MODERN API GÜVENLİĞİ:

TEHDİTLER VE ÇÖZÜMLER

V1.0

NURŞEN KARABULUT

YASİN KALLİ

İÇİNDEKİLER

ÖNSÖZ

API'ler (Uygulama Programlama Arayüzleri) günümüz dijital dünyasını daha önce hayal edemeyeceğimiz bir şekilde birbirine bağlayarak adeta modern yazılım ekosisteminin vazgeçilmez bir parçası haline gelmiştir. Birçok uygulama ve hizmet, sorunsuz bir şekilde entegre olmak için API'lere güvenirken, bu bağlantı noktalarının güvenliği her geçen gün daha kritik bir durum arz etmektedir. Dolayısıyla, dışarıya açık söz konusu uç noktalardaki güvenlik açıkları ve potansiyel zayıflıklar kötü niyetli aktörler için istismar kapısı arayabilmektedir.

API'ler, farklı yazılım bileşenlerinin iletişim kurmasına ve veri aktarmasına olanak tanıyan bir dizi protokoldür. Yazılım işlemleri arasında bir köprü görevi görerek, son kullanıcılar tarafından fark edilmeden verilerin akıcı bir şekilde değiş tokuş edilmesini sağlar. API'ler, bilgilerin doğru şekilde gönderilmesini garantilemek için yazılım platformlarının çeşitli bölümlerini birbirine bağlar. Bu bağlantı noktaları, dahili bir iletişim yolu ve harici araçların aynı verilere erişmesi için sanal bir aracı görevi görür.

API'ler aynı zamanda modern uygulama mimarisinin ve sistem tasarımının yapı taşları olarak hizmet ederler. Uygulama geliştiriciler, güçlü, dayanıklı, güvenli ve kullanıcı ihtiyaçlarını karşılayabilen uygulamalar oluşturmak amacıyla API'leri kullanır. Onları göremeseniz bile API'ler her yerdedir; modern hayatımız için olmazsa olmaz olan dijital deneyimlerinizi desteklemek için arka planda sürekli çalışırlar. Gerçek zamanlı analizler ve üst düzey entegre araçlarla, iş ve görevlerin daha hızlı ve verimli bir şekilde yürütülmesini sağlarlar.

Neredeyse tüm dijital faaliyetlerinizde, iş uygulamaları, e-ticaret siteleri, sağlık hizmetleri, bankacılık uygulamaları, ev otomasyonu ve mobil uygulamalarda API'leri kullanıyorsunuz. (Örneğin, yaygın kullanılan Google arama motorundan hava durumu uygulamasına, harita navigasyonundan seyahat rezervasyonuna kadar). API'lerin bu kadar çok veri ve hizmet paylaşımına olanak tanıması doğal olarak siber saldırganların da iştahını kabartan birincil bir hedef olmalarını sağlıyor.

Diğer taraftan kuruluşlar, saldırganların ele geçirme konusunda oldukça cazip olabilecek kişisel, hassas ve kurumsal bilgileri korumada yetersiz kalmaktadır. Güvenlik ilkelerinin uygulanmadığı API'ler, kimlik doğrulama bilgilerinin, müşteri PII verilerinin, kurumsal sırların ve çok daha fazlasının tehlikeye atılmasına yol açmaktadır.

Araştırmalara göre son yıllarda endişe verici bir artış eğiliminde olan API saldırılarından (2024 yılında yapılan bir araştırmada katılımcılardan %84'ünün son 12 ayda bir API güvenlik olayı yaşadığı ortaya çıktı. Bu oran; tüm zamanların en yüksek seviyesi olarak 2023 yılında tespit edilen %78 'den daha fazla!). Bu manzara; sadece güvenlik uzmanları değil, yazılım geliştiricilerden 3. taraf iş paydaşlarına varıncaya kadar, kritik uygulama ve verileri güvence altına almak, iş ve uygulama başarısını ve sürekliliğini sağlamak amacıyla API dünyasına uyum sağlamanın, API uç noktalarının neden ve nasıl korunacağını konusunda daha donanımlı hale gelmenin önemini vurgulamaktadır.

KİTAP HAKKINDA

Bu kitapta öncelikle modern bilişimde biraz tecrübesi olan teknik insanların seviyesi gözetilmiş olsa da hem deneyimli geliştiriciler hem de uygulama güvenliği alanına yeni adım atanlar için sağlam bir temel için lazım olan bilgiler de yer alıyor. Kitap boyunca, yalnızca teorik bilgi değil, aynı zamanda gerçek dünyada karşılaşabileceğiniz senaryolar ve bu senaryolara yönelik en iyi uygulamaların da yer aldığı çözüm önerileri de sunulmaktadır. Amacımız; API'lerin işlerinizi kolaylaştıran etkin yönleriyle birlikte güvenliğini sağlama sürecinde sizlere yol göstererek, dijital projelerinizin gizliliğini, bütünlüğünü ve erişilebilirliğini korumanıza yardımcı olmaktır.

Ana kaynak olarak hem içerik hem de seviye bakımından daha geniş bir kitlenin faydalanması amacıyla Michael Isbitski ‘nin API Security for Dummies kitabını referans aldık. Söz konusu kitapta özellikle OWASP API Security 2019 Top 10 ile ilgili olan bölüm yerine OWASP API Security 2023 Top 10 ‘da yer alan risk ve tehditleri alarak güncelledik. Ayrıca kitabı hazırlarken yararlandığımız kaynaklara kaynakça kısmında yer verdik. Kitapçığı özellikle V1.0 olarak çok uzatmamaya gayret ettik, V2.0 için geri bildirimleri ve notlarımızı almaya başladık ve bu alanda çalışan uzmanların geri dönüşlerini sabırsızlıkla bekliyoruz. Son olarak bu kitapçık Nurşen’in üstün gayretleriyle hazırlandı, en fazla teşekkürü o hak ediyor. Uygulama güvenliği konusunda gayreti takdire şayan ve bu alanda başarılı olacağına inanıyorum. Her türlü görüş, düşünce ve yorumlarınız için iletişim bilgilerimiz aşağıda yer alıyor.
Faydalı olmasını dileriz.

2025 Nisan

Nuşen KARABULUT
nursenkarabulut1@outlook.com

Yasin KALLİ
yakalli@gmail.com

API DÜNYASINA GİRİŞ 01

Eğer API'ler hakkında temel bir giriş ve arka plan bilgisi arıyorsanız, doğru bölüme geldiniz. Bu bölümde, özellikle internet ve web tasarımı bağlamında, uygulama programlama arayüzü (API) için işlevsel bir tanım yapılıyor. Ayrıca, API'lerin modern uygulama geliştirme ve bulut tabanlı mimariye olan etkileri ele alınıyor.

API'leri Yakından Tanıyalım

API tüketici (consumer) türlerini ve API protokollerini anlamak, mevcut en etkili güvenlik tekniklerini ve kontrollerini seçmek için kritik öneme sahiptir. Uygulama programlama arayüzleri (API'ler), bazen kod içerisinde düzenli olarak başvurulan fonksiyonlar veya kütüphaneler ile eş anlamlı olarak kullanılır. Örneğin, web API'leri, web tasarımı ve iletişim için özel olarak tasarlanmış bir API türüdür. Web API tasarım kalıpları, servis odaklı mimari (SOA) ve web protokolleri aracılığıyla uygulamaları destekleyen arka uç (back-end) servislerinden evrilerek 20 yılı aşkın süredir varlığını sürdürmektedir. Aşağıdaki bölümlerde, API türleri ve API'lerin çalışma prensipleri daha ayrıntılı olarak ele alınmaktadır.

Yaygın API Türleri

Tüketim (consumption) terimi, API çağrısının (API call) bir API'ye istekte bulunarak belirli bir işlevi gerçekleştirmesi, veriyi sorgulaması veya değiştirmesi anlamına gelir. API türleri, farklı iş senaryoları ve kullanım modellerine uyum sağlamak için yıllar içinde önemli ölçüde gelişmiştir.

Bazı yaygın API türleri şunlardır:

➢**Harici API'ler (External APIs):** Bu API'ler, mobil iş gücünü ve dünyanın her yerinden hizmetlere erişen müşterileri destekler. Adından da anlaşılacağı gibi, genellikle korumalı bir ağın veya internetin dışındaki kullanıcılara açıktır ve ağ kısıtlamaları daha gevşektir. Kimlik doğrulama (authentication) ve yetkilendirme (authorization) kullanılabilir veya kullanılmayabilir.

➢**Genel API'ler (Public APIs)**: Genel API'ler, harici API'lerin bir alt türü olup, internet üzerinden hem kullanıcılar hem de makineler tarafından tüketilmek üzere tasarlanmıştır. Genellikle erişim kontrolleri gevşektir veya kullanımın artmasını sağlamak için anonim erişime izin verirler.

➢**Açık API'ler (Open APIs):** Bu API'ler, özellikle finans sektöründeki açık bankacılık (open banking) uygulamalarında yaygın olarak görülmektedir. İlgili sektörde inovasyonu teşvik eder, hizmet entegrasyon seviyelerini artırır ve müşterilere her yerden işlem yapma veya verilere erişim özgürlüğü sağlar. Genellikle kimlik doğrulama (authentication) ve devredilmiş yetkilendirme (delegated authorization) mekanizmaları bulundurur.

➢**Dahili API'ler (Internal APIs):** Genellikle bir veri merkezinin veya özel bir bulut segmentinin kısıtlı ağ ortamında dağıtılan ve işletilen API'leridir. Bu API'ler, yalnızca ilgili kısıtlı ağdaki diğer uygulamalar veya kullanıcılar tarafından tüketilmek üzere tasarlanmıştır. Bu tür API'lerde kendi ortamı dışında görülme riski yani API ifşası (API exposure) sınırlı olduğundan kimlik doğrulama (authentication) ve yetkilendirme (authorization) kullanımı gevşek tutulabilir.

➢**Ortak API'ler (Partner APIs):** Kuruluşlar bazen dahili API'lere sınırlı erişim sağlayarak belirli dış tedarikçilere dijital tedarik zincirlerini (digital supply chain) genişletme ve güçlendirme imkânı sunar. Erişim kontrol kapsamı, dahili ve harici API'ler arasında bir noktada konumlandırılır.

➢**Üçüncü Taraf API'ler (Third-party APIs):** Genellikle bulut dağıtımlı (cloud delivered) hizmetler veya Hizmet Olarak Yazılım (SaaS) şeklinde sunulan bu API'ler, organizasyonların belirli işlevleri yeniden oluşturmasına gerek kalmadan daha hızlı hareket etmelerini sağlar ve yazılım geliştirme (software development) süreçlerinde birikerek uzun vadede ek maliyet ve karmaşıklık yaratan teknik borç (technical debt) oluşumunu engeller.

➢**Edinilmiş API'ler (Acquired APIs):** Bu API'ler tasarım tercihinden ziyade miras (inheritance) olarak devredilen bir türdür. Organizasyonlar, ticari/açık kaynaklı (open source) yazılım paketlerini satın alıp entegre ettikçe ve dağıttıkça bu API'leri miras olarak devralır.

Popüler API Protokolleri

API protokolleri, mimari seçimleri, test araçlarını ve çalışma zamanı (runtime) güvenlik kontrollerini etkiler. Ayrıca, protokoller bazen API şema tanımlarıyla iç içe geçebilir ve bu durumda API dokümantasyonunun bir parçası olarak ele alınmalıdır. Bölüm 2, API dokümantasyonu ve şemasıyla ilgili konuları kapsamaktadır.

Şimdilik, API protokolünü, bir API ile nasıl iletişim kurduğunuzu belirleyen bir araç olarak; API şemasını ise bu iletişimde verilerin nasıl görünmesi gerektiğini veya hangi işlevlerin mevcut olduğunu tanımlayan bir yapı olarak düşünebilirsiniz.

İşte en yaygın kullanılan API protokollerinden bazıları:

➢**Temsili Durum Transfer Protokolü (REST):** RESTful tasarım, istemci-sunucu (client-server) mimari desenlerini ve arayüzün arka uç (back-end) hizmetlerden ayrılmasını destekler. REST ile ilgili en zorlu kavramlardan biri, API uç noktalarının (endpoints), tasarım ve kodlama biçimine bağlı olarak organizasyonlar arasında büyük ölçüde farklılık gösterebilmesidir. URL yapısındaki öğeler fonksiyonları veya değişkenleri temsil edebilir. Ayrıca, HTTP metodları beklenenden farklı şekillerde kullanılabilir.

➢ **GraphQL:** GraphQL, bir sorgu dili (query language) olmasının yanı sıra veri manipülasyonu için de kullanılabilir. Facebook, GraphQL'i iki temel sorunu çözmek amacıyla geliştirmiştir:

- ➢ Birincisi, gereksiz Web API çağrılarını (API call) azaltmak,
- ➢ İkincisi ise REST API tasarımlarında bazen doğası gereği ortaya çıkan fazla veya eksik veri çekme sorunlarını ele almaktır.

Ön uç (front-end) performansına sağladığı faydalar nedeniyle GraphQL giderek daha fazla ilgi görmektedir, ancak REST API'ler varlığını sürdürmeye devam etmektedir.

➢**Uzaktan Prosedür Çağrısı (RPC) Tabanlı Protokoller:** JSON-RPC (JavaScript Object Notation RPC) ve XML-RPC (Extensible Markup Language RPC) terimlerini hala görebilirsiniz. Burada, JSON ve XML, API isteklerinde kullanılan kodlama formatlarını ifade eder. Google tarafından geliştirilen gRPC (google Remote Procedure Call), yüksek hızlı mikro servis iletişimi (microservice communication) için popülerlik kazanmıştır. Ancak, REST ve GraphQL'den farklı olarak, gRPC'nin tarayıcı tabanlı (browser-based) ön uçlar (front-end) için tercih edilen bir protokol olduğu pek görülmez.

➢**Basit Nesne Erişim Protokolü (SOAP):** SOAP API'leri hala bazı dahili API'lerde ve kurumsal uygulamalarda görülmektedir. Ancak, SOAP API uygulamaları genellikle ağır sıklet (heavyweight) olarak değerlendirilir ve büyük ölçüde REST API tasarımı ile yer değiştirmiştir.

API'lerle Modern Uygulama Tasarımı

API’ler, yazılım geliştirme ekiplerinin uygulama oluşturma biçimini önemli ölçüde değiştirmiştir. Kuruluşunuzun API’lerini ister kendiniz inşa edin, ister satın alın veya entegre ediyor olun, API’lerin ön uç (front end) tasarımı, işlevsellik ve veri alışverişi üzerindeki etkileri büyüktür. API’ler ayrıca mikro servis mimarisi (Microservices Architecture - MSA) ve bulut tabanlı (cloud-based) tasarım desenlerinde yaygın olarak kullanılmaktadır. Daha fazla ayrıntı için aşağıdaki bölümleri okumaya devam edebilirsiniz.

Ön Uç (Front-End) ve Arka Ucun (Back-End) Ayrılması

Çoğu kullanıcı ve hatta bazı uygulama geliştiriciler bir uygulamayı düşündüğünde, ön uç (front-end) arayüzünü veya grafiksel kullanıcı arayüzünü(GUI) hayal eder. Ancak, çoğu uygulama tasarımında, ön uç (front-end) arayüzleri arka uç (back-end) hizmetlerden ve API’lerden ayrılmıştır. Böylece, her bir bileşen bağımsız olarak güncellenebilir ve bakım (maintenance) süreçleri daha kolay hale gelir. Ön uç (front-end) kodu, web tarayıcısında çalışan bir betik (script) veya Android ve iOS için tasarlanmış yerel bir mobil uygulama kodu olabilir.

Kuruluşlar genellikle kullanıcı cihazlarına yüklenen ön uç (front-end) kodunu güvence altına almak ve güçlendirmek için çaba gösterir. Ancak, bir kuruluşun kontrol alanı dışındaki son kullanıcı cihazlarında bu korumayı sağlamak zor olabilir. Çalışanlar için geliştirilen mobil uygulamalar (employee mobile apps - örneğin, slack) için bu yaklaşım, kendi cihazını getir (Bring Your Own Device - BYOD) ve kurumsal sahipli, kişisel kullanım etkin (Corporate-Owned, Personally Enabled - COPE) senaryolarında teknik olarak daha uygulanabilirdir.

Ancak, müşteriler, hastalar veya vatandaşlar için geliştirilen mobil uygulamalar söz konusu olduğunda, kuruluşların son kullanıcı cihazları üzerinde sınırlı kontrolü vardır. Bu nedenle, istemci tarafı kod korumaları (client-side code protections) kolayca atlatılabilir.

Mikro Servis Mimarisinin (Microservice Architecture) API’lere Etkileri

Bazı durumlarda, mikro servisler ve API terimleri birbirinin yerine kullanılır. Ancak, bu kavramları daha iyi anlamanıza yardımcı olması için aşağıdaki tanımları göz önünde bulundurmak faydalıdır.

➢**Mikro Servis (Microservice):** Yalnızca tek bir işlevi yerine getirmek üzere tasarlanmış bir servistir. Bu tasarım deseni, birçok farklı işlevi yerine getiren monolitik tasarımla karşıtlık oluşturur. Mikro servis yaklaşımı, daha anlaşılır kod ve hizmetler arasında daha gevşek bir bağ (loose coupling) sağlamayı amaçlar.

➢**Mikro Servis Mimarisi (MSA):** Bir sistemin birçok birlikte çalışan mikro servisten oluştuğu bir mimari modeldir. MSA; gelişmiş tasarım esnekliği ve sürekli dağıtım ile daha hızlı servis ve altyapı ayağa kaldırmak gibi avantajlar sağlar.

➢**Monolitler:** Monolitler birçok farklı işlevi yerine getirmek için tasarlanmıştır. Bu nedenle, güncellemelerin ve bakımlarının yapılması zor olabilir. Bilişim sektöründe MSA'ya olan yoğun ilgiye rağmen, monolitik mimari hala yaygın olarak kullanılmaktadır. Bazı kuruluşlar, geliştirici deneyimi (developer experience) veya kurumsal geçmişleri (pedigree) nedeniyle monolitik tasarımı tercih edebilmektedir.

Bir mikro servisin sunduğu işlevselliğin mikro servis ortamının dışına açılması gerekmeyebilir. Ancak, kuruluşunuz mikro servis işlevselliğini dışarıya açmak istiyorsa, API'ler bu sürecin en uygun yoludur. Mikro servisler arası iç iletişim (inner microservice communication) için genellikle gRPC gibi bir protokol kullanılırken, mikro servislerin dış dünyaya açılması için REST veya GraphQL API'leri yaygın olarak tercih edilir. Bu süreçte API ağ geçitleri (API gateways) ile API'lerin yönlendirilmesi sıkça karşılaşılan bir yaklaşımdır. API aracılığı (API mediation) konusuna 2. Bölümde daha ayrıntılı olarak değinilmektedir.

MSA, yüksek dağıtılmış hizmet sayısı nedeniyle operasyonel karmaşıklığı artırabilir. Çoğu kuruluş hem monolitleri hem de mikro servisleri bir arada kullanır. Gerçek dünyada, birçok mimari model bu iki yaklaşımın birleşimi gibi görünmektedir.

API Altyapısının Değişimi: Bulut-Yerel (Cloud-Native) Yaklaşımının Rolü

Bazı durumlarda, mikro servisler ve API terimleri birbirinin yerine kullanılır. Ancak, bu kavramları daha iyi anlamanıza yardımcı olması için aşağıdaki tanımları göz önünde bulundurmak faydalıdır.

Bulut-yerel (cloud-native), farklı BT çevçevelerinde (framework) farklı anlamlar taşıyabilir. En geniş tanımıyla; bulut özelliklerini gösteren ve aynı zamanda bulut servis sağlayıcılara güç veren teknolojileri kullanan bir tasarım veya mimaridir. Yaygın bulut özellikleri arasında, web ölçek kapasitesi (web scale capacity) ve esneklik bulunur; burada, bir uygulama veya servis çalıştırmak için ihtiyaç duyduğunuz işlem gücü neredeyse sonsuzdur ve yalnızca bulut hizmet sağlayıcınızın veri merkezlerindeki mevcut donanımı ile sınırlıdır; bu donanım ise genellikle sizden soyutlanmıştır.

Bulut bilişimi (cloud computing) oluşturan teknolojiler şunlardır:

➢**Sanallaştırma (Virtualization):** Sanallaştırma, hypervisor teknolojisi kullanarak donanımı işletim sisteminden soyutlama yöntemidir. Belirli bir fiziksel ana bilgisayarda birçok sanal makine çalıştırılabilir ve makine durumu genellikle yoğunluk (density) terimiyle tanımlanır. Daha fazla sanal makineyi bir ana bilgisayara yerleştirmek donanımın daha verimli kullanılmasını sağlar ve donanımın boşta kalmasını böylece engellemiş olursunuz. Sanal makineler, uygulamaları ve servisleri çalıştırmak için yaygın olarak kullanılır, ancak sanal makinelerin hafif sıklet (lightweight) ve yüksek performanslı olmaları gerekir, özellikle de bunlar mikro servis mimarileri (MSA) için kullanılıyorsa!

➢**Konteynerleştirme (Containerization):** Konteynerleştirme, uygulamaları ve bağımlılıklarını (dependency) konteynerlere paketlemeyi içerir. Bu, işletim sistemini uygulamalardan ve hizmetlerden soyutlayarak yoğunluğu (density) daha da artırır. Konteynerler, taşınabilirliği (portability) ve ortam tutarlılığını (environment consistency) iyileştirir. Konteynerler, genellikle MSA içinde mikro servisleri çalıştırmak için işlem birimi olarak kullanılır.

Bulut-yerel (cloud-native) teknolojiler, API'lerle birkaç farklı şekilde ilişkilidir, ancak bilinen en yaygın iki yol şunlardır:

➢Kuruluşlar, API kodları ihlale uğradığında bir güvenlik ihlali durumunda potansiyel zararı ifade eden patlama alanını (blast radius) sınırlamak için konteynerleştirmeyi ve sanallaştırmayı birleştirir.

➢Tüm altyapılar, Kubernetes ve bulut servisleri gibi konteyner platformları içeren yapılardaki API'ler aracılığıyla ilan edilebilir ve işletilebilir.

API GÜVENLİĞİ İÇİN TEMEL TAŞLAR

02

API'leri güvence altına almanın geleneksel yaklaşımları arasında birçok yöntem bulunmaktadır. Bunlar arasında test etme (testing) ve yönetim aracı noktaları (mediation) da yer almaktadır. Bu bölümde, söz konusu yaklaşımlar ayrıntılı olarak incelenmektedir.

API Dokümantasyonu

API’leri dokümante etmek, API yaşam döngüsü boyunca bir dizi güvenlik ve güvenlik dışı amaç için hizmet edebilir.

Dokümantasyon, kuruluşunuza birkaç temel avantajı vardır:

➢API’lerle nasıl iletişim kurulacağını, sağladıkları işlevselliği ve değiştirdikleri verileri belirterek API saldırı yüzeyinizi (attack surface) daha iyi anlamanızı sağlamak.

➢Tasarım gözden geçirmeleri, güvenlik testleri, operasyonlar ve koruma gibi diğer faaliyetlere girdi sağlamak.

Tüm dokümantasyon türlerinde olduğu gibi, ekipler genellikle API’leri ve yeni işlevleri dokümante etmeyi ihmal ederler. Bu durum, API kayması (API drift) olarak adlandırılan bir soruna yol açar. API kayması, bir API'nin belgelenen veya beklenen davranışından zamanla sapması anlamına gelir ve genellikle versiyon güncellemeleri, yanlış yapılandırmalar (misconfigurations) veya belgelenmemiş değişiklikler nedeniyle ortaya çıkar. Bu durum, API envanterinizde (inventory) ve güvenlik duruşunuzda (security posture) büyük boşluklara neden olabilir.

API'lerinizi oluşturduğunuzda dokümante etmeyi ihmal etmeyin ve bu verilerinizi API envanteri veya kataloğunu beslemek için kullanın. Yeterli dokümantasyon sağlayan üçüncü taraf API'leri de buna dahil edin. Yeni API uç noktalarını (endpoints) ve işlevlerini keşfetmek için ortamınızı sürekli tarayarak ve trafiğinizi analiz ederek API envanterindeki boşlukları azaltın.

Aşağıdaki bölümler, API'ler dokümante edilirken hatırda tutulması gereken birkaç önemli noktaya değinmektedir.

Geleneksel Dokümantasyon Yaklaşımlarından Kaçının!

API dokümantasyonu planlarken, geleneksel bir dokümantasyon çalışması olarak ele alıp, uzun metin belgeleri veya görsel diyagramlar üretme amacını gütmekten kaçının. Söz konusu yazılı belgeler, slaytlar veya görsel diyagramlar, bazen uyumluluk (compliance) gereklilikleri veya tasarım gözden geçirmelerinin (design reviews) bir parçası olarak gerekli olabilmektedir.

Bazı işletmeler, güvenliği geleneksel uyumluluk (compliance) ve şelale (waterfall) yaklaşımlarından ayırma eğilimindedir. Ancak birçok BT organizasyonunun geliştirme ekipleri çevik (agile) metodolojileri ve DevOps uygulamalarını benimsedikçe, API geliştirmeyle birlikte gelen güvenlik temalı baskıların arttığını hissedeceklerdir.

Geleneksel dokümantasyon, güvenli tasarım incelemeleri ve tehdit modelleme (threat modelling) için yararlı olabilir. Ancak geleneksel dokümantasyon biçimleri, üretilmesi ve sürdürülmesi açısından genellikle zordur. Bu durumlarda API kaymasının (API drift) belirgin yan etkileri olabilir ve API dokümantasyonu, tüm değişikliklerin belgelendirilmesi için ek iş gücü tahsis edilse bile üretim ortamına (production deployment) alınan API'lerin beklentileri karşılayamadığı gerçeğiyle yüzleştirebilir.

API Şema Tanımlama Formatları

API şema tanımlama formatları, işinizi kolaylaştırmak için tasarlanmıştır. Şema, API oluşturulurken tanımlanabilir ve dokümante edilebilir. Şema tanımlamaları ayrıca test, entegrasyon, yayınlama (publishing) ve operasyonlar için yeniden kullanılabilir. Birçok tasarım, taklit ve geliştirme aracı, bir API'yi entegre ederken veya kodlarken API şema tanımlarını otomatik olarak oluşturabilir.

Bu özellikleri kullanarak, dokümantasyon yükünü azaltabilir ve sonrasında yaşanacak baş ağrılarını önleyebilirsiniz. Web API'leri içeren açık kaynak (open source) yazılım paketleri, genellikle ilgili API şema tanımlarını aldığınız kod havuzunda (repository - örneğin, git) veya paket yöneticisini (package manager - örneğin, npm) de içerir.

REST API'ler için en yaygın kullanılan makine formatları arasında Swagger veya OpenAPI Specification (OAS) bulunur. API tasarımınıza, geliştirme veya yayınlama araçlarınıza bağlı olarak RAML veya API Blueprint gibi diğer formatlar da mevcut olabilir. GraphQL API'lerini keşfediyorsanız, GraphQL şema tanımları ile de çalışmanız gerekecektir.

API Testleri

Shift-left API güvenlik uygulamaları kapsamında özel bir odak noktası, işlem hattının (pipeline) güvence altına alınmasıdır. Bu, güvenlik araçlarının sürekli entegrasyon/sürekli teslimat (CI/CD) yapı hatlarına ve git tabanlı geliştirici iş akışlarına (workflows) entegre edilmesini gerektirir.

Yapı hatlarını (build pipelines) güvence altına almak için ;

➢Bağımlılık analiz araçları (dependency analyzers),
➢Statik analiz araçları(static analyzers),
➢Dinamik analiz araçları(dynamic analyzers),
➢Şema doğrulayıcılar(schema validators),
➢Fuzzing araçları ve güvenlik tarayıcıları gibi çeşitli güvenlik test araçları kullanılır.

Gerekli güvenlik araçlarının türü, yapı hattından geçen bileşenlere, inşa edilmesi gereken öğelere ve bunların dağıtılacağı yerlere bağlı olarak değişir. Aşağıdaki bölümlerde, yapı hatlarında güvenlik testi uygulamalarının avantajları ve dezavantajları ele alınmaktadır.

Uygulama Güvenliği Test Araçları (SAST & DAST)

Statik Uygulama Güvenlik Testi (SAST), orjinal kaynak kodunu (source code) potansiyel zayıflıklar ve güvenlik açıkları açısından analiz etmek için kullanılır. Genellikle kodun versiyon kontrol sistemine (VCS) gönderildiği veya derleme (build) aşamalarında çalıştırılır. Buna karşılık, Dinamik Uygulama Güvenlik Testi (DAST), çalışan bir uygulamayı istismar edilebilir durumlar açısından analiz etmek için kullanılır. Genellikle üretim (production) ortamına dağıtımdan önce veya üretim ortamında sürekli olarak çalıştırılır çünkü uygulamanın çalışır durumda olması gerekir.

SAST ve DAST, özel API kodunuzdaki zayıflıkları ve istismar edilebilir durumları ortaya çıkarabilir. Ancak, bu tarama yöntemleri, saldırganların hedef aldığı ve istismar ettiği iş mantığı açıklarını (business logic flaws) tespit edemez. İş mantığı ve API'leri nasıl tasarlayıp kodladığınız, işletmenize özgü bir durumdur. Sonuç olarak, iş mantığını temsil eden kodlar, SAST veya DAST araçlarının imzalar (signatures) oluşturabileceği belirli kalıpları genellikle takip etmez.

Çoğu test aracı, kimlik doğrulama veya yetkilendirme (authentication & authorization) testlerinde yüzeysel kontrollerin ötesine geçemez. Genellikle ve yalnızca Temel ve Özel Erişim (Basic ve Digest Access) gibi zayıf kimlik doğrulama yöntemlerini tespit etmek veya kimlik bilgilerini nasıl girdiklerini, ilettiklerini ve depoladıklarını analiz etmek gibi işlemler yapar.

Bazı DAST araçları, yetki yükseltme (privilege escalation) zafiyetlerini tespit edebilir. Ancak, bunu gerçekleştirebilmek için ilgili uygulama ve API'ye karşı birden fazla kez çalıştırılması gerekir. Ne yazık ki, birçok organizasyon sıkışık yayın (release) takvimleriyle çalıştığı için, burada zaman kritik bir faktördür.

DAST araçları, karmaşık uygulamalarda uzun süre çalışmalarıyla bilinir. İşlem hattındaki (pipeline) taramaların tamamlanması için yeterli zaman ayrılmalı veya taramaların yayın (release) süreçlerini engellemeyecek (non-blocking) şekilde gerçekleştirilmesi sağlanmalıdır.
SAST ve DAST her zaman belirli eksikliklere sahip olmuştur. Ancak, API güvenliği söz konusu olduğunda bu eksiklikler daha da belirgin hale gelir. Evet, bu taramaları özelleştirilmiş uygulamalarınızda ve API kodlarınızda çalıştırmalısınız, ancak bu tarayıcıların tüm güvenlik açıklarını tespit etmek için tasarlanmadığını kabul etmeniz gerekir. API'lerin çalışma zamanındaki (runtime) davranış analizinin önemi yadsınamaz bir gerçektir. (Daha fazla ayrıntı için Bölüm 4'e bakınız).

Satatik Analiz için API Şema Validatörleri

Statik analiz yöntemlerinden biri olan API şema validatörleri (schema validators), genellikle yapı hattı (build pipeline) güvenliği için DevOps dostu bir çözüm olarak tanıtılır. Bu çözüm genellikle şu şekilde sunulur: "Bize şema tanımlarınızı verin; API'lerinizi tarayalım, uyumluluğunuzu sağlayalım ve güvenlik açıklarınızı tanımlayalım."

Ancak şema doğrulama (validation) yaklaşımında dikkate alınması gereken bazı sorunlar vardır:

➢**Tüm öğeler API şeması içinde tanımlanmak zorunda değildir.** OAS (OpenAPI Specification) ve Swagger gibi API spesifikasyon formatları, API dokümantasyonunda tüm alanları veya fonksiyonları tanımlamayı zorunlu kılmaz. Geliştiriciler genellikle eksik dokümantasyon bırakır. Özellikle Postman gibi API tasarım araçlarını kullanmıyorlarsa, geliştiriciler API'leri tam olarak dokümante etmeyi unutabilir.

➢**Birçok organizasyon dokümantasyon konusunda zayıf kalmaktadır.** İnsanlar genel olarak dokümantasyon konusunda başarılı değildir, özellikle her şeyi tam anlamıyla belgelemek söz konusu olduğunda... Dokümantasyon eksikliği yalnızca geliştiricilere özgü bir sorun değildir. OAS (Open API Specification) kendi kendini dokümante etme (self-documenting) avantajına sahip olsa da, yine de manuel çaba gerektirir. Ayrıca, bazı araçlar OAS tanımlarını üretme konusunda diğerlerinden daha yetenekli olabilir.

➢**API kayması (API drift) doğası gereği meydana gelir.** Üretimde çalışan API ile orijinal spesifikasyon arasındaki farklılıklar yaygındır. API kayması (API drift), organizasyonların güvenli tasarım gözden geçirmeleri (secure design review) ve tehdit modelleme süreçlerinde karşılaştığı en büyük sorunlardan biri ile paralellik gösterir. Çoğu zaman, planlanan spesifikasyonlar ile gerçek dünyadaki ürün arasında büyük farklılıklar olabilir.

Seçtiğiniz AST (Application Security Testing) veya şema analiz araçları, entegre ve otomatik hale getirilerek işlem hattına (pipeline) hizmet etmelidir. Araçlar, mevcut Git iş akışlarına (workflows) ve CI/CD süreçlerine uygun çalışmalıdır. Manuel taramalar yapmak, sorunları bulmak ve rapor üretmek yeterli değildir. Bu yaklaşım, çoğu organizasyonun benimsediği çevik (agile) metodolojiler ve DevOps uygulamaları için uygun değildir.

Şema validasyonu (schema validation) ve zorunlu uyumluluk (enforcement), neyin izin verildiğini tanımlayan ve diğer her şeyi reddeden eskide kalmış pozitif güvenlik paradigmasının yeniden yorumlanmış halidir. Eskiden güvenlik ekipleri kurallar (rules) veya imzalar (signatures) oluşturmaktan sorumlu iken, bu yük artık geliştirme ekiplerine kaydırılmaktadır. Şema validasyonu (schema validation), yalnızca belirli istismar edilebilir durumları ve yanlış yapılandırmaları (misconfigurations) tespit edebilir. Şema analizi, iş mantığı (business logic) hatalarını tespit edemez.

API Aracılığı (API Mediation) 101: API'ler ve İlgili Mekanizmalar

Bir API'yi doğrudan bir web veya uygulama sunucusu aracılığıyla hizmete açmak mümkün olsa da, bu uygulama tipik kurumsal mimarilerde daha az yaygındır.

API aracılığı (API mediation);

➢Ağ yük dengeleyicileri (network load balancers),
➢Uygulama dağıtım denetleyicileri (application delivery controllers),
➢Kubernetes giriş denetleyicileri (Kubernetes ingress controllers),
➢Yan bileşen proxy'leri (sidecar proxies),
➢Hizmet ağı girişleri (service mesh ingresses) gibi çeşitli diğer mekanizmalarla gerçekleştirilebilir.

Buradaki önemli nokta, API trafiğine aracılık edebileceğiniz ve gözlemleyebileceğiniz kurumsal mimarideki birden fazla noktayı anlayabilmektir. Çeşitli API aracılığı (API mediation) noktalarının temel bir fikrini edinmek, API güvenliği için kritik önem taşır; böylece tüm trafiği görebilir, gerektiği şekilde uygulayabilir ve hızlıca iyileştirme yapabilirsiniz.

Belirli bir yönetim aracı noktası mekanizmasının seçimi veya API trafiğini hangi mimari noktasında vekil (proxy) işlevi göreceğine karar verme konusu, bu kitabın kapsamı dışında kalmaktadır.

Bu bölümler, API'ler ve aracılığı (mediation) hakkında bilmeniz gerekenleri temel seviyede açıklamaktadır.

Vekil (Proxy) Sunucu Dağıtımı — Ters ve İleri Proxy

API arabirimleri ve ön uç (front-end) ile arka uç (back-end) arasındaki "API facade" gibi tasarım desenleri, API'lerin önüne vekil (proxy) aracılığıyla bir yönetim aracı noktası (mediation) katmanı eklemeyi içerir. API aracılığı genellikle aşağıdaki iki vekil sunucu türünden birini (veya her ikisini) dağıtarak gerçekleştirilir:

➢**Ters Vekil (Reverse proxy):** Ters proxy, API veya servise gelen trafiği analiz eder ve buna göre işlem yapar. Örneğin, API kullanımını veya tüketimini (consumption) izlemek amacıyla gelen istekleri (inbound calls) yönlendirmek isteyebilirsiniz.

➢**İleri Vekil (Forward proxy):** İleri proxy, API veya servisten çıkan trafiği analiz eder ve buna göre işlem yapar. Örneğin, bulut tabanlı yazılım olarak hizmet (SaaS) olarak sunulan bir API bağımlılığına (dependency) giden istekleri (outbound calls) proxy'den geçirmek isteyebilirsiniz.

İşletme gereksinimleri ve kullanım senaryoları, her iki vekil (proxy) dağıtım türünün benimsenmesini yönlendirecektir. API aracı noktaları, geliştirilmiş görünürlük, hızlandırılmış dağıtım, artan operasyonel esneklik ve özellikle API erişim kontrolü konusunda gelişmiş uygulama yetenekleri gibi geniş bir alanda fayda sağlar. Herhangi bir mimaride, hem ileri hem de ters proxy'lerin bulunmasını bekleyebilirsiniz. Vekil(proxy) sunucular, API pratiği alanında ve uygulama geliştirilmesinden ziyade altyapı ve operasyonlar, ağ mühendisliği ve kurumsal mimari alanına giren bir konudur.

API Aracılığı Olarak İşlev Gören Noktalar

Kurumsal mimarilerde, API isteklerini ve yanıtlarını yönlendirmek için farklı vekil (proxy) mekanizmaları bulacaksınız:

➢**Ağ Yük Dengeleyicileri (NLB):** Ağ yük dengeleleyicileri (NLB), fiziksel donanım veya yazılım tabanlı olabilir ve ağ yükünü dengelemek için sunucular ve servisler arasında gelen istekleri dinamik olarak yönlendirmekten sorumludur.

➢**Uygulama Dağıtım Denetleyicileri (ADC):** Uygulama dağıtım denetleyicileri (ADC), ağ yük dengeleyicilerine (NLB) benzer şekilde çalışır, ancak, genellikle uygulamaya özgü yönlendirme, yük dengeleme ve önbellekleme (caching) gibi ek işlevler içerir. Bazı durumlarda ADC (uygulama dağıtım denetleyicileri) ve NLB(ağ yük dengeleyicileri) birbirinin yerine kullanılabilir ve bu fark yalnızca hizmet sağlayıcı (vendor) terminolojisine bağlı olabilir.

.➢**API Ağ Geçitleri (API Gateways):** API ağ geçitleri, API trafiğini yönlendirmek için özel olarak tasarlanmıştır. Mesaj çevirisi (message translation) ve farklı protokoller arasında köprü oluşturma (bridging) gibi işlemleri destekleyerek iç ve dış mimariler arasındaki iletişimi kolaylaştırabilir.

➢**API Yönetimi (APIM):** API yönetimi (APIM) çözümleri, yalnızca API ağ geçitlerine (API gateway) bağlı kalmayarak API yaşam döngüsünü yönetmeye yönelik ek yetenekler sağlar. Genellikle organizasyonların ağ geçitleri ve API uç noktaları (endpoints) arasında politikaları merkezi olarak yönetmesine, genel izleme sağlamasına ve geliştirici veya iş ortakları için self-servis erişim imkânına olanak tanır. Ancak, API yönetimi (APIM) çözümleri yine de API ağ geçitlerine bağlıdır ve politikaların uygulanması için bir aracı vekil (mediating proxy) mekanizması olarak API ağ geçitlerinden faydalanır.

Bir organizasyon mikro servis mimarisini (MSA) hızlı bir şekilde benimsiyorsa, büyük olasılıkla Kubernetes giriş denetleyicileri (Kubernetes ingress controllers) ve servis ağları (service meshes) kullanıyordur. Giriş (ingress) yapıları, API ağ geçitleri (API gateway) ile ağ yük dengeleyicilerinin (NLB) birleşimi gibidir ve mikro servislerin tasarımına bağlı olarak birden fazla protokolü destekleyebilir. Bu yapılarda proxy'ler, mikro servisleri veya API'leri çalıştıran belirli iş yüklerine daha yakın bir noktada dağıtılabilir. Bu tür vekillere "sidecar proxy" adı verilir.

Sidecar proxy, bir mikro servisin yanında çalışan ve gelen/giden trafiği yöneten bir proxy türüdür. Hizmetler arası güvenlik, trafik kontrolü, yük dengeleme ve gözlemlenebilirlik gibi görevleri üstlenir. Sidecar proxy'ler, aynı zamanda loopback proxy mantığıyla çalışır. Loopback proxy, bir sistemin kendi içinde trafiği yönlendirmesine olanak tanır ve dış ağa çıkmadan iletişimi sağlar. Bu yapı, iç trafiğin denetlenmesi, güvenli hale getirilmesi ve yönetilmesi açısından kritik bir rol oynar.

API Yönetimi (APIM) ile Politikaların Uygulanması

API yönetimi (APIM) çözümleri, belirli kullanım senaryolarını karşılamak için bir dizi yetenek sunmaktadır.

Bu yetenekler genellikle aşağıdakileri maddeleri içerir:

➢**Ağ Güvenliği:** NLB'ler (ağ yük dengeleyicileri) veya ADC'ler (uygulama dağıtım denetleyicileri) gibi, ağ bağlantısı, API ağ geçidi (API gateway) içindeki API'lerle sınırlı olabilir. Bu genellikle, belirli API ile iletişim kurabilecek kaynak IP adreslerinin iletişim kuracağı hedef IP adreslerini kısıtlamak için izin ve engelleme listeleri (allow & deny list) ile taleplerin ne sıklıkta yapılabileceğini sınırlamak amacıyla oluşturulan hız sınırlamalarını (rate limits) ve transit halinde taşınan mesajların (data in transit) gizliliğini ve bütünlüğünü sağlamak için TLS şifrelenmeyi içerir.

➢**Kimlik Doğrulama (Authentication) ve Yetkilendirme (Authorization):** Kuruluşlar genellikle API ağ geçitlerinde (API gateway) erişim kontrolü uygular, böylece API çağrıları (API call) kimlik doğrulamasından ve yetkilendirme mekanizmasından geçirilir. Yaygın protokoller arasında kimlik doğrulama için OIDC ve yetkilendirme için OAuth2 bulunur. Token çevirici (token translation) desteği genellikle, örneğin bir API uygulamasının eski protokollerle (Security Assertion Markup Language - SAML gibi) entegrasyon gerektirdiği durumlar için kullanılabilir.

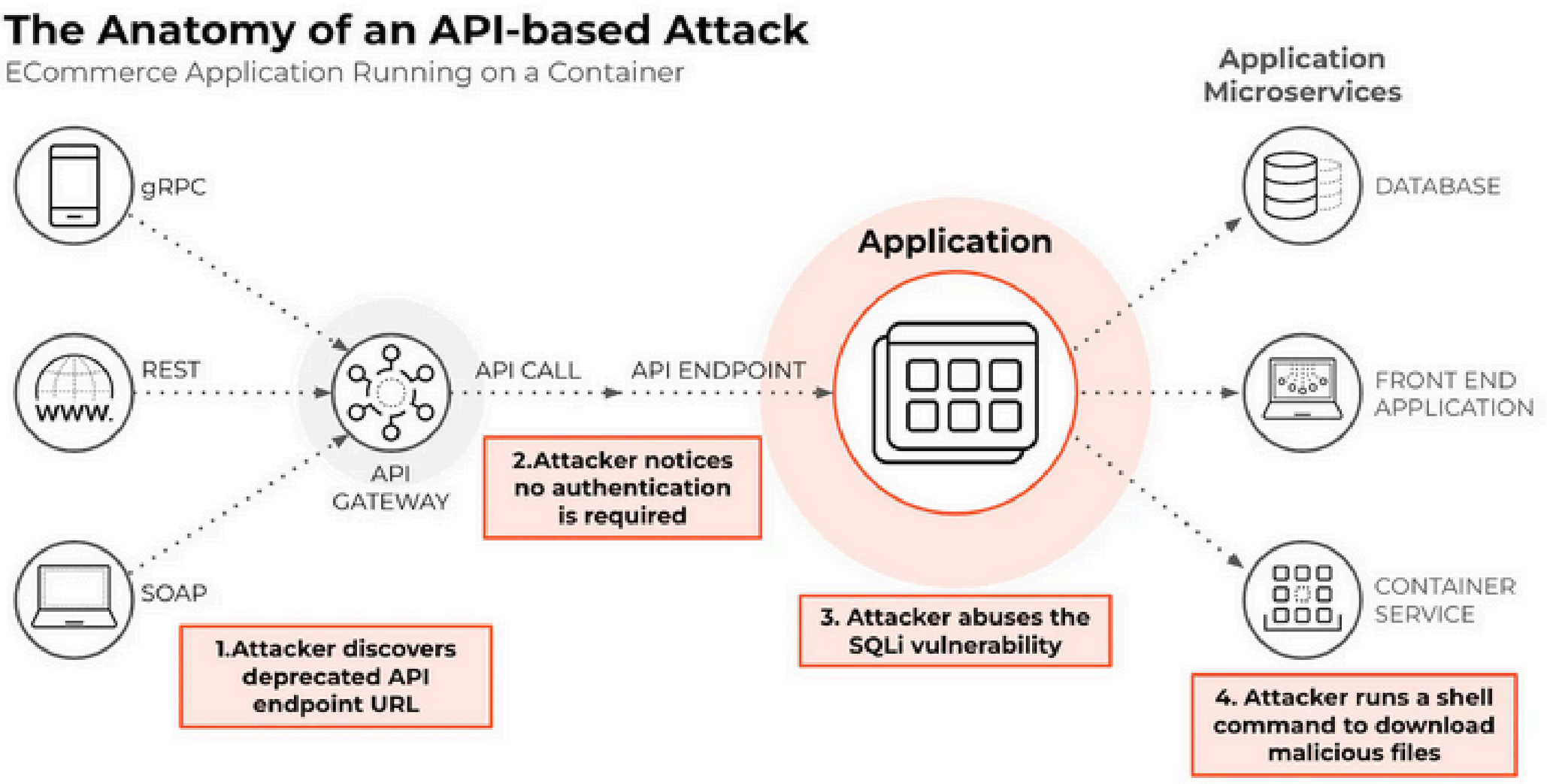
➢**Temel Tehdit Koruması:** API aracı (mediation) mekanizmaları, tasarım gereği mesaj filtreleme ve protokol çevirici (protocol translation) işlemlerini destekler. Birçok API yönetimi (APIM) çözümü, kod enjeksiyonu (code injection) saldırılarında yaygın olarak kullanılan kötü amaçlı karakter setlerini engellemek için temel kurallar sağlar. Diğer tehdit koruma yetenekleri arasında, gelen API taleplerini API şema tanımlarına veya manuel konfigürasyonlara dayalı olarak kısıtlama yer alır. Parametre uzunlukları, parametre değerleri, dizi boyutları (array size) ve daha fazlası üzerinde kısıtlamalar uygulanabilir.

Mümkün olduğunda, diğer sistemlerle API entegrasyonlarını bozmadığı sürece API yönetimi (APIM) güvenlik kontrollerini etkinleştirin. Bu ayarlar bazı API saldırı türlerini hafifletebilir (mitigation), ancak çoğu API kötüye kullanımı (API abuse) ve iş mantığı saldırılarına (business logic attacks) karşı koruma sağlayamaz.

YAYGIN API SALDIRILARI 03

Bu bölüm, API saldırılarının uygulama saldırılarından nasıl farklı olduğunu sade bir dille açıklamaktadır. Ayrıca, yaygın API güvenlik açıklarını anlamak için iyi bir başlangıç noktası olan OWASP API Security 2023 Top 10 hakkında daha fazla bilgi edinebilirsiniz. Bu bölüm ayrıca kaba kuvvet (brute force), kimlik bilgisi doldurma (credential stuffing) ve veri kazıma (data scraping) gibi API'leri özel olarak gerçekleştirilen otomatize saldırı türlerinden de bahsetmektedir.

API Saldırılarının Uygulama Saldırılarından Farkları



API tabanlı saldırılarının anatomisi (Kaynak: Palo Alto)

API saldırı modelleri (attack patterns), ağ güvenliği ve uygulama güvenliği alanlarında yaygın olarak bilinen saldırılardan farklılık gösterir. Saldırılar, bu iki alandan ortak unsurlar içerebilir, ancak çoğunlukla API kullanım senaryolarına ve iş mantığına özgü olarak gerçekleşir.

Saldırganlar, altyapı kontrollerindeki yanlış yapılandırmaları (misconfigurations), kaynak kodda yer alan güvenlik açıklarını (vulnerabilities) veya bunların birleşimini istismar edebilir. Güvenliği yalnızca bir yazılım geliştirme ekibine bırakmak zaten kötü bir stratejidir. Bazı shift-left yaklaşımlar ve yanlış yönlendirilmiş DevOps uygulamaları, güvenlik sorumluluğunu gereğinden fazla yazılım geliştirme ekiplerine yükleyebilmektedir. Bu durumda uygulama geliştiriciler, altyapı ve güvenlik kavramlarında yeterli uzmanlığa sahip değillerse kaçınılmaz olarak API güvenliğinde açıklar oluşmasına neden olur.

Saldırganlar, ön uç (front-end) uygulamalarınızı kullanarak arka uç (back-end) API'lerinize bağlanır ve iş mantığınızı çözümlemeye çalışır. Ayrıca, modern uygulamaların birçok birinci taraf (first-party) ve üçüncü taraf (third-party) API'lerle yüksek düzeyde bağlantılı olması saldırganlar için büyük bir avantajdır. Bu API'lerden herhangi biri saldırganlar tarafından sömürülebilir (exploitable) durumda olabilir.

İstismarlara Kapı Açan Ön Uç (Front-end) Uygulamalar

Güvenlik ekipleri bazen API'leri güvence altına almak için uç noktaları (endpoints) korumaya veya istemci (client) uygulamalarını güçlendirmeye çalışır. Ancak, bazı API tüketim (API comsumption) senaryolarında uç noktaları güvenli hale getirmek veya istemci kodunun değiştirilmediğinden emin olmak mümkün değildir. Bu gerçek, müşterilerin harici (external) API'leri, herkese açık (public) API'leri ve açık (open) API'leri güvenli olmayan ağlardan (örneğin, internet) çağırdığı senaryolar için özellikle geçerlidir.

Saldırganlar istemci tarafı (client-side) veya ön uç (front-end) kodunu tersine mühendislik (reverse engineering) araçlarıyla analiz eder. Bunu yapmak için uygulama ikili (binary) dosyaları açan, kaynak koda dönüştüren (decompile) veya ayrıştıran (disassemble) araçlar kullanırlar. Saldırganlar ayrıca, Burp Suite veya OWASP Zed Attack Proxy gibi HTTP trafik yakalama araçlarını da kullanır. Bu araçlar, genel uygulama sorunlarını gidermek ve güvenlik değerlendirmesi (security assessment) yapmak için kullanılan vekil (proxy) araçlardır. Ancak, hem yetişmiş güvenlik uzmanlarının hem de saldırganların elinde son derece güçlüdürler.

Dikkatli olun; her zaman bir uç nokta ihlale uğradığında (compromised) istemci tarafı kodunun değiştirilmiş olabileceğini varsayın ve saldırganlar için en değerli hedeflerin arka uç hizmetleri (back-end services) ve sağladıkları veriler olduğunu unutmayın!

Dijital Tedarik Zincirlerine (Supply Chain) API Desteği

Kuruluşunuzun API ekosistemi sadece geliştirdiği API'lerden ibaret değildir. Satın alınan uygulamalardaki (acquired applications) API entegrasyonları (API integrations) ve API bağımlılıkları (API dependencies), herhangi bir kuruluşun API portföyünü tamamlar. Tüm bu API'ler toplu olarak geniş bir dijital tedarik zinciri (digital supply chain) oluşturur ve kuruluşlar için saldırı yüzeyini (attack surface) artırır. Birinci taraf (first-party) ve üçüncü taraf (third-party) API'lerin ve altyapının birleşimi, hangi güvenlik kontrollerinin mevcut olduğunu ve hangi kodun görülebilir olduğunu daha da karmaşık hale getirir.

Saldırganlar, dağıtık mimarilerin ve tedarikçi entegrasyonlarının doğasını çok iyi bilirler ve genellikle en zayıf halkayı hedef alırlar. Aslında, saldırganlar genellikle bir API'deki güvenlik açığını başlangıç saldırı vektörü (initial attack vector) olarak kullanır ve ardından diğer ağlara, sunuculara, zararlı yüklere (payloads), uygulamalara ve API'lere yönelir.

Bu çok adımlı saldırı dizileri (multi-step attack sequences), geleneksel güvenlik kontrollerinden kaçınmayı başarır. API ekosistemlerinin bu gerçekleri, çalışma zamanı (runtime) davranış analizine duyulan ihtiyacı daha fazla vurgular. Bu analiz, yeni ortaya çıkan istismar yöntemlerini (exploitationtechniques) tespit etmek için gereklidir. 4. Bölümde, davranış analizi (behavior analysis) ve çalışma zamanı koruması (runtime protection) konularına daha derinlemesine değinilmektedir.

API Açıklarına Mercek Tutalım: OWASP API Security 2023 Top 10

Open Web Application Security Project (OWASP), yıllar içinde popülerlik kazanmış bir organizasyondur ve Application Security Top 10 güvenlik sektöründe sıklıkla referans alınmaktadır. 2019 yılında OWASP, API Security Top 10 listesini yayınlayarak en yaygın 10 API güvenlik açığını tanımlamıştır. Bu liste, API'lerde görülen yaygın zayıflıkları ve güvenlik açıklarını anlamaya başlamak için temel bir kaynaktır. Bunun yanı sıra, bu liste eğitim ve farkındalık sağlama amacıyla da faydalıdır ve API güvenlik açıklarını sınıflandırmak için hafif sıklet bir taksonomi (lightweight taxonomy) olarak da kullanılabilir.

2023 yılında OWASP, API Security Top 10 listesini güncelleyerek, gerçek dünya verileriyle uyumlu gelişen tehditlere ve API güvenlik açıklarına daha yakından bakma ve değerlendirme imkanı sağlamıştır.

API güvenliği, her geçen gün daha fazla önem kazandığından, bu liste sadece potansiyel güvenlik açıklarını belirlemekle kalmaz, aynı zamanda saldırganların bu açıkları nasıl istismar edebileceğini ve organizasyonların bu tür tehditlerle nasıl başa çıkabileceklerini de anlamalarına yardımcı olur.

OWASP API Security Top 10 listesindeki güvenlik açıkları, genellikle kimlik doğrulama (authentication), yetkilendirme (authorization), aşırı veri ifşası (excessive data exposure), yanlış yapılandırmalar (misconfigurations) ve kod enjeksiyonları (code injection) gibi temel güvenlik sorunlarını içerir. Bu güvenlik açıkları, API'lerin doğru şekilde yapılandırılmaması veya yetersiz güvenlik önlemleri ile korunmaması nedeniyle ortaya çıkabilir. Bununla birlikte, bu açıklar, API'ler aracılığıyla yapılan saldırıların karmaşıklığını artırabilir, otomatize saldırılara yol açabilir ve daha büyük zararlara sebep olabilir.

Saldırganlar, OWASP API Security Top 10'da tanımlanan güvenlik açıklarını birleştirerek saldırı zincirleri (attack chains) oluşturabilir. Bu şekilde, API'lerin dökümünün çıkarılması (enumeration) ve veri kazıma (data scraping) gibi otomatik saldırılara karşı işletmeleri savunmasız hale düşürebilirler. Ayrıca, sızdıran API'ler (leaky APIs) olarak bilinen API'ler, hassas verilerin istemeden ifşa edilmesine yol açabilir. Sızdıran API'ler (leaky APIs), genellikle yetersiz erişim kontrolleri veya yanlış yapılandırmalar nedeniyle, kişisel veya hassas bilgilerin istemci tarafında ifşa olmasına neden olur.

API'ler üzerindeki güvenlik endişeleri, sadece OWASP API Security Top 10 ile sınırlı kalmamalıdır. Bu liste, API güvenlik açıklarını tanımlamanın bir aracı olmanın yanı sıra, güvenlik zafiyetlerinin uygulama geliştirme sürecinde nasıl daha iyi ele alınabileceği konusunda da rehberlik sağlar. Ayrıca, değerli varlık yönetiminin (asset management) yetersiz olması veya yetersiz loglama ve izleme (insufficient logging and monitoring) gibi güvenlik zafiyetleri, tüm sistemi tehdit eden kritik konulardır. Ancak, hızlı geliştirme döngüleri (rapid development cycles), dokümante edilmemiş API değişiklikleri (undocumented API changes) ve sürekli devam eden entegrasyon çalışmaları (integration work), API'lerde bu sorunları daha belirgin hale getirebilmektedir.

Sonuç olarak, OWASP API Security Top 10, organizasyonların API güvenliği konusunda daha bilinçli olmalarını sağlayan, önemli bir rehberdir. Ancak, bu liste yalnızca başlangıçtır ve API güvenliğini sağlamada daha geniş bir güvenlik yaklaşımının parçası olarak değerlendirilmelidir.

API1:2023 – Kırık Nesne Düzeyinde Yetkilendirme (Broken Object Level Authorization - BOLA)

Nesne düzeyinde yetkilendirme, bir API çağrısında (API call) bulunan kullanıcının belirli bir nesneye erişim yeteneğini doğrulamak için kullanılan bir kontrol mekanizmasıdır. Bir uygulama altyapı düzeyinde doğru yetkilendirme kontrolleri uygulansa bile, geliştiriciler çoğu zaman bu kontrolleri belirli nesnelere erişim izni vermeden önce uygulamayı ihmal ederler.

Siber saldırganlar, kırık nesne düzeyinde yetkilendirme (BOLA) güvenlik açığı bulunan API uç noktalarını (endpoints) manipüle ederek kolayca istismar edebilir. Bunu yapmak için API isteği içinde gönderilen nesne kimlik bilgilerini (ID) değiştirirler. Bu güvenlik açıkları API tabanlı uygulamalarda son derece yaygındır, çünkü sunucu (server) bileşeni, istemcinin (client) durumunu tam olarak takip etmez. Bunun yerine, sunucu istemciden gelen nesne kimlik bilgileri (object IDs) gibi parametrelere dayanarak hangi nesnelere erişilebileceğine karar verir.

Her API uç noktası(endpoints), bir nesnenin kimlik bilgisini (ID) alıp o nesne üzerinde herhangi bir işlem gerçekleştiriyorsa, nesne düzeyinde yetkilendirme kontrollerini uygulamalıdır. Bu kontroller, oturum boyunca sürekli olarak yapılmalı ve kimliği doğrulanmış (authenticated) kullanıcının talep edilen işlem için yetkili olup olmadığı da ayrıca kontrol edilmelidir.

Nesne düzeyinde yetkilendirmenin uygulanmaması, veri sızıntısı (data exfiltration), yetkisiz veri görüntüleme, veri değiştirme veya silme gibi ciddi güvenlik açıklarına yol açabilir. BOLA güvenlik açığı, hesabın tamamen ele geçirilmesine (account takeover) de neden olabilir. Örneğin, saldırganlar parola sıfırlama akışlarını (password reset flow) ele geçirerek, yetkili olmadıkları hesapların kimlik bilgilerini sıfırlayabilirler.

API2:2023 – Kırık Kimlik Doğrulama (Broken Authentication)

Kimlik doğrulama mekanizmaları, özellikle herkese açık (public APIs) veya koruma mekanizmaları zayıf olduğunda, saldırganlar için cazip hedefler haline gelir. Bazı API kullanım senaryolarında, kullanıcı veya makine kimlik doğrulama bilgilerini (authentication material) girmeye zorlamak mümkün olmayabilir. Bu durumlar, kimlik doğrulama bileşenini birçok güvenlik açığına karşı savunmasız hale getirir.

API'lerde kimlik doğrulama (authentication) açıkları genellikle aşağıdaki nedenlerden kaynaklanır:

➢**Koruma Mekanizmalarının Eksikliği:** API uç noktalarının (endpoints) kimlik doğrulama (authentication) gerektirmemesi veya yetersiz doğrulama mekanizmalarına sahip olması gibi durumlarda ortaya çıkar. Bu durum, iç ağlara (internal networks) açık API'lerde, üçüncü taraf (third party) entegrasyonlarında ve ara katman uygulamalarda (middleware) sıkça görülür.

➢**Yanlış veya Hatalı Kimlik Doğrulama Yapılandırmaları:** Kimlik doğrulama (authentication) mekanizmasının saldırı vektörleri (attack vectors) dikkate alınmadan uygulanması, API'lerin belirli kullanım senaryoları için uygun olmaması veya güvenlik ilkelerine aykırı yapılandırılması güvenlik açıklarına yol açabilir. Örneğin, IoT cihazları için tasarlanmış bir kimlik doğrulama mekanizmasının bir finansal hizmet uygulaması için güvenli bir seçenek olmaması gibi durumlar yaşanabilir.

Bu OWASP maddesi, kimlik doğrulama ile ilgili geniş kapsamlı güvenlik risklerini ele alır. Söz konusu güvenlik riskleri arasında;

➢Zayıf parola politikaları ve yetersiz parola karmaşıklığı,
➢Yetersiz hesap kilitleme eşikleri (lockout thresholds),
➢URL'lerde kimlik doğrulama (authentication) bilgilerinin açığa çıkması,
➢Düşük entropiye sahip kimlik doğrulama token'ları,
➢Sadece API anahtarlarının kimlik doğrulama (authentication) için kullanılması ve
➢Çok faktörlü kimlik doğrulama (MFA) eksikliği yer almaktadır.

Kimlik doğrulama mekanizmalarındaki güvenlik açıklarını başarıyla istismar eden saldırganlar, kullanıcı hesaplarını ele geçirebilir, yetkisiz erişim sağlayabilir, başkalarının kimlik bilgileriyle işlem yapabilir ve güvenilir bağlantıları kötüye kullanarak daha büyük ölçekli saldırıları gerçekleştirebilir.

API3:2023 – Kırık Nesne Özellik Düzeyinde Yetkilendirme (Broken Object Property Level Authorization - BOPLA)

Bazı geliştirme ekipleri, API'leri tasarlarken arka uç (back-end) sistemlerin bir işlev için gerekli olabilecek tüm veriyi istemciye sağlamasını ve istemci tarafı kodunun ise bu verileri uygun şekilde filtrelemesini bekler. Ancak bu yaklaşım, hassas veya yetkisiz verilerin istemciye döndürülmesine ve saldırganlar tarafından keşfedilmesine yol açabilir. API'ler, birçok farklı tüketici (consumer) türü tarafından kullanıldığından, arka uç (back-end) mühendisleri genellikle veri hassasiyetini veya gizliliğini göz önünde bulundurmadan API'leri geniş veri setleri sunacak şekilde tasarlar.

Geleneksel güvenlik tarama araçları (security scanning tools) ve çalışma zamanı analizi çözümleri (runtime analysis solutions), API'lerden dönen meşru veriler ile istemciye döndürülmemesi gereken hassas veriler arasındaki farkı ayırt edemez. Bu nedenle, kırık nesne özellik düzeyinde yetkilendirme (BOPLA) açıklarını belirlemek, uygulama tasarımının ve API'lerin bağlamının (API context) derinlemesine anlaşılmasını gerektirir.

Kırık nesne özellik düzeyindeki (BOPLA) güvenlik açığının istismar edilmesi oldukça kolaydır. Saldırganlar, API trafiğini analiz ederek istemci arayüzünde görüntülenmeyen ancak istemciye döndürülen hassas verileri tespit edebilir.

Bunu gerçekleştirmek için:

➢Ara vekil (proxy) araçları kullanarak API trafiğini dinleyebilirler. (Örneğin, BurpSuite veya OWASP ZAP gibi araçlarla.)

➢API yanıtlarını inceleyerek, istemci arayüzünde gösterilmeyen ancak döndürülen verileri keşfedebilir.

➢Yetkisiz nesne özellikleri üzerinde değişiklik yaparak API'ye gönderilen istekleri manipüle edebilirler.

Bu güvenlik açığı, saldırganların yetkilendirme kontrollerini aşarak hassas bilgilere erişmesine, yetkisiz değişiklikler yapmasına ve hatta ayrıcalık yükseltme (privilege escalation) saldırıları gerçekleştirmesine neden olabilir. Güvenli bir API tasarımı için; arka uç (back-end) mühendisleri tarafından sadece yetkili kullanıcılara ilgili nesne özelliklerinin döndürülmesi yanında sunucu tarafında sıkı erişim kontrolleri sağlanmalıdır.

API4:2023 - Sınırsız Kaynak Tüketimi (Unrestricted Resource Consumption)

API'ler, çeşitli hizmetleri sağlamak için ağ bant genişliği (bandwidth), işlemci (CPU), bellek (RAM) ve depolama gibi kaynaklara dayanır. Bu kaynakları tüketirken bazı API'ler; her bir istek için ücret talep edilebilen e-posta, SMS, telefon aramaları gibi hizmetleri de sağlayabilir. Ancak, API'lerin tasarımında bu kaynakların sınırsız bir şekilde tüketilmesine izin veren açıklar, ciddi güvenlik ve performans sorunlarına yol açabilir. Bu tür bir güvenlik açığı, saldırganların API'leri kötüye kullanarak aşırı kaynak tüketimi (resource exhaustion) sonucunda sistemin çökmesine neden olabilir. Aynı zamanda, API'lerin sağladığı hizmetlerin maliyetini de artırarak operasyonel sorunlara neden olabilir.

Sınırsız Kaynak Tüketimi (Unrestricted Resource Consumption) güvenlik açığı, kaynak sınırlamaları ve denetimlerinin eksik olduğu durumlarda ortaya çıkar. Kötü niyetli kullanıcılar, API'leri manipüle ederek hizmetin gereksiz kaynaklarını tüketebilir. Örneğin, API'lere yoğun veri veya işlem talepleri göndererek sistemin performansını etkileyebilir ve bu yolla hizmet dışı bırakma (DoS) saldırıları gerçekleştirebilir. Özellikle, API'lerin sunduğu hizmetler için harici entegrasyonlar yapılıyorsa (örneğin, SMS veya e-posta gönderimi gibi), bu tür aşırı yüklenmeler, yalnızca hizmetin kesintiye uğramasına değil, aynı zamanda yüksek maliyetli başka sonuçlara da neden olabilir.

Bu tür saldırıların başarılı olabilmesi için, API'lerin kaynak kullanımı konusunda kısıtlamalar getirmemesi veya bu kısıtlamaların düzgün bir şekilde uygulanmaması gerekir. Kötü niyetli bir kullanıcı, API'lerin belirlediği kaynak sınırlarını aşarak, daha fazla işlem yapabilir ve bu da sistemin aşırı yüklenmesine yol açabilir. Kaynak yönetimi (resource management) olmadan, API'lerin sunduğu her bir hizmet için yapılan taleplerin sayısı ve büyüklüğü kolayca yükselebilir ve bu da hizmet sağlayıcılarının ekstra maliyetler üstlenmesine neden olabilir.

Sınırsız kaynak tüketiminin engellenebilmesi için API tasarımında; kaynak sınırlamaları, kısıtlama (throttling) ve hız sınırlamaları (rate limiting) gibi mekanizmaların etkin bir şekilde uygulanması gerekir. Kaynaklar üzerinde sınırlandırmalar getirilmesi, aşırı yüklenmeyi engeller ve API'lerin yalnızca yetkili kullanıcılara uygun kaynakları sunmasını garanti eder. Ayrıca, API kullanımının izlenmesi ve olağandışı taleplerin tespit edilmesi; hızlı bir şekilde anomali oluşturan durumlar karşısında müdahale edilmesini sağlayabilir.

API5:2023 – Kırık Fonksiyon Düzeyi Yetkilendirme (Broken Function Level Authorization - BFLA)

Kırık fonksiyon düzeyi yetkilendirme hatası (BFLA), nesne düzeyindeki yetkilendirme açıkları (BOLA) ile benzerlik gösterir; ancak burada hedef BOLA'da olduğu gibi API'lerin etkileşimde bulunduğu nesneler değil BFLA'da API işlevleridir. Kötü niyetli aktörler, API'leri hedef alırken BOLA ve BFLA'yı istismar ederek dikey (vertical) ya da yatay (horizontal) ayrıcalık yükseltme (privilege escalation) saldırıları yapmaya çalışırlar.

BFLA güvenlik açıkları, API çağrılarının (API call) genellikle yapılandırılmış ve tahmin edilebilir olması nedeniyle saldırganlar tarafından kolayca keşfedilebilir. API dokümantasyonu ya da şema tanımları olmadığı takdirde, istemci tarafı kodunun tersine mühendislik (reverse engineering) çalışmaları ve uygulama trafiğinin izlenmesi yoluyla zayıf API uç noktalarını (endpoints) bulmak mümkündür. Bazı API uç noktaları; normal ve ayrıcalıksız kullanıcılara açılmış olabilir, bu da BFLA zafiyetlerinin saldırganlar tarafından daha kolay keşfedilmesine yol açar.

Saldırganlar, BFLA zafiyetlerini istismar etmek için yetkili olmadıkları bir API uç noktasına meşru API istekleri gönderir veya istemci uygulamalarından gelen API isteklerini yakalayarak ve değiştirerek saldırıyı gerçekleştirir. Örneğin, saldırganlar, HTTP metodunu GET'ten PUT'a değiştirebilir. Alternatif olarak, bir sorgu veya mesaj gövdesi (message body) parametresini de değiştirebilirler. Örneğin, "users" yerine "admins" yazarak yetkisiz erişim sağlamaya çalışabilirler. Saldırganlar, BFLA zafiyetlerini istismar ederek yetkisiz kaynaklara erişebilir, hesapları ele geçirebilir, değişiklik yapabilir veya ayrıcalıklarını yükseltebilirler.

API6:2023 - Hassas İş Akışlarına Sınırsız Erişim (Unrestricted Access to Sensitive Business Flows)

API'ler, birçok farklı iş akışını (workflows) destekleyebilir, bu da onları hassas verilerin işlenmesi için kritik hale getirir. Ancak, eğer API'ler uygun güvenlik önlemleri olmadan tasarlanırsa, saldırganlar iş akışlarına (workflows) erişebilir ve bu akışları kötüye kullanabilir. API6:2023, API'lerdeki hassas iş akışlarına (workflows) sınırsız erişim imkânı sunan güvenlik açığından bahseder. Bu açık, özellikle iş süreçlerinin yanlış yapılandırılması (misconfiguration) veya zayıf erişim kontrolü (weak access control) ile ortaya çıkar.

İş akışları (workflows), genellikle bir organizasyonun finansal işlemlerinden müşteri verilerine kadar birçok kritik bilgiyi içerir. Bu akışların, yetkisiz kişiler tarafından erişilmemesi veya manipüle edilmemesi çok önemlidir. Eğer API'ler, kullanıcı kimliğini veya rolünü doğrulamadan hassas iş akışlarını (workflows) doğrudan çağırabiliyorsa, saldırganlar bu açığı kullanarak sistemin güvenliğini ihlal edebilirler. Örneğin, saldırganlar, API'lere doğrudan erişim sağlayarak, yetkisiz işlemler başlatabilir, finansal verileri değiştirebilir hatta kimlik doğrulama (authentication) bilgilerini bile çalabilirler.

Bu tür bir güvenlik açığının en yaygın nedenlerinden biri, API'lerde yeterli erişim denetiminin ve rol tabanlı yetkilendirme (RBAC) mekanizmalarının yer almamasıdır. API'nin kritik iş akışlarına (workflows) erişimi kontrol etmek için kullanıcı rolü, oturum durumu veya erişim seviyeleri göz önünde bulundurulmalıdır. Ayrıca, API uç noktalarında, işlemler için daha fazla doğrulama (örneğin, multi-factor authentication - MFA) ve işlem bazında kontrol mekanizmaları eklenmesi önemlidir.

API'lerde hassas iş akışlarına (workflows) sınırsız erişim, sadece doğrudan veri sızdırılmasına yol açmaz, aynı zamanda organizasyonun itibarına ciddi zararlar verebilir ve hukuki sonuçlar da doğurabilir. Söz konusu açıkların istismarı, genellikle kullanıcıların yetkileri dışında verilere erişim sağlayarak, finansal kayıplar, veri hırsızlığı ve gizlilik ihlallerine neden olabilir.

Bu tür güvenlik açıklarının önlenmesi için;

➢API'lerde güçlü rol tabanlı erişim kontrolü (RBAC) ve en az ayrıcalık (least privilege) ilkesi uygulanmalıdır.

➢Ayrıca, API çağrılarında işlemlerin doğruluğunu ve yetki durumunu kontrol eden güvenlik katmanları oluşturulmalıdır. Bu, yalnızca yetkili kullanıcıların, doğru rol ve izinlere sahip olduklarında hassas iş akışlarına (workflows) erişmesini sağlar.

➢API uç noktalarındaki erişim kontrolü (access control), sadece kullanıcı doğrulama (authentication) ve yetkilendirmeyle (authorization) değil, aynı zamanda doğası gereği kritik olan belirli iş akışlarını (workflows) anlayarak daha kapsamlı bir denetim mekanizması ile güçlendirilmelidir.

API7:2023 - Sunucu Tarafı İstek Sahteciliği (Server Side Request Forgery - SSRF)

Sunucu Tarafı İstek Sahteciliği (SSRF), bir API'nin, istemciden aldığı verileri kullanarak sunucu tarafında dış kaynaklara istek (request) göndermesini sağlayan bir güvenlik açığıdır. Saldırganlar, API'lere kötü niyetli bir istek (request) göndererek sunucuya, iç ağlara (internal networks) veya dış kaynaklara istenmeyen istekler (request) göndermesini sağlayabilir. Bu, sunucunun dışarıya kapalı sistemlere veya hassas iç ağlara erişmesine yol açabilir.

SSRF, genellikle API'nin dış kaynaklara erişim sağlaması gerektiğinde ancak doğru bir doğrulama (validation) ve filtreleme (filtering) yapılmadığında ortaya çıkar. Örneğin, bir API'nin bir URL'yi alıp, o URL'ye istek (request) göndermesi gerekiyorsa ve bu isteklerde herhangi bir kısıtlama yoksa, saldırgan zararlı URL'ler göndererek iç ağdaki kaynaklara (internal resource) veya dış sistemlere (external system) erişim sağlayabilir. Bunun sonucu olarak, saldırganlar veritabanı (database) sunucularına, yerel dosyalara veya diğer dahili servislere erişebilir, güvenlik açıklarından faydalanarak yetkisiz işlem yapabilir.

Bu güvenlik açığının önlenmesi için, API uç noktalarındaki URL doğrulama (URL validation) ve filtreleme (URL filtering) işlemleri sıkı bir şekilde yapılmalıdır. Dışarıya açılan istekler (request), yalnızca güvenilir ve belirli IP adreslerine veya URL'lere yapılmalıdır. Ayrıca, sunucunun yalnızca gerekli olan kaynaklara erişimine izin verilmesi, SSRF saldırılarının etkisini önemli ölçüde azaltır.

API8:2023 - Güvenlik Yapılandırma Hataları (Security Misconfiguration)

Güvenlik Yapılandırma Hataları, API'lerin güvenlik ayarlarının hatalı yapılandırılması sonucu ortaya çıkan bir güvenlik açığıdır. API'lerin doğru şekilde yapılandırılmaması, sistemin farklı katmanlarında (sunucu, uygulama, veritabanı, ağ vb.) güvenlik açıklarına yol açabilir. Bu tür hatalar, çoğu zaman varsayılan yapılandırmaların (default configurations) veya yanlış konfigüre edilmiş izinlerin kullanılmasından kaynaklanır. Ayrıca;

➢Eksik güncellemeler (missing updates),

➢Gereksiz açık portlar (unnecessary open ports),

➢Aşırı yetkili kullanıcılar (over-privileged users),

➢Zayıf şifreleme yöntemleri (weak encryption methods) de güvenlik yapılandırma hataları (security misconfiguration) arasında yer alır.

Güvenlik yapılandırma hataları, API'lerin kötüye kullanılmasına, veri sızıntısına (data exfiltration), sistemin çökmesine veya yetkisiz erişimlere yol açabilir. Örneğin, bir API'nin arka uç (back-end) bileşenleri, yanlış yapılandırılmış bir güvenlik duvarı (firewall) nedeniyle dış dünyaya açılabilir, ya da API uç noktaları, istemci tarafından erişilmemesi gereken verilere izinsiz erişim sağlayabilir. Ayrıca, varsayılan parolalar (default passwords) veya zayıf kimlik doğrulama (weak authentication) yöntemleri, saldırganlar için kolay hedefler oluşturur. Bu tür hataların önlenmesi için, her bileşenin güvenlik yapılandırmalarının dikkatlice denetlenmesi gerekmektedir.
Özellikle, API'lerin yapılandırılması sırasında;

➢ Güvenlik duvarı kuralları (firewall rules),

➢ Şifreleme yöntemleri,

➢ Kullanıcı izinleri,

➢Kimlik doğrulama (authentication) mekanizmalarının düzgün şekilde uygulanması önemlidir.

Ayrıca;

➢API uç noktalarındaki (endpoints) gereksiz hizmetlerin devre dışı bırakılması,

➢Varsayılan ayarların değiştirilmesi,

➢Düzenli güvenlik taramaları yapılması da bu tür güvenlik açıklarının azaltılmasına yardımcı olacaktır.

API9:2023 - Uygunsuz Envanter Yönetimi (Improper Inventory Management)

Uygunsuz envanter yönetimi; envanter verilerinin yanlış bir şekilde işlenmesi sonucu ortaya çıkan bir güvenlik açığıdır. Bu tür bir açık, API'ler yoluyla stok bilgilerinin veya diğer envanter verilerinin doğru bir şekilde yönetilememesi nedeniyle saldırganların envanter üzerinde yetkisiz değişiklikler yapmasına olanak tanıyabilir. Bu, hem finansal kayıplara hem de müşteri güven düzeyine ciddi zararlar verebilir.

Yanlış envanter yönetimi genellikle, API'lerin veri doğruluğunu (data accuracy) sağlamadan veya yeterli erişim kontrolleri uygulamadan envanter bilgilerini işlemelerinden kaynaklanır. Örneğin, bir API, kullanıcıların stok durumlarını sorgularken, yalnızca doğru erişim izinlerine sahip kullanıcıların bu verilere erişimini sağlamıyorsa, saldırganlar bu veriye erişebilir ve stok bilgilerini manipüle edebilirler. Ayrıca, API'lerin doğru envanter verilerini sunmaması, hatalı siparişlerin işlenmesine ve yanlış ürünlerin müşterilere gönderilmesine de neden olabilir.

Bu tür açıkların önlenmesi için;

➢Envanter verilerinin sadece yetkilendirilmiş (authorized) ve doğrulanmış (authenticated) kullanıcılar tarafından erişilebilir olması sağlanmalıdır.

➢API uç noktalarında, veri doğruluğu (data accuracy) için sıkı kontrol mekanizmaları ve gereksiz verilerin sızmasını engellemek amacıyla filtreleme yapılmalıdır.

➢Ayrıca, API ile yapılacak her işlemde, işlemi gerçekleştiren kişinin yetkilerinin uygun olup olmadığı doğrulanmalıdır.

➢Envanter yönetimi ile ilgili tüm süreçler, doğru rol tabanlı erişim kontrolü (RBAC) ve güvenli veri işleme yöntemleri ile desteklenmelidir.

API10:2023 - Güvensiz API Tüketimi (Unsafe Consumption of APIs)

Güvensiz API Tüketimi, bir uygulamanın, dış kaynaklardan (externals) veya 3. taraflardan (third parties) aldığı API verilerini güvenli bir şekilde işlememesi durumunda ortaya çıkan bir güvenlik açığıdır. API'ler, farklı sistemler arasında veri paylaşımı ve etkileşimi sağlamak için yaygın olarak kullanılsa da, bu verilerin güvensiz bir şekilde işlenmesi, bir dizi güvenlik riskine yol açabilir. Özellikle, verilerin doğrulanmadan veya uygun güvenlik önlemleri alınmadan paylaşılması saldırganların API'leri kötüye kullanmasına olanak tanıyabilir.

Güvensiz API tüketimi, özellikle dış API'ler veya üçüncü taraf (third party) servislerle entegre olan uygulamalarda sıkça görülebilir. Bir uygulama, harici bir API'den aldığı verileri doğru şekilde doğrulamaz, sanitize edilmez veya filtrelenmez ise, saldırganlar bu verileri manipüle edebilir ve uygulamanın işleyişine zarar verebilir. Örneğin, zararlı veriler API aracılığıyla uygulamaya girebilir ve SQL enjeksiyonu, XSS (Cross-Site Scripting) veya diğer web tabanlı saldırılara zemin hazırlayabilir.

Bu tür açıkların önlenmesi için şu adımlar atılmalıdır:

➢Dış API'lerden alınan tüm veriler, sıkı bir şekilde denetlenmeli, sanitize edilmeli ve filtrelenmelidir.

➢Kullanıcıdan veya harici kaynaktan gelen verilerde herhangi bir değişiklik veya manipülasyon riski olup olmadığı her zaman kontrol edilmelidir.

➢Güvenli API tüketimi için veri şifrelemesi, kimlik doğrulama ve yetkilendirme (authentication & authorization) işlemleri uygulanmalıdır.

➢Tüketilen API'ler arasında yalnızca güvenilir ve onaylanmış API'lerin kullanılması, bu tür güvenlik açıklarını önemli ölçüde azaltır.

➢Üçüncü taraf API'lerden gelen tüm veriler, kullanılmadan veya görüntülenmeden önce mutlaka doğrulanmalı ve sanitize edilmelidir.

➢Uygulama ile API arasındaki bağlantı güvenli hale getirilmelidir. Her zaman HTTPS kullanılmalı ve SSL sertifikalarının geçerliliği kontrol edilerek ortadaki adam (man-in-the-middle) saldırılarına karşı koruma sağlanmalıdır.

➢Uygulamanın sorunsuz çalışabilmesi için uygun zaman aşımı (timeout) ve istek sınırı (rate limiting) ayarları yapılandırılmalıdır. Bu uygulamalar, API'lerin güvenli bir şekilde tüketilmesini ve hem uygulamanın hem de kullanıcıların güvenliğinin korunmasını sağlayan güçlü bir temel oluşturur.

➢API etkinliklerinin izlenmesi ve günlüklenmesi zorunludur. API istekleri ve yanıtları kaydedilmeli, olağan dışı kalıplar veya potansiyel güvenlik tehditleri büyük bir soruna dönüşmeden önce tespit edilmelidir.

➢Üçüncü taraf entegrasyonlar ve API'ler düzenli olarak gözden geçirilmelidir. Bu, saldırganlardan önce zafiyetlerin tespit edilmesini sağlayabilir.

Otomatik Saldırı Kalıplarını Tanıyalım

Saldırganlar, API saldırılarını gerçekleştirmek ve otomatize etmek için genellikle özel yazılmış kod, Python betikleri, komut satırı betikleri, hazır botlar ve iletişimi engelleyen veya değiştiren proxy araçları kullanırlar. Saldırganlar, API'lere entegre edilen özel iş mantığı (business logic) zafiyetlerini istismar ederek yeni saldırı yöntemleri geliştirir.

Bu bölüm, tüm sektörleri etkileyen otomatize üç yaygın saldırı yöntemini ele almaktadır:

➢Kimlik bilgisi doldurma (credential stuffing),
➢Web kazıma (web scraping) ve
➢Hesap ele geçirme (account takeover - ATO).

Kaba Kuvvet (Brute-force), Kimlik Bilgisi Doldurma (Credential Stuffing) ve Hesap Ele Geçirme (Account Takeover) Saldırıları

Kaba kuvvet saldırılarında (Brute-force attacks), saldırganlar harf ve rakam kombinasyonlarını sistematik olarak deneyerek geçerli kullanıcı adı ve parola çiftlerini bulmaya çalışır. Kaba kuvvet saldırıları genellikle bir kullanıcı adını birden fazla parola ile deneme yöntemiyle gerçekleştirilir. Saldırganlar, başlangıçta sahip oldukları bilgi seviyesine bağlı olarak, kullanıcı isimlerini de kaba kuvvet yöntemiyle tahmin etmeye çalışabilir.

Kimlik bilgisi doldurma saldırısı (Credential stuffing attack), önceden ele geçirilmiş kullanıcı adı/parola listelerini kullanarak oturum açmaya çalışma yöntemidir. Bu saldırı türünde, kullanıcıların farklı servislerde aynı kimlik bilgilerini tekrar kullanma alışkanlığı istismar edilir.

Kaba kuvvet ve kimlik bilgisi doldurma yöntemlerinin nihai amacı Hesap Ele Geçirme (ATO) saldırısını gerçekleştirmektir. ATO saldırılarında, saldırganlar geçerli kimlik bilgilerini ele geçirerek sistemde yetkili bir kullanıcı gibi oturum açabilir. Kimlik doğrulama (authentication) aşaması başarıyla geçildikten sonra, saldırganlar hassas verilere veya yetkilendirme dahilinde belirli işlevlere erişim sağlayabilir ve yetkilerini daha da yükseltmeye çalışarak büyük çapta zararlara neden olabilir.

Kimlik Bilgisi Doldurma (Credential Stuffing) Saldırılarının Durdurulması

Kimlik bilgisi doldurma ve kaba kuvvet saldırıları, belirli sayıda başarısız giriş denemesinden sonra hesabı kilitleyen politikalar uygulanarak hafifletilebilir. Ancak, aşırı ve katı hesap kilitleme eşikleri belirlemek kullanıcı deneyimini de olumsuz etkileyebilir. Bir uzlaşma çözümü olarak, bazı organizasyonlar daha esnek bir kilitleme politikası uygular, örneğin bir saat içinde art arda on başarısız giriş denemesi yapıldığında hesabı kilitlemek gibi. Bu durumda başarısız giriş denemesi sayacı 60 dakika sonra sıfırlanır.

Ancak saldırganlar, bu gevşek eşik değerlerini kendi avantajlarına çevirerek giriş denemelerini duraklatır ve eşik değerleri ile sayaçların sıfırlanmasını bekleyerek saldırılarına devam edebilirler. Bu saldırı tekniği, çalışma zamanı (runtime) davranış analizinin API'lerin kötüye kullanımını tespit etmek ve önlemek için neden gerekli olduğunu ayrıca ortaya koymaktadır.

Veri Kazıma (Data Scraping) Salgını

Genel erişime açık (public) API'ler doğası gereği risk taşırlar, çünkü tasarım gereği anonim erişime izin verme eğilimdedir ve geleneksel erişim kontrol mekanizmaları genellikle bir lüks olarak görülür. Kullanıcı kaydı yapılmadan veya MFA gibi ek güvenlik adımları kullanılmadan güçlü kimlik doğrulama (authentication) ve yetkilendirme (authorization) mekanizması kurmak mümkün değildir. Ancak böyle bir kısıtlayıcı yaklaşım kullanıcı deneyimini ve hizmetlerin benimsenmesini olumsuz etkileyebilir.

Saldırganlar, bu tür gevşek erişim kontrollerine sahip kamuya açık API'lerden sıklıkla faydalanırlar. API'ler çok fazla veriyi açığa çıkarabilir ve hız sınırlamalarına (rate limits) sahip olmayabilir.

Bu tasarım hatalarının tamamı, belirli bir API uç noktasına uygulandığında, farkında olmadan hassas veya özel verileri açığa çıkaran yani sızdıran bir API (leaky API) ile karşı karşıya kalabilirsiniz. Saldırganlar, temel betikler (scripts) kullanarak API uç noktalarının kolayca dökümünü (enumeration) elde edebilir ve böylece büyük ölçekte verinin kazınmasına (scaping) yol açabilir.

Tehdit aktörleri, veri uzmanlarının anlamlı desenler (pattern) çıkarmak, verileri bir araya getirmek ve ilişkilendirmek (aggregation & correlation) için aynı veri analitiği araçlarını kullanarak büyük ölçekte ve hacimlerde verileri toplayabilir. Toplanan veri kümesindeki bilgilere bağlı olarak, saldırganlar; dolandırıcılık (fraud) faaliyetini sürdürmek, bireyleri sosyal mühendislik ve kimlik avı (phishing) saldırıları ile hedef almak veya kaba kuvvet saldırısı (brute-force) ile hesaplarını ele geçirmek (takeover) için söz konusu verileri kullanabilir.

API'LERİ NASIL GÜVENLİ HALE GETİREBİLİRİZ?

04

Geleneksel yaklaşımlar, API izleme ve gözlemlenebilirliği (monitoring & observability) için faydalı olabilir ve güvenliğin bazı yönleri için de işlevsellik sağlayabilir. Yeni nesil bazı uygulamalar ise API'lerin nasıl performans gösterdiğini anlamak ve kullanıcı deneyimi ile eğilimlerini ortaya çıkarmak için kullanılabilir. Ancak, API'lerin yaşam döngüleri boyunca güvenliğini sağlamak için yeni yaklaşımlara ihtiyaç vardır.

Bu bölüm, API güvenliğinde mimarinin önemini ve tek seferlik araçlar ile kontrollerin yarattığı sorunlardan kaçınmak için neden bir platform yaklaşımına (platform approach) ihtiyaç duyulduğunu ele almaktadır. Ayrıca, API'leri tüm yaşam döngüleri boyunca güvenli hale getirmek için en kritik yetenek grupları:

➢**Sürekli keşif (continuous discovery),**

➢**Koruma (protection) ve**

➢**İyileştirme (improvement)** adımlarıdır.

Bu bölüm ayrıca, tüm organizasyonların dikkate alması gereken bazı en iyi API güvenliği uygulamalarından (best practice) bahsetmektedir.

Güvenli Bir Mimari Niçin Gereklidir?

Geleneksel yaklaşımların bıraktığı API güvenlik duruşundaki (API security posture) boşluklar, özel olarak tasarlanmış API güvenliği ihtiyaçlarını doğurmuştur. API güvenliği, mühendislik ekipleri tarafından bir araya getirilen parçalanmış araçlarla ele alınamaz ve rastgele işletilemez. Bu yaklaşım, operasyonel zorluklara (operational challenges), ölçekleme problemlerine (scaling issues) ve güvenlik olaylarının meydana gelme olasılığının artmasına yol açar. Aşağıdaki bölümler, herhangi bir API güvenlik çözüm mimarisinin neden önemli olduğunu ve bir çözümde bulunması gereken temel nitelikleri açıklamaktadır.

Odak Noktası: API Mimarisi

Kuruluşunuz için değerlendirdiğiniz herhangi bir API güvenlik aracının, yeteneklerden oluşan bir platform olarak inşa edilmiş olması beklenir. API güvenlik stratejisi ise tam yaşam döngüsü yaklaşımını (full life cycle approach) gerektirir çünkü güvenlik sorunları, güvenlik açıkları, mantık hataları (logic errors) ve yanlış yapılandırmalar (misconfigurations), API tasarımı, geliştirme, dağıtım ve operasyonel süreçlerin farklı aşamalarında ortaya çıkabilir.

API güvenlik araçları, Büyük Veri (big data) teknolojilerini kullanarak geniş API telemetri verilerini toplamalı ve saklamalı, API trafiğini ilişkilendirmeli, bağlam (context) sağlamalı ve böylece hızlı saldırı tespitini sağmakla olay müdahale (incident response) süreçlerine güç katmalıdır. Ayrıca, bu araçların yapay zeka ve makine öğrenmesi (AI/ML) araçları kullanarak BT ekipleri için sürekli olarak anlamlı ve uygulanabilir sinyaller üretmesi sağlanmalıdır. Piyasada olgunlaşma süresi (time-in-market) de önemli bir faktördür, çünkü algoritmalar eğitim yoluyla zamanla gelişir ve daha fazla kullanıcı ve API çağrılarıyla (API call) oluşan ağ etkisi sayesinde veri kümelerini zenginleştirir.

Bir API Güvenlik Platformunda Bulunması Gereken Temel Özellikler

Değerlendirdiğiniz herhangi bir API güvenlik aracının otomasyon ve bulut ölçeğinde kapasitesi düşünülerek tasarlanmış olması gerekir. Gerçekçi bir bakış açısıyla bu tasarım, otomatik ölçeklenen altyapı (auto-scaling infrastructure) bileşenleri, bulut depolama ve bulut analitiği gibi bulutta doğan (cloud-born) teknolojilerden yararlanan bulut yerel (cloud-native) bir tasarım anlamına gelir. Bu yaklaşımda, API benimsenme oranı arttıkça kuruluşunuzun tüm ortamlarını destekleme olanağı da artar.

API güvenlik araçlarının sahip olması gereken temel mimari özellikler şunlardır:

➢**Çevreden Bağımsızlık (Environment Agnostic):** API güvenlik araçları, modern ve eski altyapıları destekleyebilmelidir; nerede barındırıldıkları fark etmeksizin çalışabilmelidir. Ayrıca, yük dengeleyiciler (load balancers), API ağ geçitleri (API gateways) ve web uygulama güvenlik duvarları (WAFs) gibi ağ bileşenleriyle entegre olabilmelidir.

➢**Ek Sunucu Aracıları (Agents) ve Proxy'lerden Bağımsızlık:** API güvenlik araçları, ek sunucu aracıları (server agents) veya ağ proxy'leri gerektirmemelidir. Aynı zamanda, trafiğe CAPTCHA veya JavaScript eklemek gibi istemci taraflı kodları kullanarak saldırıları durdurmaya çalışmamalıdır. Bu yaklaşımlar ön uç (front-end) performansında sorunlara yol açar ve doğrudan API iletişiminde etkisiz kalır.

➢**Bulut Tabanlı Depolama ve Veri Analitiği:** API güvenlik araçları, bulut tabanlı depolama ve Büyük Veri Analitiği (Big Data Analytics) kullanmalıdır. Bu yaklaşım, API davranışlarını ve tüketim (consumption) modellerini baz alacak kadar büyük veri kümelerini saklamanın, analiz motorlarını çalıştırmanın ve olası veri kayıplarını (data loss), gizlilik ihlallerini veya diğer güvenlik olaylarını belirlemenin tek yoludur.

➢**Yapay Zekâ (AI) ve Makine Öğrenmesi (ML) Tabanlı Analiz:** API güvenlik araçları, toplanan tüm veri ve telemetriyi analiz etmek ve anlamlı güvenlik sinyalleri üretmek için yapay zeka (AI) ve makine öğrenmesi (ML) kullanmalıdır. Makine destekli yaklaşımlar, API sorunlarını en iyi nerede hafifletmek (mitigation) gerektiğini veya hangi güvenlik kontrolünün en uygun olduğuna karar verme gibi tespit ve uygulama yeteneklerini güçlendirmek için kritik öneme sahiptir. Ayrıca, geleneksel yaklaşımlarda sıkça karşılaşılan yüksek yanlış pozitif oranlarını (false positives) azaltmaya yardımcı olur.

API güvenlik araçlarının, şifrelenmiş taşıma (encrypted transport) ortamlarında çalışmak üzere tasarlanmış olduğundan emin olun. Bazı uygulamalar, trafik inceleme (traffic inspection) mekanizmalarıyla görünürlüğün azalmasından dolayı etkinlik kaybı yaşayabilir.

Kataloğunuzu Geliştirin: API'lerin Sürekli Keşfi

API kataloğunuz, API geliştirme, API entegrasyonu ve üçüncü taraf (third party) API bağımlılıkları (API dependencies) nedeniyle sürekli olarak gelişecektir. DevOps uygulamaları, artan versiyon yayın hızıyla (release speed) bu süreci daha da karmaşık hale getirmektedir. API uç noktalarını (endpoints) ve parametrelerini sürekli olarak tanımlamalı, açığa çıkan hassas verileri sınıflandırmalı ve diğer API güvenliği faaliyetlerini desteklemek için API'lerinizi kataloglara ayırmalısınız.

Eğer organizasyonunuz bir API yönetim platformu (APIM) kullanıyorsa, bu platformda bulunan API kataloğu büyük olasılıkla eksiktir. Konfigürasyon yönetimi (configuration management) ve değerli varlık yönetimi (asset management) veritabanları genellikle ya çok eski ya da API bağlamından (API context) çok uzak kalabilmektedir.

Aşağıdaki bölümlerde, keşif ve kataloglama (discovery & catalog) için sahip olunması gereken teknolojik yetenekler ele alınmaktadır. Ayrıca bu keşif yeteneği, hangi tür hassas veya özel verilerin ifşa olduğunu vurgulamak için veri sınıflandırmasını (classification) da içermelidir.

İhtiyaç Duyulan Keşif Yeteneklerini Değerlendirme

Bir organizasyonda meydana gelen sürekli değişimler ve API ekosisteminin hızlı gelişimi nedeniyle API keşfi (API discovery) sürekli bir süreç olmalıdır. Belirli bir anlık görüntü (snapshot) almak, güvenlik kontrollerini uygulamaya koyduğunuz anda geçerliliğini kaybedecektir. İşte gerekli API keşif yetenekleri:

➢**Tüm ortamları ve API türlerini kapsamalıdır.** API güvenlik araçları, farklı ortam türleri genelinde API'ler hakkında otomatik olarak veri ve üst veri (metadata) toplamalıdır. Üçüncü taraf API tüketimi (API consumption) de dahil edilmelidir. API keşfi, yalnızca şema tanımlamalarına (schema definitions) değil gerçek trafik analizlerine de dayanmalıdır. Bu sayede tasarlanan API ile üretim ortamında (production) kullanılan API arasındaki sapmalar tespit edilebilir.

➢**Gölge API'leri (Shadow APIs) tanımlamalıdır.** API güvenlik araçları, bilinmeyen veya dokümante edilmemiş API'leri (shadow APIs) tespit edebilmelidir. Bu tespit gölge API uç noktaları (endpoints), API fonksiyonları ve parametreleri dahil olmak üzere kapsamlı olmalıdır. Operasyon ve güvenlik ekiplerinin radarına takılmamış olan bu bilinmeyen API kaynakları, saldırganlar tarafından tasarlanmış yeni saldırı vektörlerine (attack vectors) hedef olabilir.

➢**Zombi API'leri (Zombie APIs) tanımlamalıdır.** API güvenlik araçları, zombi API'leri yani eski veya kullanımdan kaldırılmış API'leri tespit edebilmelidir. Geniş ölçekte API geliştirme ve işletme süreçlerinde, eski API versiyonları ve kodları sistemde uzun süre varlığını sürdürebilir. Zombi uç noktaları (endpoints); hatalı veya güvenlik açıklarına sahip kodlar barındırabilir, gereğinden fazla veri veya işlevsellik açığa çıkarabilir, artık izlenmiyor ve üretim ortamına yönelik güvenlik önlemlerinden yoksun olabilir.

Birçok tarama aracı, IP adresi ve ana bilgisayar (host) bilgilerine odaklanır, ancak bu veriler API güvenliği için tek başına yeterli değildir. Etkili API keşfi (API discovery) ve katalog yapısı; API uç noktaları (endpoints), API işlevleri, yollar (paths) ve mesaj gövde yapıları (message body structures) gibi tüm uygun API üst verilerini (metadata) içermelidir.

Veri Sınıflandırmasını Sürece Dahil Etme

API'ler, hassas olabilecek verileri sıkça değiş tokuş eder. Dolayısıyla herhangi bir API güvenlik aracının; API parametreleri ve yüklerindeki hassas veri türlerini tanımlamasını ve olası veri ifşalarının (data exposure) farkına varabilmek için API uç noktalarını (endpoints) uygun şekilde etiketlemesini istersiniz.

Kişisel tanımlanabilir bilgiler (Personally Identifiable Information - PII) ve diğer veri türleri, çeşitli düzenlemelere tabidir. Örneğin: Hassas veriler; Sağlık Sigortası Taşınabilirlik ve Sorumluluk Yasası (HIPAA) tarafından tanımlanan ve Korunan Sağlık Bilgileri (PHI) ve Özel veriler; Genel Veri Koruma Yönetmeliği (GDPR) tarafından tanımlanan bilgi türleri gibi.

Hassas verileri tanımlamamak ve güvence altına alamamak, düzenleyici kurumlar tarafından ağır yaptırımlara, marka itibarının ciddi şekilde zedelenmesine ve müşteri kayıplarına yol açabilir. API keşfi (API discovery), güvenlik denetimlerini desteklemek ve API güvenlik önceliklendirmesini belirlemek için etkili bir araçtır.

Sürekli Olarak API'leri Koruma Yaklaşımı

Bölüm 2 de, API ağ geçitleri (API gateways) gibi aracı teknolojilerin API çalışma zamanı (runtime) korumasına nasıl katkı sağladığını ele almıştık. Ancak, bu tür aracılık (mediation) mekanizmaları tek başına kullanıldığında, API güvenlik duruşunuzda (API security posture) boşluklara sebep olabilir.

Geleneksel çalışma zamanı (runtime) güvenlik yaklaşımları, API dünyası için tasarlanmamıştır ve tam bir API bağlamı (API context) sağlamada yetersiz kalır. Bu nedenle, API saldırılarını erken tespit edebilen ve değişen API saldırı yüzeyine (attack surface) dinamik olarak uyum sağlayabilen API güvenlik yeteneklerine sahip olmanız gerekiyor.

Yaygın bir tehdit koruma yaklaşımı; API aracı (API mediation) noktalarının önüne ek proxy'ler koymaktır. Bu proxy'ler arasında yeni nesil güvenlik duvarları (Next-Generation Firewalls - NGFW) veya web uygulama güvenlik duvarları (Web Application Firewalls - WAF) bulunur. Ancak, bu yaklaşımlar gecikmelere (latency) neden olur ve çoğu API ağ geçidinin sunduğu mesaj inceleme (message inspection) yeteneklerinin ötesinde minimal veya hiç ek koruma sağlayamaz. Üstelik, bu güvenlik kuralları, benzersiz API iş mantığına (API business logic) göre tasarlanmamıştır ve tam bir bağlam sağlamada yetersiz kalır.

Aşağıdaki bölümlerde, gereksinim duymanız gereken tespit (detection) ve koruma (protection) yetenekleri hakkında detaylı bilgiler verilmektedir. API koruma çözümlerini seçerken ve uygularken her iki yönü de dikkate almalısınız.

İhtiyacınız Olan API Saldırı Tespit (Attack Detection) Yetenekleri

API saldırılarını hızlı ve erken bir aşamada tespit edebilen yeteneklere ihtiyacınız vardır. WAF'lar ve API ağ geçitleri, izole işlemlere (transactions in isolation) odaklanır, tam bir API dizisini (API sequence) analiz ederek geniş bir bağlam sunmaz. API yönetimi ve entegrasyon platformlarının bileşenleri olarak var olanlar da dahil olmak üzere API ağ geçitleri (API gateway); halihazırda aşırı yüklenmiş olabilecek API aracıları (API mediation) ve erişim denetimi uygulayıcılarıdır.

Şemaya bağımlı API güvenlik araçları, kırık nesne seviyesi yetkilendirme (BOLA) gibi belirli API saldırılarını tespit etmekte başarısız olabilir. API şeması tanımlarına güvenlik koruması sağlamak, çalışma zamanında (runtime) trafiği incelemenin göz ardı edilmesiyle bazı sınırlamalara yol açar.

Bazı kuruluşlar, saldırı tespiti (Intrusion Detection Systems - IDS), saldırı önleme (Intrusion Prevention Systems - IPS) ve güvenlik duvarı (Next-Generation Firewall - NGFW) çözümlerini API güvenliği için yeniden kullanmaya çalışabilir. Ancak, bu sistemler çok protokollü ve geniş yelpazede saldırı tespiti sağladığından, uygulama katmanında (application layer) ve API odaklı koruma sağlamakta yetersiz kalırlar. İhtiyacınız olan saldırı tespit yetenekleri şunlardır:

➢**Tüm Ortamları ve API Türlerini Kapsama Alma:** API güvenlik araçları, farklı ortam türleri genelinde API'ler hakkında otomatik olarak veri ve üst veri (metadata) toplamalıdır. Üçüncü taraf (third party) API tüketimi (API consumption) de dahil edilmelidir. API keşfi, yalnızca şema tanımlamalarına (schema definitions) değil gerçek trafik analizlerine de dayanmalıdır. Bu sayede tasarlanan API ile üretim ortamında kullanılan API arasındaki sapmalar tespit edilebilir.

➢**Davranış Analizi ve Anomali Tespiti:** API güvenlik araçları, API iş mantığını (business logic) ve davranışlarını programatik olarak ayrıştırarak (parse) API güvenlik duruşuna (security posture) etkilerini değerlendirmelidir. Araçlar, kullanıcı ve varlık davranış analitiği (UEBA) tekniklerini kullanarak API'lerin kötüye kullanımlarını ve otomatize saldırıları tespit edebilmeli, API tüketim (API consumption) desenlerinde (pattern) anormal sapmaları belirleyebilmelidir.

➢**Erken Saldırı Tanıma:** API güvenlik araçları, API saldırılarını erken ve hızlı bir şekilde sürekli olarak tespit edebilmelidir. Saldırganlar (attackers), pasif ve gizli bir şekilde API hedeflerini araştırırken erken keşif (reconnaissance) aşamasından geçer. Bu pasif analiz teknikleri (passive analysis techniques), genellikle meşru trafik gibi göründüğü için çoğu tespitten kaçınır. API güvenlik araçları, otomasyon komut dosyaları (automation scripts) ve tersine mühendislik (reverse engineering) araçlarından kaynaklanan API tüketim (API consumption) desenlerindeki (patterns) ince varyasyonları tespit edebilmelidir.

API Tehdit Koruma Yeteneklerinin Geliştirilmesi

Geleneksel yaklaşımların sunduğu API tehdit koruma yeteneklerinin ötesine geçmek kritik öneme sahiptir. Sadece API'leri test etmek veya güvenlik açıklarını tespit etmek yeterli değildir. Saldırganlar hassas verilerinizi sızdırmadan (data exfiltration) veya kuruluşunuza zarar vermeden önce API saldırılarını durdurmak istiyorsanız, daha kapsamlı bir yaklaşıma ihtiyacınız vardır. İşte sahip olmanız gereken koruma yetenekleri:

➢**OWASP API Security Top 10 Güvenlik Açıklarını İstismar Eden Saldırıları Durdurma:** API güvenlik araçları, OWASP API Security Top 10'da tanımlanan güvenlik açıklarının istismarlarını durdurabilmelidir (Bkz. Bölüm 3). Bu liste, BOLA hataları, BFLA hataları, kırık kimlik doğrulama, aşırı veri ifşası (data exposure), kaynak veya hız sınırlamaları eksikliği, yanlış yapılandırmalar (misconfigurations) ve kod enjeksiyon (code injection) gibi hataların istismarlarını içerir.

➢**API Mantığını Öğrenirken Kötü Amaçlı İstekleri Engelleme:** API güvenlik araçları, kuruluşunuza özel iş mantığını (business logic) öğrenirken API saldırılarını engellemeli veya en azından hafifletmelidir. Bazı API saldırıları, bir kuruluşun API'leri nasıl tasarladığına bakılmaksızın tespit edilip durdurulabilir. Bunlar arasında kod enjeksiyon (code injection) saldırıları ve aşırı API tüketimi (API consumption) yer alır.

➢**Kimlik Bilgisi Doldurma (Credential Stuffing) ve Kaba Kuvvet Saldırılarını (Brute-force Attack) Durdurma:** API güvenlik araçları, saldırganların hesap ele geçirme (Account Takeover - ATO) amacıyla gerçekleştirdiği otomatize saldırıları (automated attacks) engellemelidir (Bkz. Bölüm 3). Kimlik doğrulama (authentication) ve yetkilendirme (authorization) gerektiren herhangi bir API'ye sahip kuruluşlar için ATO riski bulunmaktadır. Ek kimlik doğrulama adımları kullanılsa bile, saldırganlar güçlü erişim kontrollerini aşmak için çeşitli teknikleri birleştirebilmektedir.

➢**Uygulama Katmanı Hizmet Reddi (DoS) Saldırılarını Durdurma:** API güvenlik araçları, uygulama katmanı hizmet reddi (Denial of Service - DoS) saldırılarını durdurabilmelidir. Hizmet reddi veya dağıtık hizmet reddi (Distributed Denial of Service - DDoS) saldırıları genellikle aşırı trafik veya istek (request) oranları ya da hacimsel saldırılar (volumetric attacks) açısından ele alınır. Bununla birlikte, daha gizli ve tehlikeli bir DoS türü, uygulama katmanı DoS (application-layer DoS) veya 7. katman (Layer 7) DoS saldırılarıdır. Uygulama katmanı DoS saldırıları, uygulama ve API'lerin benzersizliği nedeniyle durdurulması daha zordur. Güvenlik araçlarınızın yalnızca 3. katman ve 4. katman DoS saldırılarını durdurmakla kalmadığından emin olun. Araçlar özellikle API'ler için 7. katman DoS saldırılarını da kapsamalıdır.

API ağ geçitleri (API gateways) veya WAF'larda bulunan geleneksel hız sınırlama (rate limiting) ve mesaj filtreleme (message filtering) mekanizmaları genellikle çok statik, operasyonel olarak çok karmaşık veya yeterince iyi bakım yapılmamış olabilir. Eğer API tüketiminiz sınırlıysa statik sınırlar (static limits) kullanabilirsiniz. Ancak, API tüketicileriniz (API consumer) fazla sayıda ise veya trafik örüntüleri daha az tahmin edilebilir bir düzeydeyse dinamik sınırlandırma mekanizmaları (dynamic limiting mechanisms) kullanmayı tercih etmelisiniz.

API'leri İyileştirme Adımları: Yetenekleri Artırarak İş Akışını Kolaylaştırma

DevOps ve DevSecOps uygulamaları, geri bildirim döngüleri (feedback loops) kavramını güçlü hale getirmektedir. Bu uygulama geliştirme döngüsüne güvenlik araçlarınızın entegre ve otomatik hale getirilmesiyle iş akışlarınızın (workflow) sorunsuz olmasını arzu edersiniz. İş akışları genellikle "git" tabanlı versiyon kontrol sistemleri (git-based version control systems - VCS) ve işlem hatları (pipelines) etrafında şekillenir. Güvenlik ekipleri ve güvenlik dışı ekipler, harekete geçmek ve sorunları işbirliği ile çözmek için gerekli bilgilere hızlı bir şekilde erişim sağlayabilmelidir.

İyileştirme iş akışları (remediation workflows), normal çalışma akışlarını ve iş faaliyetlerini minimum düzeyde kesintiye uğratmalıdır. Bunu başarmak için en iyi yöntem, gerekli bilgileri ve çözüm yeteneklerini normal iş süreçlerinin bir parçası olan araç zincirlerine (toolchains) entegre etmektir.

Aşağıdaki bölümler, sahip olmanız gereken iyileştirme (remediation) yeteneklerine ve olay müdahale (incident reponse) iş akışlarına (workflows) uyum sağlamanın niçin gerekli olduğunu açıklamaktadır.

İhtiyacınız Olan İyileştirme (Remediation) Yetenekleri

Kuruluşlar sıklıkla güvenlik açığı taramaları (vulnerability scanning) sonucu üretilen CVE numaraları ile mücadele ederler. Ancak, tasarım kusurları (design flaws), yazılım zayıflıkları (software weaknesses) ve iş mantığı hataları (business logic flaws), CVE numaralarıyla doğrudan eşleşmez. API güvenliği açısından, geniş bir yelpazedeki API ile ilgili kusurları, güvenlik açıklarını ve altyapı yanlış yapılandırmalarını (infrastructure misconfigurations) tespit edebilen iyileştirme yeteneklerine sahip olmanız gerekmektedir. Bu iyileştirme yetenekleri API'lerin tam yaşam döngüsü boyunca geliştirme (development), derleme (compilation) ve çalışma zamanı (runtime) aşamalarında sürekli olarak çalışmalıdır.

Bir güvenlik iyileştirmesi her zaman kaynak kod düzeyinde olmayabilir, çünkü bir sorunu kodla çözmek teknik olarak mümkün olmayabilir, kod düzeyinde bir iyileştirmeyi zamanında devreye almak pratik olmayabilir veya sorunu kod içerisinde çözmek yerine diğer altyapı bileşenleri üzerinden hafifletmek (mitigation) daha uygulanabilir bir yol olabilir.

İhtiyacınız olan API iyileştirme yetenekleri şunlardır:

➢**API Güvenlik Açıkları ve Zafiyetlerin Tespiti:** API güvenlik araçları, organizasyondaki tüm API'lerin güvenliğini değerlendirmek için çeşitli teknikler kullanmalıdır. Araçlar, kuruluş mimarisinde hem şirket içi (on-premises) hem de bulut ortamlarında (off-premises) akan API trafiğini pasif olarak analiz etmelidir. Mümkün olduğunda API şema tanımlarını analiz etmeli ve geliştirme ekipleri, operasyon ekipleri veya her ikisi tarafından iyileştirilmesi gereken API zayıflıkları belirlenmelidir.

➢**Farklı Kullanıcı Profillerine Uygun İyileştirme Rehberliği:** API güvenlik araçları, yazılım geliştirme (development) perspektifi için kod düzeyinde iyileştirme önerileri sağlamalıdır. Ayrıca, operasyon perspektifi için altyapı yanlış yapılandırmalarına (infrastructure misconfiguration) karşı iyileştirme rehberliği sunmalıdır. Sorunlar, uygun olduğunda OWASP API Security Top 10 (bkz. Bölüm 3) ile eşleştirilmelidir. Ancak, teknik ayrıntılar yalnızca güvenlik uzmanlarına yönelik yazılmamalı, tüm ilgili paydaşlar tarafından anlaşılabilir olması sağlanmalıdır.

➢**Hata Takip Sistemleriyle Entegrasyon:** API güvenlik araçları, mevcut güvenlik ve uygulama geliştirme iş akışlarını (workflows) desteklemek için harici hata takip sistemleri (defect tracking systems) ile entegre olmalıdır. Hata takibi, kuruluşun BT ve güvenlik programlarına bağlı olarak harici DevOps paketlerinde, BT servis yönetimi (ITSM) araçlarında veya zafiyet yönetimi (Vulnerability Management -VM) platformlarında yürütülebilir.

➢**Kod Havuzu ve İşlem Hattı Entegrasyonu:** API güvenlik araçları, geliştirme (development), derleme (build) ve dağıtım (release) sistemleriyle entegrasyon sağlayacak mekanizmalar sunmalıdır. Bu entegrasyon, versiyon kontrol sistemleri (Version Control Systems - VCS) ile entegrasyon aracılığıyla API kodunun veya şema tanımlarının statik analizi ile sağlanabilir. CI/CD entegrasyonu aracılığıyla, API'lerin üretim öncesi (pre-production) veya üretim (production) ortamlarında çalışma zamanında (runtime) dinamik analizi yoluyla da gerçekleştirilebilir.

API'lere Yönelik Olay Müdahale Süreçlerini Uyarlama

API saldırıları kaçınılmazdır ve kuruluşunuz tehdit aktörleriyle birden fazla cephede mücadele etmek zorundadır. Çalışma zamanında (runtime) API'lerinizi korumak kritik öneme sahip olsa da, bir saldırı durumunda kuruluşunuzun müdahale kabiliyeti de aynı derecede önemlidir. Tüm API güvenlik riskleri doğrudan saldırı amaçlı olmayabilir. Örneğin, en büyük endişeler arasında bir saldırgan tarafından veri sızdırılması (data exfiltration) veya veri kazınma (scraping) olayı yer alabilir. Olay müdahale (incident response) rehberiniz, yalnızca saldırıları değil, istem dışı veri ifşası (data exposure), gizlilik ihlalleri ve erişilebilirlik sorunları gibi öngörülemeyen API olaylarını da kapsamalıdır.

API odaklı olay müdahale (incident response) yeteneklerine sahip olmanız gerekmektedir. Bu yetenekler, yazılım geliştirme (development) ve güvenlik operasyonları (SecOps) ekiplerinin iş akışları (workflows) ve araçlarıyla entegre olmalıdır. Kuruluşunuzun Güvenlik Bilgi ve Olay Yönetimi (SIEM) sistemine veri akışı sağlanmalıdır, ancak bu süreç akıllıca yönetilmelidir. Sadece temel bir log akışı veya veri dökümü (data dump) sağlamak yeterli değildir. API güvenlik araçları, olayları etkin bir şekilde önceliklendirmeli, eyleme geçirilebilir güvenlik uyarıları sunabilmeli ve modern Güvenlik Operasyon Merkezi (SOC) ve BT ekosisteminin iş akışlarını (workflows) desteklemelidir.

API Güvenliği İçin En İyi Uygulama (Best Practices) Örnekleri

API tasarım desenleri (patterns) ve API tüketici (consumer) türlerinin geniş yelpazesi, kuruluşunuz için güvenlik gereksinimlerini karmaşık hale getirir. API ekosisteminin çeşitliliği, standart en iyi bir uygulama (best practice) seti oluşturmayı zorlaştırır. Kuruluşların API güvenliği için belirlediği en iyi uygulama örnekleri, kapsayıcı olmalı ve çok çeşitli teknoloji bileşenlerini içermelidir.

API’ler, organizasyon içindeki birçok farklı rol tarafından kurgulanır, çalıştırılır veya kullanılır. Bunlar arasında uygulama geliştirme ve güvenliği ekipleri, API ürün ve operasyon ekipleri ile bunlardan bağımsız Siber Güvenlik operasyonları (SecOps) ekipleri yer almaktadır. DevOps uygulamalarında olduğu gibi, güvenli API'ler oluşturmak ve işletmek için iş birliği kritik bir faktördür. Aşağıdaki bölümler, benimsenmesi gereken çeşitli API güvenliği en iyi uygulama örneklerini detaylı olarak ele almaktadır.

API Keşfi (API Discovery) ve Kataloglama

Doğru bir API envanteri, BT’ye bakan birçok yönü itibariyle kuruluşunuz için kritik öneme sahiptir. Uyumluluk (compliance), risk ve gizlilik (privacy) ekipleri, özellikle düzenleyici otoritelere hesap vermek zorunda oldukları için API envanterine ihtiyaç duyarlar. Ayrıca Güvenlik ekiplerinin de saldırı yüzeyinizi (attack surface) ve risk duruşunuzu (risk posture) gerçeğe uygun bir şekilde değerlendirebilmek ve geniş API güvenlik faaliyetleri kapsamında tehditleri önceliklendirebilmek amacıyla API envanterine ihtiyaçları vardır.

API keşfine yönelik en iyi uygulamalar (best practice) şunlardır:

➢API’leri yalnızca üretim (production) ortamında değil, daha düşük ortamlar (lower environments) için de keşfedin. Daha düşük ortamlar genellikle daha gevşek güvenlik önlemlerine sahiptir ve birincil saldırı hedefleri olabilir.

➢API bağımlılıklarını (dependencies) ve üçüncü taraf (third party) API’leri API envanterinize dahil edin. Üçüncü taraf API’ler de saldırı yüzeyinin (attack surface) bir parçasıdır.

➢API’leri ve mikro servisleri (microservices) etiketleyin (tagging). Bu, DevOps’un en iyi uygulama örneklerinden biri olup, API yaşam döngüsünün (life cycle) birçok diğer aşaması için de kolaylaştırıcı bir rol oynar.

API Güvenlik Testi

Geleneksel tarama teknolojileri, özel olarak geliştirilmiş kodları ve iş mantığını (business logic) analiz etmekte zorlanır, çünkü tasarım desenleri (patterns) ve kodlama pratikleri geliştiricilere göre değişiklik gösterir. Geleneksel güvenlik test araçlarını kullanarak API uygulamanızdaki belirli unsurları doğrulayabilirsiniz, ancak bu araçlardaki sınırların farkında olmalısınız. Bu araçlar; yaygın yanlış yapılandırmaları (misconfigurations), güvenlik açıklarını (vulnerabilities) ve istismar edilebilir (exploitable) koşulları tespit etmek için kullanılabilir.

Güvenlik testi için en iyi uygulama örnekleri şunlardır:

➢API kodunuzu, versiyon kontrol sistemine (VCS) kod gönderildiğinde, CI/CD süreçlerinde derlendiğinde ve dağıtıldığında (compilation & delivering) iyi tanımlanmış istismar edilebilir koşullara karşı statik olarak analiz edin.

➢API kodunuzda bilinen güvenlik açıklarına sahip üçüncü taraf bağımlılıklarını (third-party dependencies) ve açık kaynak bileşenlerini (open-source component) kontrol edin.

➢Devreye alınan (deployment) API'leri dinamik olarak analiz edin ve API fuzz testi uygulayarak tamamen entegre sistemde istismar edilebilir koşulları tespit edin

API Aracılığı (API Mediation) ve Mimarisi

API aracılığı (API mediation); gelişmiş görünürlük (visibility), hızlandırılmış dağıtım (accelerated delivery), artırılmış operasyonel esneklik (operational flexibility) ve gelişmiş uygulama yetenekleri (enforcement capability) sağlar. Son madde genellikle API erişim kontrolünü (access control) uygulamak için kullanılır. Bir kuruluş, API aracılığını genellikle ters proxy (reverse proxy), ileri proxy (forward proxy) veya her ikisi gibi çalışan API ağ geçitleri (gateways) ve mikro ağ geçitlerini (micro gateways) dağıtarak sağlayabilir.

API aracılığına (API medation) yönelik en iyi uygulama örnekleri şunlardır:

➢API'lerinizi aracılık (mediation) mekanizmaları ile destekleyerek iç (inner) ve dış (outer) API'ler için izleme ve gözlemlenebilirlik (monitoring & observability) yeteneklerini geliştirin.

➢API erişim kontrolünü (access control), hız sınırlamayı (rate limiting) ve mesaj filtrelemeyi (message filtering) uygulamak için API ağ geçitleri (gateway) gibi aracılık mekanizmalarını kullanın.

➢Statik ve dinamik kontrol arasındaki farkı anlamlı hale getirecek API güvenlik araçları ile aracılık (mediation) mekanizmalarını güçlendirin.

Modern Ağ Güvenliği Yaklaşımı

Kuruluşlar, yüksek düzeyde dağıtılmış API'lere ve bulut hizmetlerine geçiş yaptıkça geleneksel ağ sınırları (network perimeters) giderek zayıflamaktadır. Altyapılar (infrastructure) daha geçici (temporary), sanallaştırılmış (virtualized) ve konteynerleştirilmiş (containerized) hale gelmektedir. Bu dönüşüm, bazı geleneksel ağ erişim kontrolü (network access control) yaklaşımlarını etkisiz hale getirmektedir. Modern ağ güvenliği, kimlik ve erişim yönetimi (IAM) ile daha fazla kesişmeye başlamaktadır. Bu yeni konsept, "Sınırları kimlik ile tanımlama" (identity as the perimeter) yaklaşımıyla da ifade edilir.

Ağ güvenliğine yönelik en iyi uygulama örnekleri şunlardır:

➢ API'lerinizin korunmasız ağlar üzerinden ilettiği verileri korumak için şifrelenmiş taşımayı (encrypted transport) etkinleştirin.

➢ API tüketicilerinizin (consumer) sayısı sınırlıysa (örneğin iş ortakları veya tedarikçi entegrasyonları gibi durumlarda), IP adresleri için temiz ve kara listeler (Whitelisting & Blacklisting) hazırlayın.

➢ API tüketicileri (consumer) çok sayıda veya öngörülemezse, API dağıtımlarında dinamik hız sınırlamayı (rate limiting) kullanmayı düşünün.

Veri Güvenliği

Verileri güvence altına almak için uygun teknikler arasında maskeleme (masking), tokenizasyon (tokenization) ve şifreleme (encryption) bulunur. Birçok veri güvenliği çalışması, arka uç (back-end) sistemlerde veri güvenliğine odaklanır; örneğin, veritabanı şifrelemesi veya alan düzeyinde şifreleme(field-level encryption) gibi. Ancak, bu şifreleme yöntemleri, bir saldırganın kimlik bilgileri veya yetkilendirilmiş (authorized) bir oturumun ele geçirilmesi (takeover) durumunda verilerin korunmasını sağlayamaz; çünkü API aracılığıyla erişildiğinde veriler saldırgana şifrelenmemiş şekilde sunulacaktır.

Veri güvenliği için en iyi uygulama örnekleri şunlardır:

➢ Şifrelemeyi yasal veya uyumluluk düzenlemelerinin gerektirdiği ölçüde ve seçici olarak kullanın, çünkü operasyonel karmaşıklık yaratabilir. Çoğu API kullanım durumda veriler için taşıma (transport) koruması yeterlidir.

➢ API çağrıcısına (callers) gereğinden fazla veri göndermekten kaçının ve veri filtreleme işleminin yalnızca istemci tarafında yapılmasına güvenmeyin. Hassas veya özel veriler her zaman ağ trafiğinde görünür hale gelebilir.

➢ Şifrelemenin etkili bir çözüm olmadığı veri kazıma (data scraping) veya veri çıkarımı (data inference) gibi modern tehditlere karşı önlemler alın.

Kimlik ve Erişim Yönetimi (IAM) ile Kimlik Doğrulama ve Yetkilendirme Mekanizmaları

Kimlik doğrulama (authentication) ve yetkilendirme (authorization) ile geniş anlamda kimlik ve erişim yönetimi (IAM), tüm güvenlik alanlarında olduğu gibi API güvenliği için de temel bileşenlerdir. Kimlik ve Erişim Yönetimi (IAM), hem işlevsellik hem de veri erişimi için yoğun olarak kullanılır. Kimlik doğrulama ve yetkilendirme uygulanırken yalnızca kullanıcı kimlikleri değil, makine kimlikleri de dikkate alınmalıdır. (Sıfır Güven (Zero Trust) yaklaşımını sonraki versiyonlarda ele almayı düşünüyoruz). Kullanıcılar için oturum sırasında ek kimlik doğrulama (authentication) adımları talep etmek mümkün olsa da, makine iletişiminde bu seçenek yoktur.

Kimlik doğrulama ve yetkilendirme için en iyi uygulama örnekleri şunlardır:

➢API tüketicilerini (consumer) oturum boyunca ve yalnızca giriş anında değil, davranışlarına göre sürekli olarak kimlik doğrulama (authentication) ve yetkilendirme (authorization) mekanizmalarıyla kontrol edin.

➢OpenID Connect (OIDC) ve OAuth2 gibi modern doğrulama ve yetkilendirme protokollerini kullanın.

➢API anahtarlarını (API keys) tek başına kimlik doğrulama mekanizması olarak kullanmaktan kaçının. API anahtarları esas olarak versiyon kontrolü (version control) için kullanılır ve mutlaka ek kimlik doğrulama yöntemleriyle desteklenmelidir.

Çalışma Zamanı (Runtime) Koruması

Çalışma zamanı (runtime) koruması, bazen tehdit koruması (threat protection) olarak da adlandırılır ve genellikle API ağ geçitleri (API gateways) ve Web Uygulama Güvenlik Duvarları (WAFs) gibi proxy'ler aracılığıyla sağlanır. Bu mekanizmalar, mesaj filtreleri (message filters) ve statik imzalara (static signatures) dayanır; belirli desenleri (pattern) takip eden bazı saldırı türlerini yakalayabilir, ancak çoğu API kötüye kullanım (API abuse) türünü tespit edemez. Çalışma zamanı (runtime) koruma mekanizmaları, API altyapısındaki yanlış yapılandırmaları (misconfigurations) tespit etmekte olduğu kadar, kimlik bilgisi doldurma (credential stuffing), kaba kuvvet saldırıları (brute-force) ve veri kazıma (data scraping) girişimleri gibi anomali içeren davranışları belirlemekte de faydalıdır.

Çalışma zamanı (runtime) koruması için en iyi uygulama örnekleri şunlardır:

➢JSON ve XML enjeksiyonu (injection) gibi zafiyetleri azaltmak için API ağ geçitleri (API gateway) ve API yönetim platformlarında (APIM) tehdit koruma özelliklerini etkinleştirin.

➢Hizmet Reddi (DoS) ve Dağıtılmış Hizmet Reddi (DDoS) saldırılarına karşı önlemler alın. Saldırganlar, bir API'yi istismar edemediğinde veya kötüye kullanamadığında genellikle bu tür saldırılara yönelir.

➢Önceden oluşturulmuş imzaların (pre-built signatures) yetersiz kaldığı yeni nesil API saldırılarını tespit etmek için geleneksel çalışma zamanı (runtime) kontrollerini Yapay Zeka (AI), Makine Öğrenmesi (ML) ve davranış analizi motorlarıyla (behavior analysis engines) destekleyin.

API'LERİNİZİ GÜVENCE ALTINA ALABİLECEĞİNİZ 10 ÖNCELİKLİ ÇÖZÜM

05

Kuruluşunuzda API güvenliğini sağlamak için ne yapmanız gerektiğinden emin değil misiniz? Aşağıda, API'lerinizi tanımlamak ve korumak için odaklanmanız gereken yüksek öncelikli 10 madde bulunmaktadır:

➢**API Güvenlik Liderlerini Belirleyin.** Eğer bir uygulama güvenliği ekibiniz varsa, işe buradan başlayarak API güvenliği liderlerini belirleyebilirsiniz. Bu kişiler; keşif, test, koruma ve olay müdahalesi (incident response) konularında diğer ekiplerle işbirliği yapmalıdır. API güvenliği uzmanlığı; geliştirme, altyapı, operasyon ve güvenlik rollerine dağılmış halde bulunabilir veya uzmanlığın API ürün ekiplerinde yoğunlaştığını görebilirsiniz.

➢**API'lere Özgü Çalışma Zamanı (Runtime) Korumasını Devreye Alın.** Saldırganlar düzenli olarak API'leri istismar etmeyi ve kötüye kullanmayı hedefler. Veri ifşaları (data exposure) ve gizlilik ihlalleri en az diğer saldırılar kadar, hatta daha fazla kuruluşunuza zarar verebilir. API davranışlarını gerçek zamanlı olarak analiz edebilen ve saldırganları erken aşamada tespit edip durdurabilen koruma çözümleri araştırın ve zaman kaybetmeden devreye alın.

➢**API'ler İçin Olay Müdahale Süreçlerini Uyarlayın.** Dijital adli analiz (forensics) ve olay müdahale (incident response) süreçlerinizi API dünyasına uygun hale getirin. API kötüye kullanımı ve veri sızıntıları (data exfiltration), başlangıçta en öncelikli tehditler arasında görülmeyebilir, ancak başarılı API saldırıları organizasyonlar için büyük olumsuz etkilere yol açar. SecOps ekiplerinin uygun API uzmanlarının sürece dahil edilerek hızlı bir şekilde olaylara müdahale edebilmesi için gerekli araç setlerini sağladığınızdan ve süreçleri olgunlaştırdığınızdan emin olun.

➢**API İyileştirme (API Remediation) Süreçlerini Tanımlayın.** API'leri keşfederken, test ederken ve korurken, API'leri savunmasız bırakan kod seviyesindeki hatalar veya yanlış yapılandırmalarla (misconfigurations) karşılaşmanız kaçınılmazdır. İyileştirme adımlarınızı resmileştirin. Bu adımlar, DevOps'ta kritik olan geri bildirim döngülerini de destekleyecektir. İyileştirme süreci genellikle birden fazla rolün karışımını gerektirebilir ve üçüncü tarafları (third parties) de içerebilir.

➢**Bir API Envanteri Oluşturun.** Başlangıçta bir API envanterine sahip olmanız ve API manzaranız geliştikçe bunu sürdürülebilir hale getirecek bir planlamanızın olması elzemdir. API envanteriniz, yalnızca varlık yönetimi veritabanlarında veya API yönetim sistemlerinde (APIM) bulunanlardan daha fazlasıdır. API uç noktalarını (endpoints) tanımlayabilen, üstveri (metadata) toplayabilen ve potansiyel olarak ifşa olan veri türlerini sınıflandırabilen keşif mekanizmalarına ihtiyacınız olacaktır.

➢**Gölge (Shadow) ve Zombi (Zombie) API'leri Tespit Edin.** API envanteri ve API şeması tanımlamaları sizi yalnızca bir noktaya kadar götürür. Gölge (shadow) ve zombi (zombie) API'lerinizi sürekli olarak yerel ve bulut ortamlarında taramanız gerekecektir. Bu API'ler önemli bir güvenlik riski oluşturur ve saldırganlar tarafından hızla ortaya çıkarılıp istismar edilebilir.

➢**API'lerdeki Veri Türlerini Sınıflandırın.** API keşfi (API discovery), güvenlik kontrolünü nereye yerleştireceğinizi veya daha fazla izleme yapmanız gereken yerleri belirlemede ve potansiyel veri sızıntılarını tanımlamada faydalıdır. Veriler, yasal ve uyumluluk düzenlemelerine bağlı olarak hassas veya özel olarak sınıflandırılabilir. Bu tür API'lere özgü veri sınıflandırması, yönetişim, risk, uyum ve gizlilik gereklerini yerine getirmek için kritik öneme sahiptir.

➢**API Şemasını ve Kodunu Analiz Edin.** Hataları erken tespit etmek ve üretim ortamı öncesi (pre-production) dağıtımdan önce taramak, güvenlik açısından en iyi uygulamalar olarak teşvik edilir. Böyle bir yaklaşım, BT döngülerinde tasarruf sağlayabilir ve bir saldırganın istismar edilebilir koşullar bulma olasılığını azaltabilir. API şemalarını sürekli analiz edin, kaynak kodu yapılandırma (configuration) aşamalarında ve/veya dağıtılmış API'leri çalışma zamanında (runtime) istismar edilebilir koşullar için tarayın.

➢**Modern Mimariler Konusunda Yetkinlik Kazanın.** Monolitik ve mikro servisler, iç ve dış API'ler, ve bulut yerel (cloud-native) tasarım desenleri arasındaki farkları anlamak, API'leri en iyi şekilde keşfetmek, aracılık yapmak ve korumak için nereye odaklanmanız gerektiğini belirlemeniz için faydalı oalcaktır. Güvenlik yaklaşımınız, bir ağ perimetresini kontrol etmeye dayanamaz çünkü modern mimarilerde perimetre erimiştir.

➢**Bütünsel Bir API Güvenlik Stratejisi Oluşturun.** Uygulama güvenliği programınızı (eğer varsa) ağ ve altyapı unsurlarını içerecek şekilde genişletin. API yol haritalarını (roadmap), en iyi uygulamaları ve sorunları gözden geçirmek için güvenlik ve geliştirme ekipleriyle düzenli bir modülasyon oluşturun. Dokümante edin, süreçleri yineleyin ve zaman içerisinde iyileştirin.

KAYNAKÇA

➢Isbitski, M. (2022). *API Security: Salt Security Special Edition*. Wiley Publishing

➢ https://www.paloaltonetworks.com/cyberpedia/what-is-api-security

➢ https://nordicapis.com/how-a-leaky-api-can-kill-your-business/

➢ https://www.getambassador.io/blog/api-throttling-best-practices

➢ https://www.akamai.com/glossary/what-is-account-takeover-ato

➢https://bigid.com/blog/unified-tagging-and-labeling-for-structured-and-unstructured-data/

➢ https://docs.gitlab.com/user/application_security/api_fuzzing/

➢ https://blog.openreplay.com/step-by-step--url-validation-in-javascript/

➢https://www.moesif.com/blog/api-engineering/api-observability/What-is-the-Difference-Between-API-Observability-vs-API-Monitoring/

➢ https://blog.dreamfactory.com/ip-whitelisting-vs.-blacklisting-for-apis

➢https://cloud.google.com/blog/products/identity-security/protecting-your-apis-from-owasps-top-10-security-threats

➢ https://learn.snyk.io/lesson/unsafe-consumption-api/?ecosystem=python

➢ https://owasp.org/API-Security/editions/2023/en/0x11-t10/