak tasarlanıp, kullanıma

sunulabilir. Örneğin bir servis C++ programlama dili ile kodlarken, diğer bir servis Java programlama dili ile kodlanabilmektedir. Bu durum hem teknoloji çeşitliliğine olanak sunarken hem de farklı teknolojileri bilen yazılım geliştiricileri aynı çatı altında buluşturabilmeyi sağlamaktadır. Mikroservis mimarisi ile yeni gelişen teknolojilere daha rahat ayak uydurulabilir ve tüm sistemi tamamıyla değiştirmeden yeni teknolojiler sistemin ilgili yerlerine entegre edilebilmektedir. Yeni teknolojiyi tecrübe etmek için karşılaşılan engellerin başında, projede değiştirilecek kısım ile yoğun bağımlılığın olması ve bir yere dokunulduğunda arkasının da geleceği riskidir. Birçok servisten oluşan bir sistemde yeni bir teknolojiyi tecrübe etmek için birden fazla alternatif servis vardır. Bu tecrübe etme işlemi için düşük riskli bir servis seçilebilir ve olası olumsuz etkilerin en az olacağı tahmin edilen bir teknoloji kullanılabilir.

Mikroservis mimarisinin diğer büyük bir avantajı esnek veya elastik oluşudur. Yekpare mimari üzerinde sistemin parçalarından birinde oluşan herhangi bir aksaklık tüm sistemi olumsuz olarak etkilendiği için genelde bu sistemler yedekli bir yapıda çalışırlar. Mikroservis mimarisi dağıtık bir yapıda olduğundan aksaklık oluşturan servis veya düğüm, gerekli olduğunda hızlı bir şekilde başka bir yerden ayağa kaldırılabilir. Ayrıca ihtiyaç durumunda hizmet veren bir servis çoklanarak sistem yükü yatay olarak yayılabilir.

Ölçekleme konusunda mikroservis mimarisinin yekpare mimariye göre çok büyük avantajı vardır. Yekpare mimaride sistemde, yük oluşturduğu düşünülen parçanın ölçeklendirilebilmesi için o parçaya bağlı tüm parçaların etkisi hesaplanarak ölçeklendirme yapılmalıdır. Örneğin, yekpare sistem ile yazılmış bir mobil uygulamanın açılış sayfasındaki performansı ölçeklendirmek için o sayfa yüklenirken yapılan tüm web servis ve veri tabanı sorguları incelenmesi gerekirken, mikroservis mimarisinde ise hangi servisin ne kadar sürede cevap verdiği, kaç adet istek aldığı gibi değerler ölçeklenerek, sistem kolayca gözlemlenebilmektedir.

Kolay bir sürüm yaygınlaştırma (deployment) işlemi mikroservis tabanlı sistemler için çok hızlı ve kolaydır. Burada tüm sistemden ziyade belirli servisler üzerinde işlem gerçekleştirilerek sürüm yaygınlaştırma işlemleri kolay bir şekilde yapılabilir. Ayrıca bu sayede olası çıkabilecek problemlere anında müdahale edebilme imkânı bulunmaktadır.

¹ Microsoft firmasına ait veri tabanı sistemleri yönetimi ürünü.

² Oracle firmasına ait veri tabanı sistemleri yönetimi ürünü.

En önemli avantajlarından biri ise tüm sistemi durdurmak zorunda kalmayıp, ilgili servisin gerekirse bir yedeği ile çalışmasını sağlayıp, sürüm yaygınlaştırma işlemi bittikten sonra tekrar yeni servise yönlendirerek genel sistemde hiçbir kesinti olmadan bu işlem gerçekleştirilebilmektedir. Yekpare sistemlerde ise çoğu zaman tüm sistemi durdurmak ve tekrar ayağa kaldırmak zorunda kalarak sürüm yaygınlaştırma işlemi gerçekleştirilmektedir.

Mikroservis mimarisi için sistem tasarımı yapılırken, ihtiyaç olan iş bilgisine ait organizasyonlar kolaylıkla tesis edilebilir. Servisin yazılım işlerini yapacak bilgi teknolojileri takımı ile iş bilgisi ihtiyaçlarını belirleyecek iş birimi takımları daha net görevlere sahip olarak birbiriyle daha koordineli olarak çalışma imkânına sahip olurlar. Örneğin, bankacılık sisteminde krediler iş biriminin iş gereksinimlerini karşılayan mikroservisleri kodlayan ekibi ayrı bir servis olarak konumlandırılırken, bunun yanında temel bankacılık operasyonlarını yürüten ekibi ayrı bir servis olarak konumlandırılabilir. Bu sayede mevcut iş ihtiyacının mahiyetine göre kullanılabilecek teknoloji de farklılaştırılabilir. Örneğin analitik bankacılık servislerinin yazılımını geliştiren bir ekiple python dili kullanarak geliştirmeler yapılabilirken, başka bir serviste C# kullanabilir ve bu ihtiyaçlar doğrultusunda yeni ekip üyeleri personel olarak şirkete alınabilmektedir.

Mikroservislerin bir diğer avantajı ise çok amaçlı olarak tasarlanabilmesidir. Örneğin bir servis hem UPA olarak dış kurumların erişebileceği bir servis olarak tasarlanabilirken, aynı zaman sistemin kendi içerisinde hizmet verecek bir yapıda da tasarlanabilmektedir. Böylece farklı amaçlara hizmet eden bir servis tasarımı mümkün hale gelebilmektedir.

Son olarak birçok avantajından bahsedilen mikroservisler ile sistem üzerinde yapılmak istenilen herhangi bir iyileştirme işlemi kolay bir şekilde gerçekleştirilebilmektedir. Çünkü sistem, mikro seviyede servislerden oluştuğundan yapılacak değişiklikler mikro düzeyde gerçekleştirilebilecektir. İyileştirme yapılmak istenen mikroservisin kullandığı teknoloji veya altyapı gibi değişiklikler, servisin dönüş değeri değiştirilmediği sürece istenilen şekilde düzenlenebilmektedir. Böylece teknolojik bir dönüşüm için sistemin uygunluğu güvenli bir şekilde tecrübe edilebilmektedir.

Mikroservislerin yukarıda saymış olduğumuz tüm bu avantajlarının yanında bazı dezavantajları da mevcuttur.

- i) İşlem (Transaction) yönetimi
- ii) Kod tekrarı
- iii) Yönetimsel zorluklar

İşlemsel bütünlük, özellikle finansal hizmet sunan sistemlerde büyük bir önem arz etmektedir. Bir istemciden başlatılan isteğin başlangıcından sonuçlanıncaya kadarki bütüncül hareketine işlem (transaction) denilmektedir. Kısaca bellek üzerinde yapılan bir grup işlemin atomik (indivisible) bir şekilde gerçekleşmesi de denilebilir [5]. Örnek olarak bir bankaya ait mobil şubeden fatura ödeme işlemi yapılırken, müşteriye ait finansal bilgilerin tutulduğu muhasebe veri tabanı tabloları, faturanın durumunun sorgulandığı fatura kurumuna ait UPA'lar ve müşterinin bu işlemi yapmaya yetkisinin kontrolünün yapıldığı veri tabanı sorgularının hepsi bütüncül olarak gerçekleştirilmesi gerekmektedir. Bu adımlardan birisinin eksik veya fazla olması ödeme işleminin yanlış bir sonuç üretmesine sebep olur. Yekpare sistemlerde bu tip işlemleri ilişkisel veri tabanlarının sağlamış olduğu işlem yönetimi (transaction) sayesinde kolay bir şekilde gerçekleştirilebilmektedir. Ancak mikroservis mimarisine sahip sistemlerde buradaki adımlardan birisinde oluşacak hatada tüm işlemin manuel bir şekilde geri alma işlemi (rollback) yapılması gerekmektedir.

Mikroservis mimarisine sahip sistemlerde yaşanan diğer bir zorluk ise kod tekrarından kaçınılamamasıdır. Çünkü her bir servis kendi iç dünyasında bağımsız ve izole bir şekilde çalışma felsefesine sahip olduğundan, diğer bir servise bağımlı olmaması gerekmektedir. Bu durumda aynı işi yapan kod parçacıklarının her bir servis için kopyalanması gerekebilmektedir.

Mikroservislerin sayısının çok fazla olması bunların yönetimini ve birbirleriyle olan ilişkilerini düzenleme konularında hem daha fazla dikkat edilmesi hem de uygun araçlarla bunların kontrol edilmesi gerekmektedir. Her bir servise ait veri tabanı, kod bakımı gibi işlemlerin yönetimi servis bazlı olarak düşünülerek kurgulanmalıdır.

2.5 Mikroservis Güvenlik Yöntemleri ve İlgili Çalışmalar

Mikroservis ve UPA'lar açısından güvenlik yöntemleri, servis odaklı mimaride uygulanan güvenlik yöntemleri ile paralellik göstermektedir. Uygulama katmanı tarafında tasarlanmış güvenlik yöntemi adımları genel olarak iki başlık altında

sınıflandırılabilir. Bunlardan ilk adım kimlik denetleme (Authentication) iken diğeri ise yetkilendirmedir. Kimlik denetleme (Authorization), UPA'lara erişim sırasında ilgili isteğin güvenlik açısından analiz edilmesini esas alır. Bu analizin amacı, kötücül niyetli isteğin erişim iznine sahip olmadan engellenmesini esas alınır. Şayet bu istek, bir şekilde güvenlik kırıcılar sayesinde erişim iznine sahip olsa bile bu isteğin kötücül olduğunu tespit etmek güvenlik analizinin esaslarından birisini teşkil eder. Kimlik denetleme yöntemleri UPA ve mikroservisler için literatürde genel olarak üç temel ana başlık altında toplanmıştır. Bunlardan ilki en klasik ve günümüzde en çok kullanılan yöntemlerin başında gelen kullanıcı adı ve şifre tabanlı kimlik denetlemedir. İkinci olarak, son yıllarda ortaya çıkan ve kullanıcıların güvenliğini biraz daha artırmayı hedefleyen bilet (token) tabanlı kimlik denetlemedir. Son olarak sertifika tabanlı kimlik denetimi olmak üzere üç temel başlıkta incelenmiştir.

2.5.1. Kullanıcı Adı ve Şifre Tabanlı Kimlik Denetleme

Şifre doğrulaması en çok kullanılan yöntemlerin başında gelmektedir. Kullanıcı adı ve şifre bilgileri sistemde saklanır ve daha sonra sisteme erişilmek istenildiğinde bu bilgiler teyit edilerek kimlik denetleme gerçekleşir. Son yıllarda bu doğrulama yöntemine ek olarak birçok ek özellik getirildi. Bunlardan biri de çift yönlü doğrulama (two-factor authentication) yöntemidir. Ancak tüm bu ek özelliklere rağmen sözlük ataklarına (dictionary attack) karşı savunması yeterli değildir [13].

2.5.2. Bilet (Token) Tabanlı Kimlik Denetleme

Bilet tabanlı kimlik denetleme felsefesi, istemci ve sunucu arasında istek yapılan çağrı üzerinde tekil bir anahtar üzerinden doğrulamayı yapmayı esas alır. Bilet tabanlı kimlik denetlemeyi esas alarak geliştirilmiş birçok yöntem mevcuttur.

2.5.2.1. Açık Kimlik Denetleme 2.0 (OAuth2.0)

RFC 6749 standardı ile belirlenmiş bir protokoldür. AKD (Açık Kimlik Denetleme) başta Google, Facebook, Twitter gibi büyük şirketlerin kullanım imkânı sunduğu çok yaygın olarak kullanılan bilet tabanlı bir güvenlik protokolüdür. Elde

edilen erişim bileti (accesstoken) ile sisteme girişe izin verilir. Belirli periyotlarla bu token yenilenmeye zorlanır ve sonrasında yenileme bileti (refreshtoken) ile yeni bilet (token) elde edilir. Sistemin kritiklik derecesine göre yenilenme periyodu belirlenir. AKD 2.0 protokolünün çalışma şekli aşağıdaki gibidir.

- i) Uygulama kullanıcıdan hesabının tamamına ya da bir kısmına erişebilmek için izin ister.
- ii) Eğer kullanıcı bu isteği onaylarsa uygulama bir yetki izni (authorization grant) elde eder.
- iii) Uygulama yetkilendirme (authorization) sunucusuna kendi bilgilerini ve almış olduğu yetki iznini (authorization grant) göndererek sunucudan erişim bileti isteğinde bulunur.
- iv) Eğer uygulama bilgileri doğruysa ve yetki izni (authorization grant) geçerli ise, yetkilendirme (authorization) sunucusu uygulamaya bir erişim bileti (access token) sağlar.
- v) Uygulama almış olduğu erişim bileti ile birlikte içerik (resource) sunucusuna erişmek istediği içerikle ilgili istek gönderir.
- vi) Eğer uygulamanın göndermiş olduğu accesstoken geçerli ise, içerik (resource) sunucusu, ilgili içeriği uygulamaya geri gönderir.



Şekil 2.6: AKD 2.0 (OAuth2.0) Protokol Akışı.

2.5.2.2. ÇOA (Çoklu Oturum Açma)

Çoklu Oturum Açma (Single Sign-On) vasıtasıyla bir kullanıcı kimliğiyle oturum başlatılarak (Signing in) Şekil 2.7’te gösterildiği gibi çok sayıda uygulamaya/servise erişim sağlanabilmektedir. Yetkilendirme (authorization) sürecinin merkezinde kimlik doğrulama (authentication) işlemi için kullanılan bir ÇOA sunucusu yer alır. İstemci bir kullanıcı adı ve şifresi ile ÇOA sunucusuna yönlendirilir, bilgilerin doğrulanmasının ardından oturum başlatılır ve bir oturum anahtarı (session token) oluşturulur.



Şekil 2.7: ÇOA Protokolü.

Oluşturulan bu anahtar ilişkili diğer uygulamalar veya servislerle ortak bir şekilde kullanılabilir. İlişkili uygulama kendisine iletilen anahtar (token) bilgisini kontrol eder. Doğrulama bir sorun olması durumunda işlem tekrarlanır. Doğrulama başarılı ise giriş yetkisi sağlanır. Bu şekilde aynı kullanıcı adı ve şifresiyle pek çok uygulama ve servise güvenli bir şekilde erişim sağlanabilmektedir [18].

2.5.2.3. JWB (Json Web Bileti)

RFC 7519 standardı ile belirlenmiş bir protokoldür. Haberleşen iki veya daha fazla sistem (Web, Mobil, Bulut vb.) arasında kullanıcı doğrulama, kullanıcı tanıma, veri bütünlüğünü ve bilgi güvenliğini koruma gibi noktalarda kullanılmaktadır. JSON tabanlı olması hız açısından büyük avantaj sağlamaktadır. Ayrıca

- i) URL üzerinde taşınabilmesi
- ii) Web çerezleri kullanma zorunluluğu olmaması
- iii) Siteler arası istek sahteciliği (CSRF) ataklarına karşı daha kapalı olması
- iv) Hızlı doğrulama yapılabilmesi
- v) Kolay ölçeklenebilir olması
- vi) Web uygulamaları açısından HTTP oturum (session) gerekmemesi, durum bağımsız (stateless) kullanıma uygun olması
- vii) Veri bütünlüğünü sağlaması

gibi bir çok avantajı bulunmaktadır.

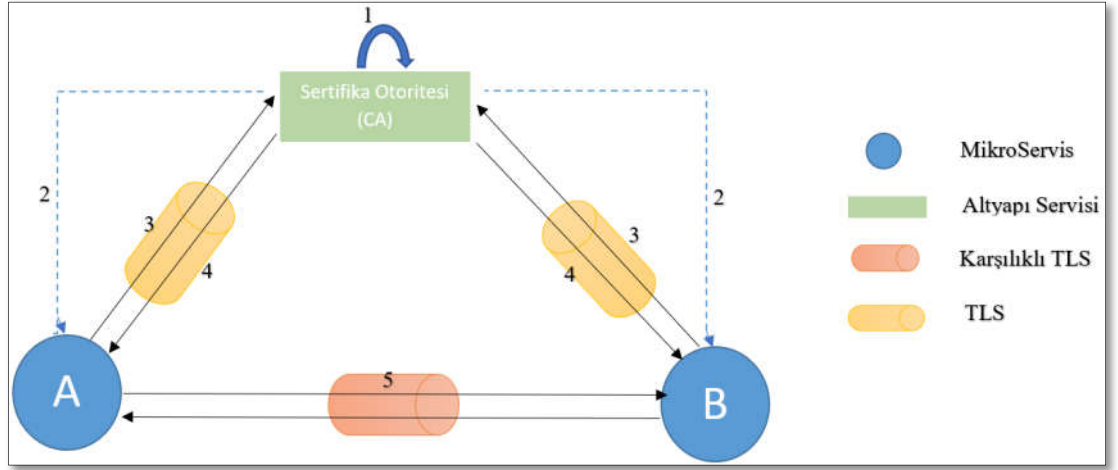
2.5.2.4. AKO (Açık Kimlik Oturumu)

AKD 2.0 (Auth2.0) protokolü üzerine kurulmuş basit bir kimlik denetleme katmanıdır. Client uygulamaların için bir yetkilendirme sunucusu tarafından son kullanıcı kimlik doğrulaması yapmasının yanı sıra TDT benzeri bir yapı ile son kullanıcının basit profil bilgilerini elde etmesini sağlar. Açık kimlik oturumu bağlantısı önceki versiyonlarına göre UPA dostu bir yapı sağlar. AKOB, AKD 2.0 yetkilendirme kodu yapısını genişleten bir eklentidir [19].

AKOB işlemleri basit tutar. AKD 2.0 üzerine kurulmuştur ve bilet (token) kullanımını kolaylaştırır. AKOB web uygulamalar, mobil uygulamalar, tek sayfa uygulamalar (Single Page Application) ve sunucu taraflı uygulamalar ile entegre olabilmektedir.

2.5.3. Sertifika Tabanlı Kimlik Denetleme

Diğer bir kimlik denetleme yöntemi ise mTLS (Mutual Transport Layer Security) yani karşılıklı TLS sertifikası kullanmaktır. Örneğin Docker Swarm dağıtık sisteme olarak tanıyan ve mikroservislerin yönetimini kolaylaştıran bir uygulamadır. Docker Swarm [14] da ki bütün düğümler tarafından mTLS kullanılarak düğümler arası network trafiği şifrelenerek kimlik denetimi uygulanmaktadır. Docker Swarm otomatik olarak tüm düğümlere kendi Açık Anahtar Altyapısı (Public Key Infrastructure)’ni yerleştirerek [15] düğümlerin kimliğinin oluşmasını sağlar. Şekil 2.8’te mTLS sisteminin çalışma prensibi açıklanmıştır.



Şekil 2.8: MTLS Çalışma Sistemi [10].

- i) Birinci adımda CA(Sertifika Otoritesi) servisi imzalanmış kök sertifikasını oluşturur ve gizli olarak paylaşır.
- ii) İkinci adımda sertifikanın kriptolanmış hash'i CA'den dışarı aktarılır ve otomatik veya manuel olarak A ve B servislerine sağlanır.
- iii) Üçüncü adımda A ve B servisleriyle CA arasında tek yönlü bir TLS bağlantısı kurulur ve daha önce sağlanan hash ile A veya B servisi şayet doğru bir CSR (Certificate Signing Request) gönderirse CA sertifikayı doğrular.
- iv) Dördüncü adımda alınan CSR nin başarılı doğrulanmasından sonra CA servisi yeni sertifikayı imzalar ve geri gönderir.
- v) Beşinci adımda A ve B servisleri MTLS üzerinden haberleşirler.

2.5.4. Elektronik İmza Tabanlı Kimlik Denetleme

E- imza veya diğer adıyla sayısal imza, elektronik ortamda oluşturulan bilgilerin imzalanmasında kullanılan ve elektronik işlemlerde gönderilen bilginin gönderim sırasında değişmediğini, gönderen kişiye ait olduğunu ve inkâr edilemeyeceğini garantileyen açık anahtar şifreleme yapısına sahip bir sistemdir [12]. CAdES (CMS Advance Electronic Signature), XAdES (XML Advance Electronic Signature), PAdES (PDF Advance Electronic Signature) olmak üzere üç farklı dosya uzantısı imzalama tipi vardır [17].

CAdES elektronik imza formatı ile byte array (byte[]) yapısındaki tüm veriler imzalanabilmektedir. Böylece mikroservis veya UPA'lar arası serileştirilen (serializable) tüm veriler bu yöntem üzerinden imzalanabilmektedir.

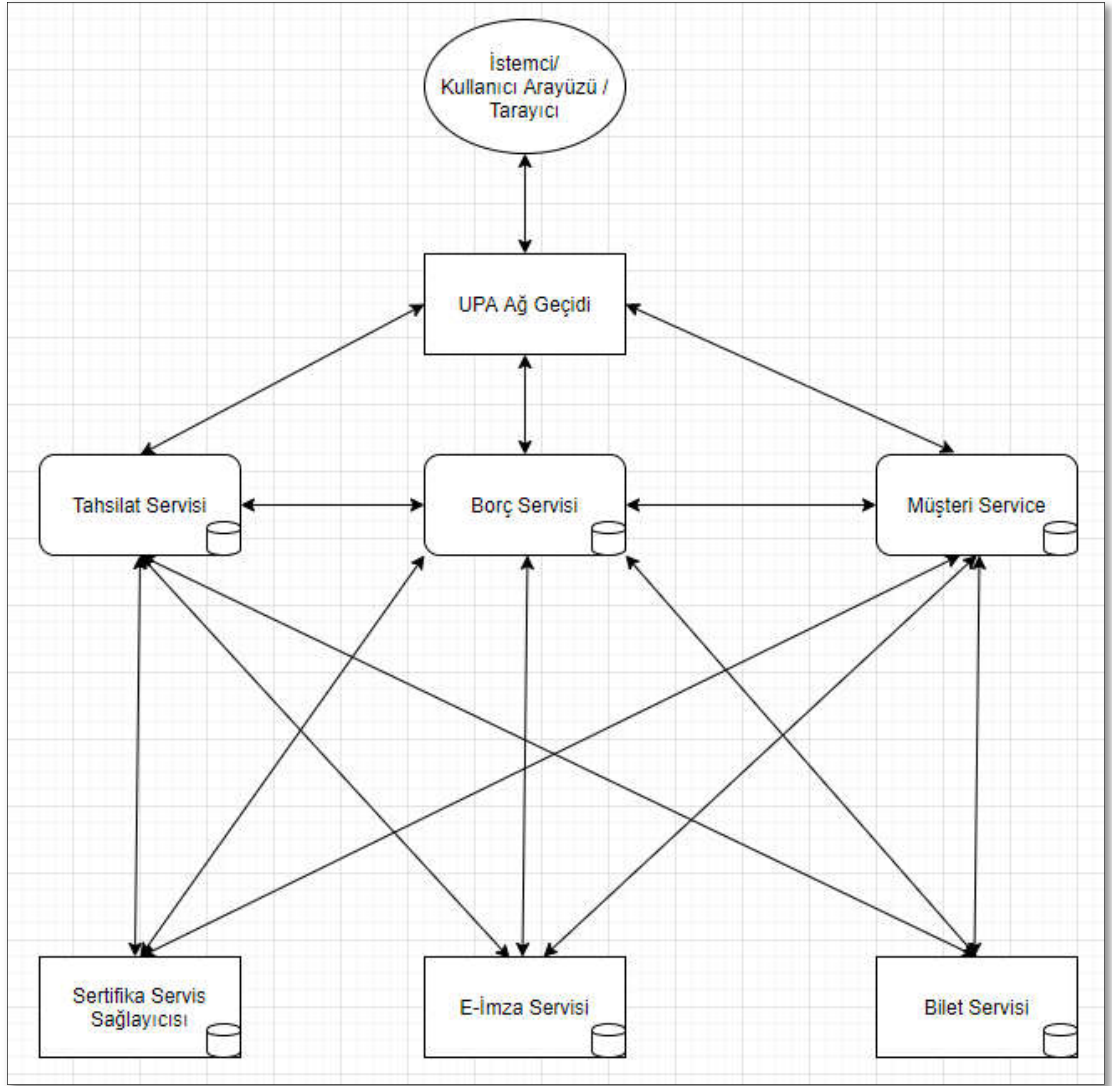
CAdES elektronik imza formatının dokuz adet alt çeşidi bulunmaktadır.

- i) Basic Electronic Signature (CAdES-BES)
- ii) Explicit Policy-based Electronic Signature (CAdES-EPES)
- iii) Electronic Signature with Time (CAdES-T)
- iv) Electronic Signature with Complete Validation Data References(CAdES-C)
- v) Extended Electronic Signature with Time Type 1 (CAdES-X Type 1)
- vi) Extended Electronic Signature with Time Type 2 (CAdES-X Type 2)
- vii) Extended Electronic Signature with Time (CAdES-X Long Type 1 or 2)
- viii) Archival Electronic Signature (CAdES-A)
- ix) Long-Term Electronic Signature (CAdES-LT)

CAdES alt imza türleri sayesinde atılan e-imzaya zaman damgası, arşiv veya türevi özellikler eklenerek imzalanan veriyi çok daha güvenli ve kontrol adımlarını artırabilme imkânına sahip olunmaktadır.

3. KARMA YÖNTEM GÜVENLİK TASARIMI

Karma yöntem güvenlik tasarımında (KYGT), sertifika, bilet ve e-imza tabanlı çözümleri bir arada uygulanarak her üç güvenlik yönteminin de güçlü olan yönlerinden istifade edilerek sistemin olası saldırılar karşısında azami düzeyde korunması amaçlanmaktadır. Bu sistemin tasarımındaki amaç, performanstan ziyade sistem güvenliğinin artırılarak olası saldırılardan asgari düzeyde etkilenmektir.



Şekil 3.1: Karma Yöntem Güvenlik Tasarımı.

Karma yöntem güvenlik tasarımının ilk aşaması webservis veya mikroservisler arası iletişimin mTLS tabanlı olması tasarlanarak, ağ üzerindeki trafiğin https üzerinden şifreli bir şekilde iletilmesi hedeflenmektedir. Böylece herhangi bir

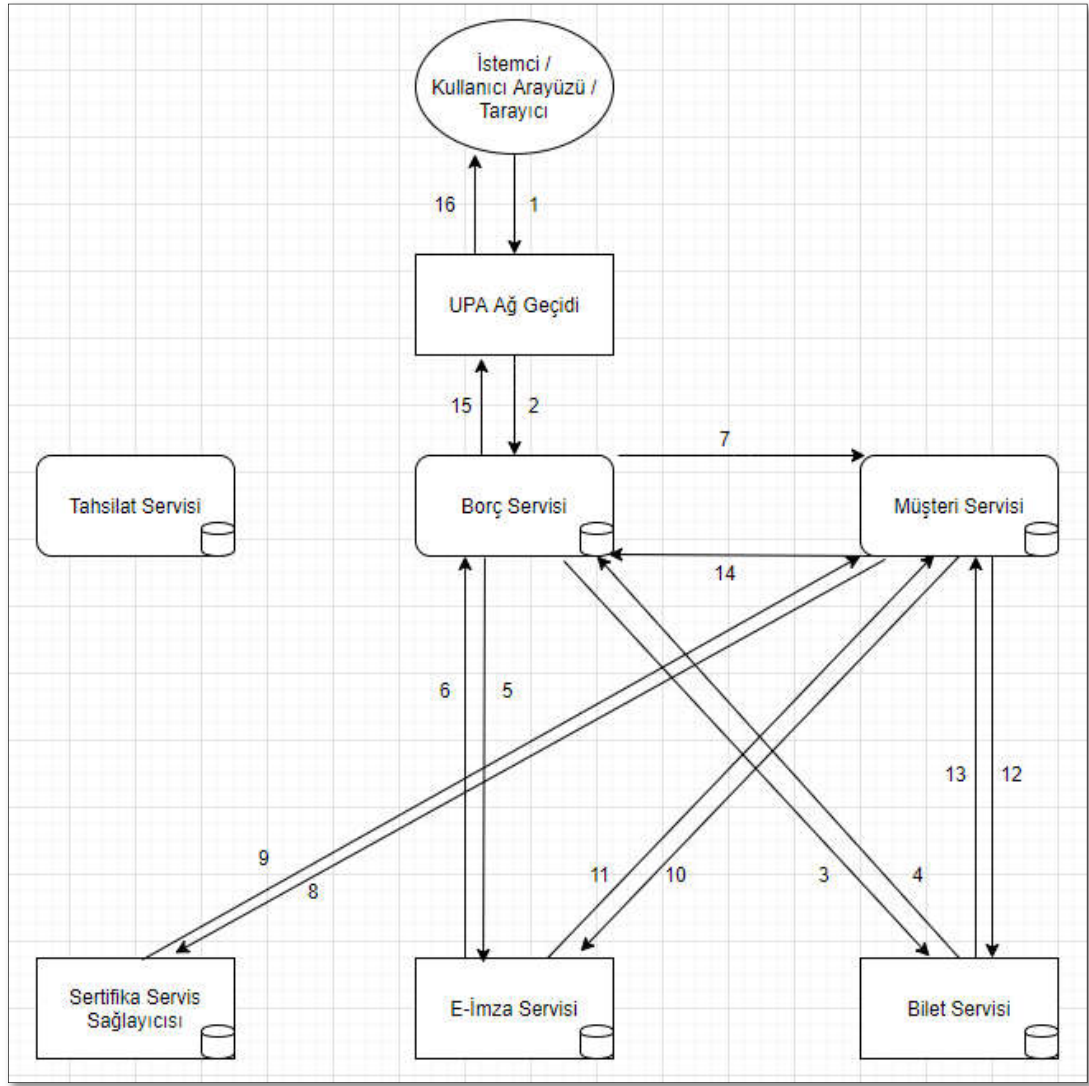
saldırgan ağ trafiğini dinlese bile veriyi görebilmesi için sertifikayı taklit etmesi gerekecektir. Docker Swarm [8] ve Netflix [9] bu yöntemi kullanarak mikroservisler arası güvenliği sağlamaktadırlar.

Önerilen güvenlik çözümünün ikinci aşaması ise servisler arası iletişimde AKD 2.0 güvenlik yöntemi kullanılarak, bilet tabanlı bir kimlik denetleme kontrolü sağlanmaktadır. Böylece ilk adımda uygulanmış olan TLS güvenlik yönteminin bir açığı olan ortadaki adam saldırısı [10], [16] gibi oluşabilecek bir zayıflığın önüne geçilmektedir.

Üçüncü ve son adım olarak ise servisler arası iletilecek veri bütünlükte tipte CAdES-BES [11] formatında imzalanarak, yukarıdaki tüm güvenlik yöntemlerini aşan bir saldırı ile karşı karşıya kalırsa bile e-imzayı çözmeden verinin okunamayacak bir şekilde korunması sağlanmaktadır.

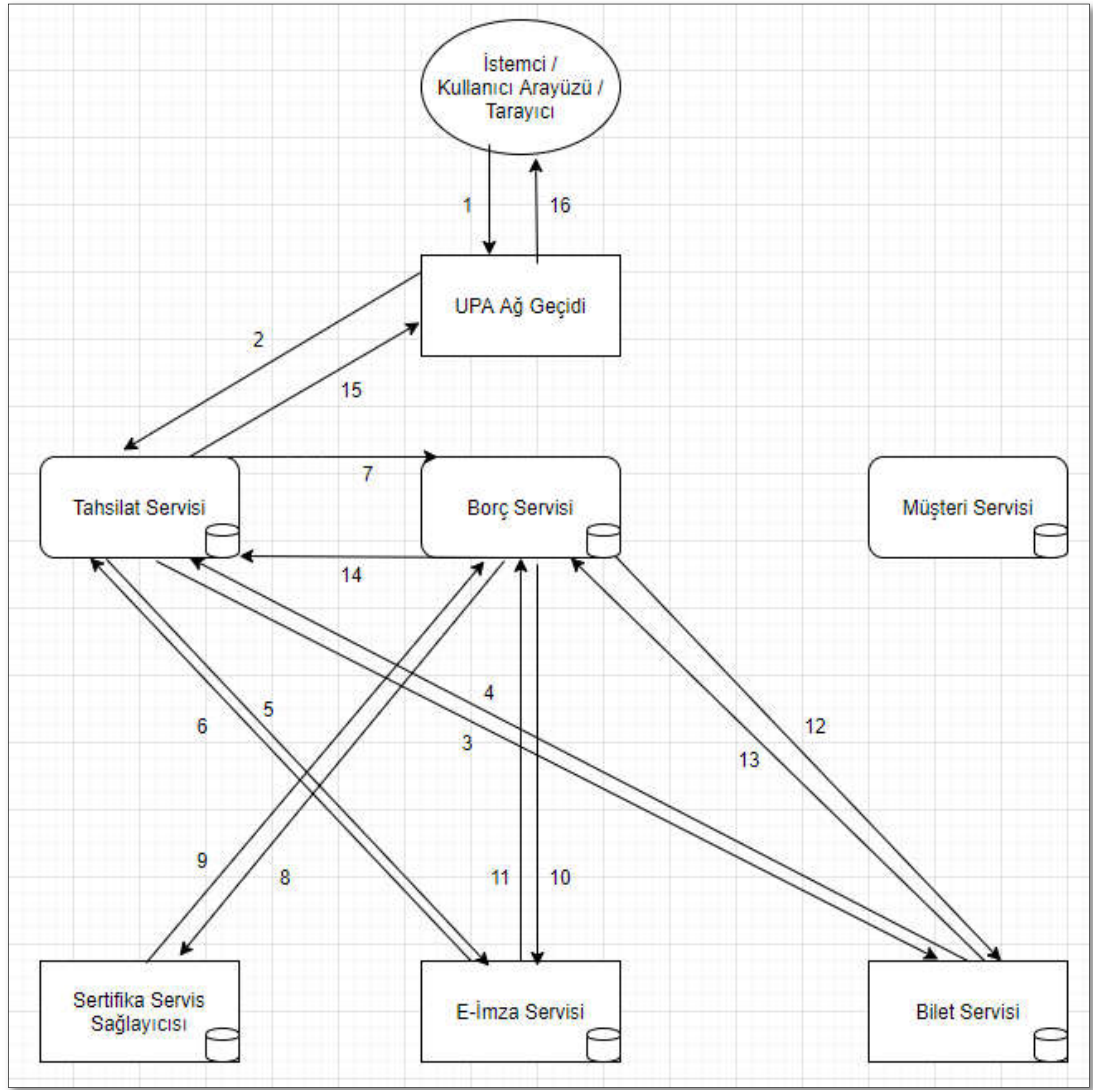
Şekil 3.1’de gösterilen, örnek olarak tasarlanmış olan bir fatura ödeme sistemi üzerinde karma yöntem güvenlik tasarımı (KYGT) uygulaması gerçekleştirilmiştir. Bu uygulamada altı adet servis bulunmaktadır. Bu servisler mantıksal olarak sınıflandırıldığında üç tanesi (Sertifika Servis Sağlayıcısı, Bilet Servisi, E-İmza Servisi) güvenliği artırmak için tasarlanmışken, diğer üç tanesi (Tahsilat Servisi, Borç Servisi, Müşteri Servisi) ise fatura ödeme işlemine ait iş ihtiyacını karşılamak üzere tasarlanmıştır.

Gerçekleştirilen senaryoda, bir kullanıcı fatura borcunu ödemek istediğinde, ekrandan abone numarası ve TCKN/VKN bilgisi girilerek borç sorgulama işlemi yapılmış ve sonrasında ekrana gelen ilgili borç tutarını seçildikten sonra kredi kartı bilgilerini girerek ödeme işlemi gerçekleştirilmiştir.



Şekil 3.2: Borç Sorgulama Örnek Akışı.

Şekil 3.2’de akışı gösterilen borç sorgulama işlemi için kullanıcı abone numarası ve TCKN/VKN verisi ile sorgulama yapıldığında, istek ilk olarak UPA Ağ Geçidi kanalı aracılığı ile Borç servisine gelir. Bu servis üzerinden girilen TCKN/VKN’e ait bir müşteri olup olmadığı kontrolü için müşteri servisiyle iletişim kurulması gerekmektedir. Müşteri servisi çağırılmadan önce borç servisi tarafından SSL sertifikasının “sertifika servis sağlayıcısı” tarafından temin edildiğini biliyoruz. Sonrasında bilet alma işlemi için bilet servisine giderek accesstoken bilgisini elde edip, son olarak e-imza servisine giderek yapılmış olan isteği CAdES-BES formatında e-imza ile imzalayarak müşteri servisi çağırılmıştır.



Şekil 3.3: Tahsilat İşlemi Örnek Akışı.

Müşteri servisi ise gelen istek karşılanarak, ağ katmanında SSL sertifikasının geçerlilik kontrolü yapıldıktan sonra yapılan isteğin CADES-BES formatlı e-imzalı halinin imzasını çözülerek isteğin e-imzasının çözülmüş halinin ortaya çıkması sağlanmıştır. Sonrasında ise bu isteğin içerisinde bulunan bilet bilgisinin doğruluğunu ve geçerliliğini kontrol etmek için bilet servisi çağrılmıştır. Bu doğrulamalarda başarıyla tamamlandıktan sonra servise cevap olarak müşteri bilgisi dönülecektir. Şayet müşteriye ait borç bilgisi mevcutsa istemciye bu bilgi dönülerek, borcunu görmesi sağlanmıştır.

Müşteri ekranda borcunu gördükten sonra faturasını ödemek isterse “Tahsil Et” butonuna tıklayarak ödeme işleminin gerçekleşmesi için gereken akış başlatılmıştır. Ödeme işlemi akışı için, borç sorgulama işleminde olduğu gibi “UPA Ağ Geçidi”

aracılığı ile ilk olarak tahsilat servisine gelinmiş ve burada işlem öncesi borç doğrulaması yapmak için borç servisine istekte bulunulmuştur.

İstekte bulunmadan önce borç sorgulama servisinde olduğu gibi güvenlik adımlarını uygulamak için önce bilet servisine ve sonrasında e-imza servisine istekte bulunarak ilgili isteğimizi güvenlik sarmalı ile sararak borç servisine erişiyoruz. Borç servisine bu istek ulaştığında SSL el sıkışmasını sağladıktan sonra güvenlik sarmalını çözerek isteğin ham haline ulaşmak için öncelikle e-imza servisine giderek e-imza'nın doğrulanmış ve çözülmüş halini elde ettikten sonra bilet servisine giderek istek üzerindeki biletin geçerlilik kontrolü gerçekleştirilmektedir.

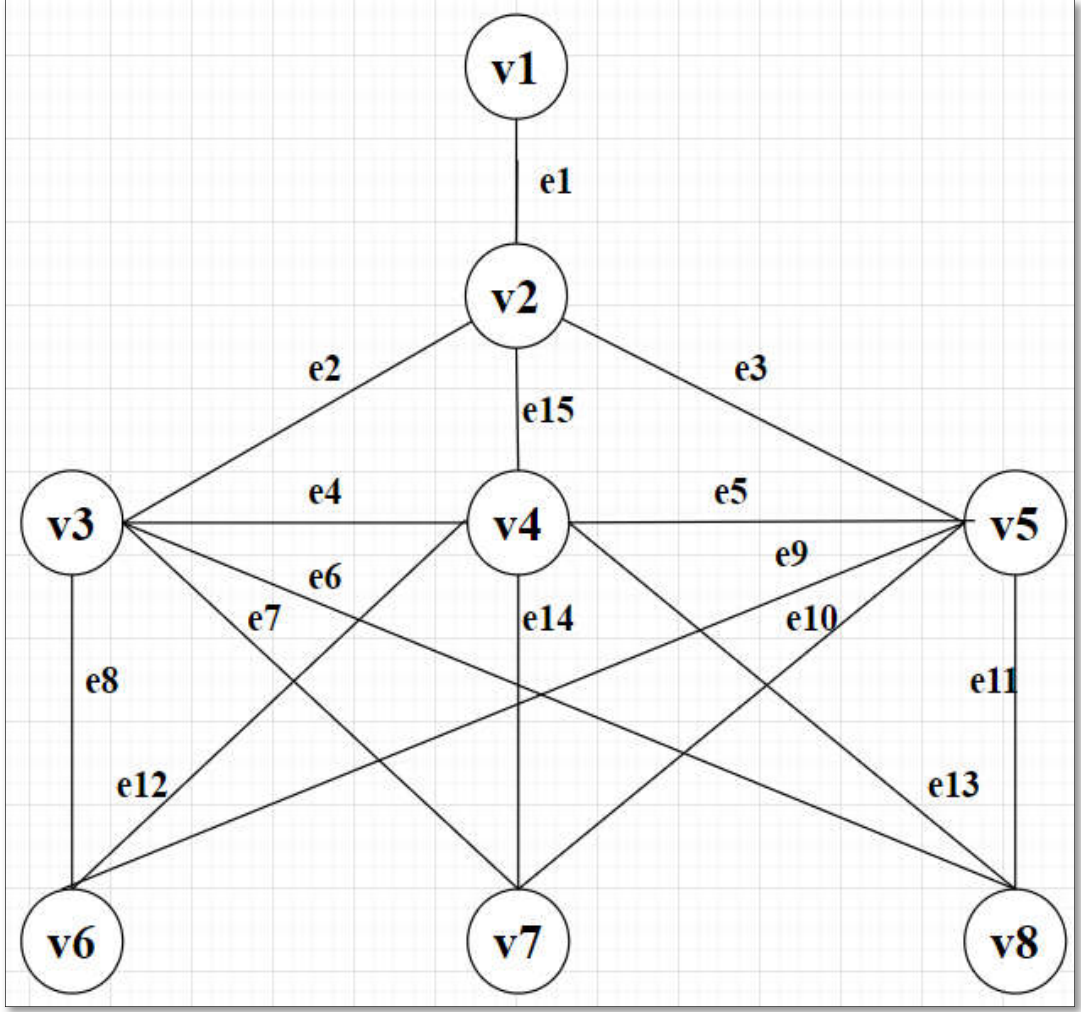
Yapılan istek, yukarıdaki aşamaları başarıyla geçtikten sonra tahsilat servisine borçla alakalı bilgilerin doğru veya yanlış olduğu cevabı dönülerek akışa devam edilmektedir. Şayet borç bilgileri yanlış ise ödeme işlemine izin verilmemektedir. Şayet borç bilgileri doğru ise ilgili kredi kartından provizyon alıp, muhasebe işlemi yapılarak istemciye işlemin başarıyla tamamlandığı bilgisini dönülmektedir.

3.1 Matematiksel Model

Ağ bağlantılı veri, ağ üzerinden elde edilen veri olarak tanımlanır. Her nesnenin diğer nesnelerden bağımsız olarak belli bir öznitelik kümesi ile tanımlandığı geleneksel veriden farklı olarak, ağ verisindeki varlıklar birbirleriyle farklı bağlantı türleri ile korelasyon içerisindeki bir veri türüdür. Ağ bağlantılı veri, ağ içerisindeki her nesneyi tanımlayan öznitelik kümelerinden ve nesneler arasındaki ilişkiyi tanımlayan bağlantı yapısından oluşmaktadır. Bağlantı yapısı daha çok kompleks ağ yapısı olarak modellenebilmektedir. Ağın topolojisi üzerine geliştirilen matematiksel modellere, bir de ağ içerisinde paylaşılan anlamsal bilgiler eşlik eder. KYGT'nin topolojisi matematiksel model üzerinde ifade edildiğinde, çizge (graph) teorisinin kullanılması uygun olmaktadır. G çizgesi için; her bir mikroservis bir V düğümüne, mikroservisler arası iletişim E bağlantısına, mikroservislerin icra ettiği görev ise F fonksiyonuna karşılık gelmektedir.

- i) $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ düğümleri,
- ii) $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}\}$ bağlantıları,
- iii) $F = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$ fonksiyonları

için $G = (V, E, F)$ çizgesini meydana getirmektedir.



Şekil 3.4: KYGT Çizgesi.

Şekil 3.4 görüldüğü üzere G çizgesi KYGT'yi temsil etmektedir. G çizgesindeki $|E|$ kerte sayısı, yani düğüme çakışan ayrıtların sayısını ifade etmek gerekirse, d düğüme bağlı kerte olmak üzere

$$|E| = \frac{\sum_i d_i}{2} \quad (3.1)$$

$$d_1 = 1, d_2 = 4, d_3 = 5, d_4 = 6, d_5 = 5, d_6 = 3, d_7 = 3, d_8 = 3$$

$|E| = \frac{30}{2} = 15$ kerte, yani bağlantı sayısını ifade etmektedir. d düğümlerinin derecelerine göre incelendiğinde v_4 düğümü en fazla bağlantı sayısına sahiptir. Bu

duruma göre v_4 düğümünün temsil ettiği “Borç Servisi” çizge üzerinde 6 bağlantı sayısı ile kritiklik derecesi en fazla olan servis olarak görülmektedir. Bu düğüm v_3 ve v_5 düğümleri arasında köprü vazifesi ifa ettiğinden sistem üzerinde performans ve güvenlik olarak önemli bir yer tutmaktadır. G çizgesi üzerinde v düğümleri aşağıdaki fonksiyonları icra etmektedir.

- i) $f_1 = \{\text{İstemci}\}$,
- ii) $f_2 = \{\text{UPA Ağ Geçidi}\}$,
- iii) $f_3 = \{\text{Borç Servisi}\}$,
- iv) $f_4 = \{\text{Tahsilat Servisi}\}$,
- v) $f_5 = \{\text{Müşteri Servisi}\}$,
- vi) $f_6 = \{\text{Sertifika Sağlayıcısı Servisi}\}$,
- vii) $f_7 = \{\text{Elektronik İmza Servisi}\}$,
- viii) $f_8 = \{\text{Bilet Servisi}\}$,

Bu fonksiyonlardan f_3, f_4, f_5 iş parçacıklarını temsil ederken f_6, f_7, f_8 güvenlik altyapısı ile alakalı servisleri temsil etmektedir. Gerçekleştirilecek olan işlem ve senaryoya göre f_3, f_4, f_5 fonksiyonlarının sayısı ve çeşidi artırılabilirken; güvenlik ve altyapıyı temsil eden f_6, f_7, f_8 fonksiyonalliklerinde bir değişme olmayacaktır.

Düğüm sayısı, servislerin üzerinde kritiklik seviyesini temsil etme noktasında sistem üzerinde bilgi sağlıyor olsa da servis tipinin altyapı ya da iş parçacığını temsil etmesi noktasında fonksiyonallitenin büyük önemi vardır. Sadece düğüm sayısına bakarak servisin önem derecesi üzerinde fikir sahibi olmak yanlış sonuçlar doğuracaktır. Güvenlik altyapısında çalışmayan bir fonksiyon sistemin güvenliği üzerinde zafiyet oluştururken, iş parçacığında çalışmayan bir fonksiyon sistemin hizmet verememesine neden olmaktadır.

4. DENEYSEL SONUÇLAR ve DEĞERLENDİRME

Mikroservis mimarisi ile tasarlanan karma yöntem güvenlik tasarımı ile alakalı iki temel performans analizi yapılmıştır. Bu analizlerden ilki karma yöntem güvenlik tasarımında önerilen mTLS, biletleme ve e-imza güvenlik yöntemlerinin ayrı ayrı ve bir arada kullanılması gibi farklı varyasyonlarıyla yapılmıştır. İkinci olarak yapılan analiz ise mikroservis mimarisinin kendisine alternatif olarak ortaya çıktığı yekpare mimari ile karşılaştırılması ve ortaya çıkan temel farkın mimari odaklı olup olmadığının tespitidir.

4.1. KYGT Karşılaştırmalı Performans Analizi

Karma yöntem güvenlik tasarımında önerilen çözüm yöntemiyle her ne kadar performans açısından bir çözüm yöntemi önerilmemiş olsa da gerçek dünyada bu çözümün kullanılabilirliği açısından yapılan performans ölçümlerinin önemli bir rolü bulunmaktadır.

Karma yöntem güvenlik tasarımıımızın performans testi için öncelikle gerçek bir uygulama geliştirme ihtiyacı hasıl olmuştur. Uygulamadaki mikroservisler:

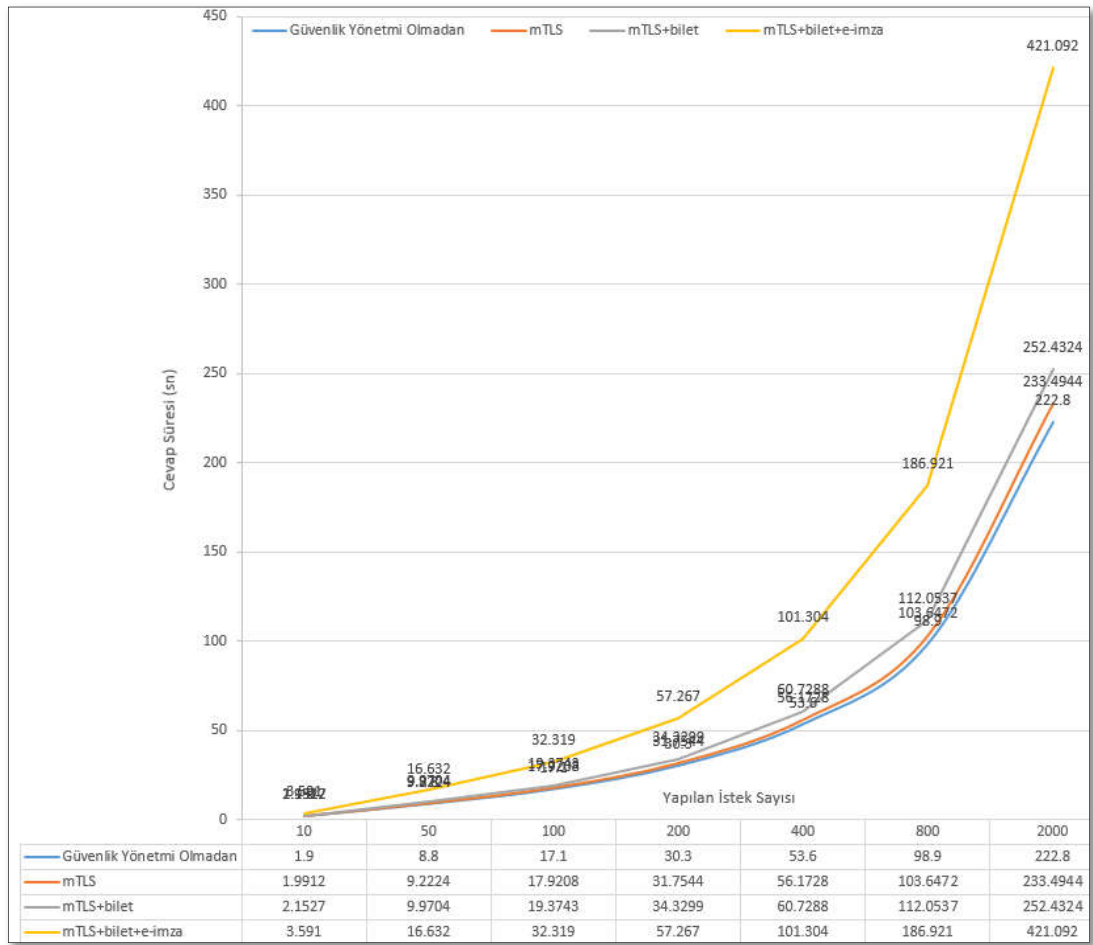
- i) UPA Ağ Geçidi: İstemci isteklerinin ilk olarak karşılandığı, merkezi giriş noktası ana servisedir.
- ii) Tahsilat Servisi: Finansal hizmetin gerçekleştiği, müşterinin hesabının uygun olup olmadığı, yapılan işlemin türüne göre kredi kartı provizyonu, hesaplar arası para geçişi gibi muhasebe işlemlerinin gerçekleştiği servistir.
- iii) Borç Sorgulama Servisi: İstemciden gelen istekteki verilere ait borç bilgisini sağlayan ve sonrasında müşteri ile ilgili kontrollerin yapıldığı servistir.
- iv) Müşteri Servisi: İstemciden gelen isteklere ait verilerin kuruma ait müşteri olup olmadığı kontrolünün yapıldığı servistir.
- v) Sertifika Servisi Sağlayıcı: Fatura ödeme servislerine (Tahsilat, Borç Sorgulama, Müşteri) ait güncel sertifika bilgilerini üreten ve ilgili servisler ile paylaşan altyapısal servistir.

- vi) Bilet Servisi: Fatura ödeme servisleri (Tahsilat, Borç, Müşteri) için gerekli olan bilet (token) bilgisini üreten ve aynı zamanda istek sırasında doğrulama işlemi sağlayan altyapı servsidir.
- vii) Elektronik İmza Servisi: Fatura ödeme servisleri (Tahsilat, Borç, Müşteri) arası yapılan istek (request) için gerekli olan e-imza işlemine ait şifreleme ve şifre çözme işlemlerinin gerçekleştiği altyapı servsidir.

Bu deney, i7-4700MQ 2.4GHZ intel marka işlemci, 16 GB bellek içeren, 128 GB SSD sabit disk ve windows10 64 bit lenova marka bilgisayar kullanılarak Visual Studio 2017 geliştirme ortamında C# dilinde kodlanarak gerçekleştirilmiştir.

Yapılan deneyde dört farklı güvenlik kombinasyonu oluşturulmuş, performans karşılaştırması yapılmıştır. Güvenlik yöntemi olmadan, mTLS, mTLS+bilet ve mTLS+bilet+e-imza güvenlik yöntemleri performansları birlikte incelenmiştir. Test makinesi üzerinden ilk olarak 10 adet istek gönderilmiş olup, daha sonra yapılan istek sayısı 50, 100, 200, 400, 800, 2000 şekilde artırılarak karşılaştırmalı olarak bu yöntemlerin performansa etkisi görülmüştür. Yapılan istekler aynı makine üzerinden eş zamanlı bir şekilde thread yapısıyla gönderilmiştir.

Test işlemi sırasında fatura ödeme senaryosu hep başarılı bir şekilde gerçekleşmiştir. Şayet bu işlemlerin gerçekleşmesi esnasında, banka provizyonu alımı sırasında veya hesapta yeterli bakiye olmaması gibi durumlardan kaynaklı hata alması durumu düşünüldüğünde bu performansa olumsuz olarak yansıyacaktır.



Şekil 4.1: KYGT Performans Analizi.

Şekil 4.4'te gösterilen karşılaştırmalı performans analizine göre güvenlik kontrolü olmadan yapılan istekler ile sadece mTLS kullanılarak yapılan istekler arasında %4.8 olan bir performans azalma etkisi olduğu görülmektedir. mTLS ve bilet tabanlı yöntemler birlikte uygulandığında ise %13.3'lük bir performans azalma etkisi olduğu görülmektedir. mTLS, bilet ve e-imza yöntemlerini birlikte uyguladığımızda ise %18.9'luk bir performans kaybı söz konusu olmaktadır.

Ortaya çıkan performans analizi sonuçlarına göre elektronik imzanın cevap süresini etkileyen, performans üzerinde ciddi bir etkisi olduğu görülmektedir. mTLS ve bilet tabanlı güvenlik yönteminin etkisi e-imza'ya göre %13.3 gibi bir oranla daha kabul edilebilir düzeyde görülmektedir. Tasarlanan sistemin kritiklik derecesine göre bu performans kaybının göze alınmasına karar verilmelidir.

Güvenlik açısından sonuçlar değerlendirildiğinde üç farklı yöntemi bir araya getirerek bunların birlikte kabul edilebilir bir performansta çalışabilir olduğu doğrulanmıştır. Böylece bu yöntemlerin (sertifika, bilet, e-imza) ayrı ayrı kullanılması

durumunda oluşabilecek zafiyetlerin önüne geçilmiştir. Sistemin performans ihtiyacına göre bu yöntemin sac ayağını oluşturan üç temel güvenlik çözümü ögesinden bir veya iki tanesini de bir araya getirmek suretiyle kullanımı mümkün olmaktadır.

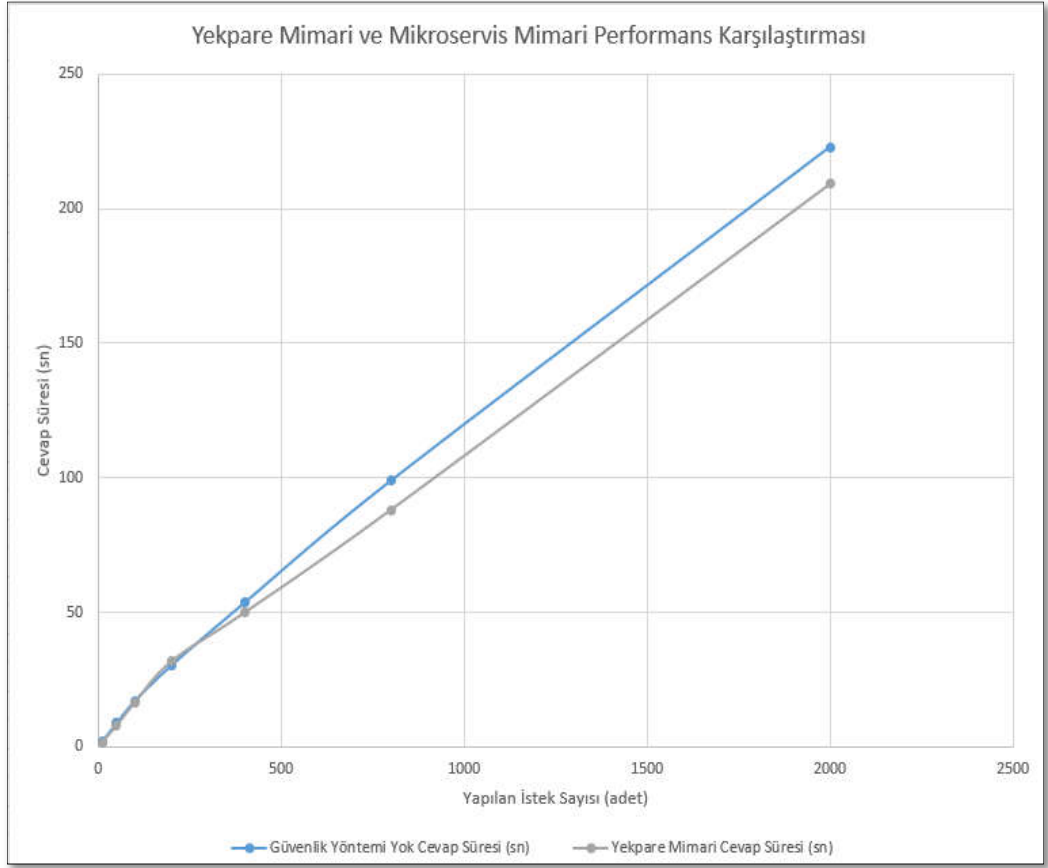
Ortaya çıkan sonuçlar, kullandığımız araçlar ve test ortamı açısından değerlendirilmesi gerekirse C# dili ve Microsoft teknolojileri ile çok basit bir servis mimarisi üzerinde bu test işlemi gerçekleştirilmiştir. Bu platform yerine farklı bir yazılım dili ve ortamı ile geliştirme işlemi gerçekleştirilerek ortaya çıkan performans tekrar değerlendirilebilir.

4.2. KYGT ve Yekpare Mimari Performans Karşılaştırması

Mikroservis mimarisinin yekpare mimariye alternatif olarak bazı avantajlarla yeni bir yaklaşım getirdiği bilinmektedir. KYGT’de güvenlik çözümü olarak, mikroservis mimarisinin kullandığı güvenlik yöntemlerinin performans üzerindeki etkisi bölüm 4.1.’de incelendi. Bu bölümde KYGT’nin sahip olduğu performansın, yekpare sistem ile karşılaştırılması yapılacaktır. Bu karşılaştırma sonrası mikroservis mimari yaklaşımının bütüncül olarak performans açısından yekpare sistemlere alternatif olup olamayacağı konusu hakkında bir fikir oluşacaktır.

Bölüm 3’te anlatılan fatura ödeme sistemi senaryosu için yekpare sistem ile benzer bir uygulama üzerinden aynı işlemler gerçekleştirilecek şekilde tasarım yapıldı.

Bu deney, i7-8665U 2.11 GHZ intel marka işlemci, 16 GB bellek içeren, 128 GB SSD sabit disk ve windows10 Enterprise 64 bit HP marka bilgisayar kullanılarak Visual Studio 2017 geliştirme ortamında C# dilinde kodlanarak gerçekleştirilmiştir.



Şekil 4.2: KYGT ve Yekpare Mimari Performans Analizi.

Test uygulaması konsol uygulaması üzerinde tasarlanmış bir istemci üzerinden, veri tabanı olarak Microsoft SQL kullanılarak geliştirilmiştir. Bu uygulama üzerinden ilk olarak 10 adet istek gönderilmiş olup, daha sonra yapılan istek sayısı 50, 100, 200, 400, 800, 2000 şeklinde artırılarak performans yansıması takip edilmiştir. Yapılan istekler aynı makine üzerinden eş zamanlı bir şekilde thread yapısıyla gönderilmiştir.

Şekil 4.5'te gösterilen performans analizi sonuçlarına göre yekpare mimari ile mikroservis mimarisi istek sayısı 500'e kadar ki kısım için birbirlerine çok yakın performans göstermiş olsalar da istek sayısı 500'ün üzerindeki performansları açısından yekpare mimari daha tercih edilebilir görülmektedir. Ancak bu deneyin sonuçları, mikroservis mimarisinin deney ortamında kâmil manada uygulanamamasından mütevellit farklı açılardan değerlendirilmelidir. Deney ortamında her bir servis aynı makine üzerinde, aynı işlemci ve bellek kullanılacak şekilde tasarlandığı için bu durumda performans açısından dar boğaz oluşması söz konusu olmaktadır. Bu servisler ayrı ve güçlü performansa sahip makinelerde kurularak farklı performans sonuçlarının ortaya çıkması mümkün görünmektedir.

5. SONUÇ VE GELECEK ÇALIŞMALAR

Mikroservis mimarisinin; nesnelerin interneti, servis odaklı mimari ve bulut tabanlı gelişen teknolojiler üzerinde kullanımı artmaktadır. Diğer nedenlerin yanı sıra, yekpare mimari ile tasarlanan yazılım sistemlerinin modernizasyonu mikroservis mimarisinin gelişmesi anlamında en önemli motivasyonlardan biridir. Yekpare sistemlerin modern mikroservislere dönüştürülmesi ve modernizasyonu konusunda kayda değer çalışmalar bulunmaktadır. Bu tez çalışmasında, mikroservis mimarisinin hem güvenli bir şekilde nasıl tasarlanacağı, hem de bu tasarım yapılırken tasarım da kullanılan her bir güvenlik çözümünün bütün sistem üzerindeki performansa nasıl etki edeceği ile alakalı sonuçlar gösterilmiştir. Ayrıca mikroservis mimarisinin güvenlik yöntemi uygulanmamış yalın hali ile yekpare mimari arasında yapılan performans karşılaştırmasıyla bu çözümün sonuçları ortaya koyulmuştur. Mikroservis mimarisi yekpare mimariye alternatif olarak ortaya çıkmış ve performans olarak iddialı bir yaklaşım olsa da vaka çalışmamızda bu durumun performans açısından yekpare mimari lehine olduğu tespit edilmiştir.

Çağımızda her geçen gün artan siber saldırılardan en çok dışarı açık sistemler etkilenmektedir. Bu tez çalışmasında önerilen karma yöntem güvenlik tasarımının ana odağı uygulama ve yazılım güvenliğini sağlayarak, dışı açık sistemlerdeki zafiyetleri asgari düzeye indirmektir. Böylece sistem üzerindeki veri çalma, sahtekarlık, veri bozma gibi saldırı durumlarında güvenli bir sistem tasarımı önerilmektedir.

Karma yöntem güvenlik tasarımının bazı riskleri mevcuttur. Bunlar tasarımın üç ana ayağını oluşturan servislerin (Sertifika Servis Sağlayıcısı, Bilet Servisi, E-İmza Servisi) hizmet verememesi, hizmet vermesinin engellenmesi veya yanlış hizmet vermesi şeklinde özetlenebilmektedir. Böyle bir zafiyetin ortaya çıkması durumunda sistemdeki tüm servisler, bu hizmetler olmadan da yalın olarak çalışabilecek şekilde tasarlanmıştır. Örneğin CA servisi hizmet veremeyecek durumda olduğunda servise ait aynı hizmet geçici olarak http portu üzerinden hizmet verecek şekilde tasarlanmıştır. Ayrıca bilet ve e-imza güvenlik çözümlerine ait geçerlilik kontrolleri ise zafiyet giderilene kadar geçici bir şekilde kapalı olarak çalışabilmektedir.

Diğer bir husus ise saldırgan temel servislerin dışında bir zafiyet sonucu diğer servislerden herhangi birini ele geçirirse bile veri e-imzalı byte array (byte[]) olarak iletileceği ve accesstoken ile geçerli olacağı için herhangi bir veriyi görme şansı

bulunmamaktadır. Yapılan bu çalışma ile sistem ve ađ üzerinde performans açısından bir miktar yük oluşmuş olsa da bu ödünleşim sayesinde güvenliđin artırılmasını sağlanmıştır.

Gelecekte bu çalışmayı performans açısından geliştirmek için veya CAdES-T gibi daha gelişmiş zaman damgası içeren elektronik imza ile imzalayarak güvenliđi daha fazla artırma çalışmaları gerçekleştirilebilir. Bu çözüm performans açısından biraz daha maliyet getireceđi için bu çalışmanın kapsamına dahil edilmemiştir ancak sistemin çok daha güvenli olmak istenildiđi yerlerde bu çözüme başvurulabilir. Ayrıca gelecekte yapılabilecek çalışmalara ek olarak sistem üzerindeki ađ trafiđinin yoğunluk haritası çıkarılarak buradaki istekler üzerinden makine öğrenmesi tabanlı sahtekarlık tespiti yapabilecek bir sistem tasarımı yapılabilir.

KAYNAKLAR

- [1] Sun Y., Nanda S., Jaeger T., (2015), "Security-as-a-service for microservices-based cloud applications", IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 50-57, Vancouver, BC, Canada, 30 November.
- [2] Web 1, (2020), https://owasp.org/www-project-top-ten/2017/Top_10.html, (Eriřim Tarihi: 10/01/2020)
- [3] Göçer B. D., Bahtiyar S., (2019), "An Authorization Framework with OAuth for FinTech Servers," 4th International Conference on Computer Science and Engineering (UBMK), 536-542, Samsun, Turkey.
- [4] Rodgers, P., (2005), "Service-Oriented Development on NetKernel- Patterns, Processes & Products to Reduce System Complexity Web Services Edge", Cloud Computing Expo, 112–117. New York, USA, 12-13 November.
- [5] Harris, T. Marlow, S. Jones, S. Herlihy, M., (2005), "Composable memory transactions", Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming.
- [6] Web 2, (2020), <https://en.wikipedia.org/wiki/Microservices>, (Eriřim Tarihi: 19/03/2020)
- [7] Web 3, (2020), <https://microservices.io/articles/whoisusingmicroservices.html>, (Eriřim Tarihi: 20/03/2020)
- [8] Ahmadvand M., Ibrahim A., (2016), "Requirements Reconciliation for Scalable and Secure Microservice (De)composition", IEEE 24th International Requirements Engineering Conference Workshops (REW), 68-73, Beijing, China, 12-16 September.
- [9] Dahlstedt A. G., Persson A., (2005), "Requirements Interdependencies: State of the Art and Future Challenges", Springer Berlin Heidelberg, 95–116, Berlin, Germany.
- [10] Yarygina T., Bagge A., (2018), "Overcoming Security Challenges in Microservice Architecture", IEEE Symposium on Service-Oriented System Engineering, 11-20, Bamberg, Germany, 26-29 March.
- [11] Web 4, (2020), <https://medium.com/@akanesen/api-nedir-entegrasyonunasi%C4%B1l-yap%C4%B1l%C4%B1r-578f93578e46>, (Eriřim Tarihi: 29/03/2020)
- [12] Web 5, (2015), Building Microservices, https://www.nginx.com/wp-content/uploads/2015/01/Building_Microservices_Nginx.pdf. (Eriřim Tarihi: 29/03/2020)

- [13] Wang D., Wang, P., (2013), “Offline dictionary attack on password authentication schemes using smart cards.”, Proceedings of 16th Information Security Conference, 221-237, Dallas, Texas, USA, 13-15 November.
- [14] Web 6, (2020), <https://docs.docker.com/engine/release-notes/17.06/>, (Eriřim Tarihi: 21/09/2020)
- [15] Web 7, (2020), <https://youtu.be/7YPIsbz8Pig>, (Eriřim Tarihi: 02/07/2020)
- [16] Web 8, (2020), https://tr.wikipedia.org/wiki/Man-in-the-middle_sald%C4%B1r%C4%B1s%C4%B1
- [17] Web 9, (2020), <https://www.btk.gov.tr/uploads/pages/elektronik-imza-kullanım-profilleri-rehberi-5a33ff5b59f93.pdf> (Eriřim Tarihi: 04/06/2020)
- [18] Web 10, (2020), <https://ceaksan.com/tr/single-sign-on-sso-nedir>, (Eriřim Tarihi: 22/10/2020)
- [19] Web 11, (2020), <https://guvenmurat.com/openid-connect-ve-oauth2-nedir>, (Eriřim Tarihi: 21/10/2020)
- [20] Web 12, (2020), https://info.whitehatsec.com/Content-2019-StatsReport_LP.html?utm_source=website&utm_medium=0819-WebsiteWhiteHat2019StatisticsReport, (Eriřim Tarihi: 19/11/2020)

ÖZGEÇMİŞ

Mehmet Nazım TAT 1987 yılında İstanbul’da doğdu. İlk ve orta öğrenimini Kocaeli’nin Darıca ilçesinde tamamladı. 2006 senesinde başlamış olduğu İstanbul Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümünü 2010 yılında başarıyla tamamladı. 2019 senesinde yüksek lisans eğitimine Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Siber Güvenlik programında başladı. 2014 yılından bu yana Kuveyt Türk Katılım Bankasında yazılım mühendisi olarak başladığı kariyerine hali hazırda yönetici olarak devam etmektedir. Evli ve iki çocuk babasıdır.