AZƏRBAYCAN RESPUBLİKASI ELM VƏ TƏHSİL NAZİRLİYİ

AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

Cavid Əfəndiyev

Mikroservis arxitekturalı sistemlərdə təhlükəsizliyin təmin edilməsi alqoritmlərinin işlənilməsi

mövzusunda

MAGİSTRLİK DİSSERTASİYASI

İxtisas: 060631 - "Kompüter mühəndisliyi"
İxtisaslaşma: "Kompüter təhlükəsizliyi"

Elmi rəhbər: **r.ü.f.d. Rəhilə Sadıqova**

BAKI – 2023

Mündəricat

Giriş

Mövzunun aktuallığı. Mikroservis arxitekturası təşkilatları çeviklik, miqyaslanma qabiliyyəti, nasazlıqların izolyasiyası və texnologiya müxtəlifliyi ilə təmin edir. Bununla belə, mikroservislərin inkişafı və yerləşdirilməsinin hər bir aspektində təhlükəsizliyi prioritetləşdirmək vacibdir. Təhlükəsiz kommunikasiya, giriş idarəetməsi, məlumatların mühafizəsi tədbirləri və effektiv monitorinq və audit həyata keçirməklə təşkilatlar sistemlərini və həssas məlumatlarını qoruyarkən mikroservislərin gücündən istifadə edə bilərlər. Proqram təminatı inkişaf etdikcə, mikroservis arxitekturasına təhlükəsizlik baxımından birinci yanaşmanın tətbiqi istifadəçilərin inamını və etibarını qorumaq və qiymətli aktivləri potensial təhlükələrdən qorumaq üçün mühüm əhəmiyyət kəsb edir.

Dissertasiya işinin məqsədi. Dissertasiya işinin məqsədi mikroservis arxitekturasında yarana biləcək təhlükəsizlik problemlərini aydınlaşdırmaq, mümkün həlləri analiz etmək, kiberhücumlara daha dözümlü, daha təhlükəsiz proqram təminatlarının yaradılmasıdır.

Tədqiqatın predmeti. Tədqiqatın predmeti mikroservis arxitekturasında təhlükəsizlik problemlərinin araşdırılması, müasir həllərinin analizi və sistemin təhlükəsizliyinin artırılması ilə bağlı təklif və tövsiyələrin araşdırılmasıdır.

Tədqiqatın obyekti. Tədqiqat obyekti kimi mikroservis arxitekturasında yazılmış proqram təminatları götürülmüşdür.

Tədqiqatın elmi və nəzəri əsasları. Tədqiqatın elmi-nəzəri əsaslarını mütəxəssilərin, informasiya texnologiyası şirkətlərinin analizləri, təcrübələri təşkil edir.

Tədqiqatın informasiya təminatı. Dissertasiya işində xarici mənbələrdən və müasir texnologiyaların texniki sənədlərindən istifadə edilib.

Tədqiqatın elmi yeniliyi. Tədqiqatın elmi yeniliyi mövcud mikroservis arxitekturası problemlərinə müasir həllərin göstərilib və analiz edilməsidir.

Tədqiqatın praktiki əhəmiyyəti. Dissertasiya işində verilən tövsiyyələrin sistemi dizayn edərkən nəzərə alınması, sistemin daha dayanıqlı, daha təhlükəsiz olmasına şərait yaradacaq.

Dissertasiya işinin strukturu. Dissertasiya işi giriş, 2 fəsil və nəticədən ibarət olaraq 75 səhifədən ibarətdir. Dissertasiya işinin girişində mövzunun aktuallığı əaslandırılır, onun predmeti və obyekti verilir, elmi-nəzəri əsasları izah edilir və elmi yenilikləri verilir.

Dissertasiya işinin 1-ci fəslində proqram təminatı mühəndisliyinin bu günündən, mikroservis arxitekturasının niyə əhəmiyyətli olmasından bəhs edilir.

Dissertasiya işinin 2-ci fəslində isə Mikroservis arxitekturasında sirlərin idarə edilməsindəki problemlərdən və həllərdən bəhs edilir. “Hashicorp Vault” proqram təminatından istifadə edərək sirlərin idarə edilməsinə aid nümunə verilir.

Dissertasiya işinin “Nəticə” bölməsində aparılmış tədqiqat əsasında əldə edilmiş nəticələr və verilmiş təkliflər öz əksini tapmışdır.

I FƏSİL. MİKROSERVİS ARXİTEKTURASI - MÖVCUD PROQRAM TƏMİNATI MÜHƏNDİSLİYİNDƏ ƏHƏMİYYƏTİ

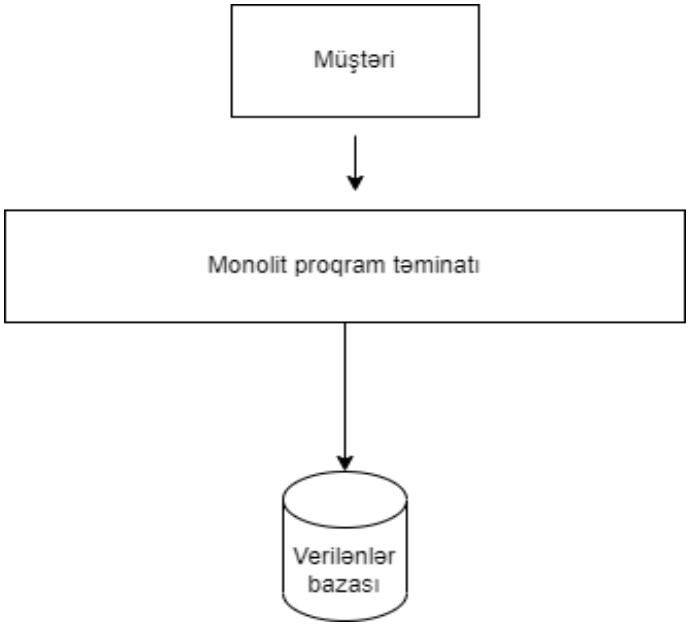
1.1. Proqram təminatı arxitekturalarının tarixçəsi

"Proqram təminatı arxitekturası" isfadəsini sənayedə tam ərtaflı izah edən bir tərif yoxdur. Buna baxmayaraq bəziləri proqram təminatı arxitekturasını bir sistemin planı kimi, bəziləri isə sistemin böyüməsinin yol xəritəsi kimi tərif edir.
Üstəlik, sürətlə dəyişən proqram təminatının inkişafı ekosisteminə görə proqram dizaynı davamlı olaraq hədəfi dəyişir.

Ötən əsrin ortalarından bu yana komputerlərin inkişafı və sürətlə yayılması ilə proqram təminatlarının sayı, onların yaradılması sürəti də artdı. Ötən əsrdə proqram təminatı yaratmaq üçün istifadə edilən bəzi metodlar artıq bugün müasir komputer arxitekturası, biznes dünyası trendləri ilə artıq ayaqlaşa bilmir. Dünyanın tələbləri dəyişdikcə, bu tələbləri qarşılamaq üçün yaradılan proqram təminatlarının strukturu, dizaynı, inkişaf etdirmə metodları da dəyişir. Günümüzədək istifadə olunan proqram təminatı arxitekturalarına da baxsaq bunu aydın bir şəkildə görə bilərik.

- **Monolitik arxitektura**

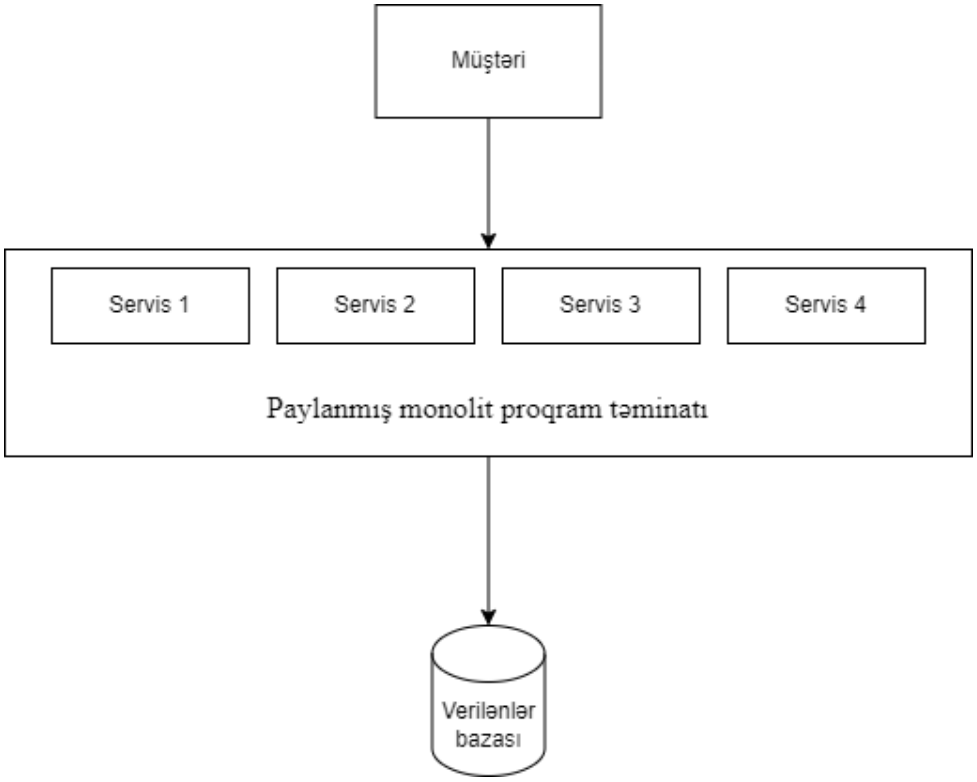
Proqram təminatı yaradılmasının ilk günlərində monolitik arxitektura bir norma idi. Bu arxitektura elə bir yanaşmadır ki, burada hər bir proqram təminatı komponentləri bir vahidin daxilinə cəmlənirdi. Bu komponentlərə istifadəçi interfeysi, biznes məntiq, məlumatlarla işləmə qatını nümunə çəkmək olar. Bu komponentlər hər biri bir icra ediləbilən faylın içərisində olur (Şəkil 1.1). Monolitik proqram təminatında sistemin komponentlərindən birinə edilən xırda bir dəyişiklik üçün bütün proqram təminatıdan yenidən artifakt yaradılmalıdır ki, bəzən bu proses vaxt alıcı və çətin ola bilir. Həmçinin sistemin miqyaslanması və proqram təminatına dəstəyin göstərilməsi də çətindir. Bu çətinlikləri aradan qaldırmaq üçün sonradan servis yönümlü və mikroservis arxitekturaları yarandı.



Şəkil 1.1. Monolit arxitektura

- **Paylanmış monolitik arxitektura**

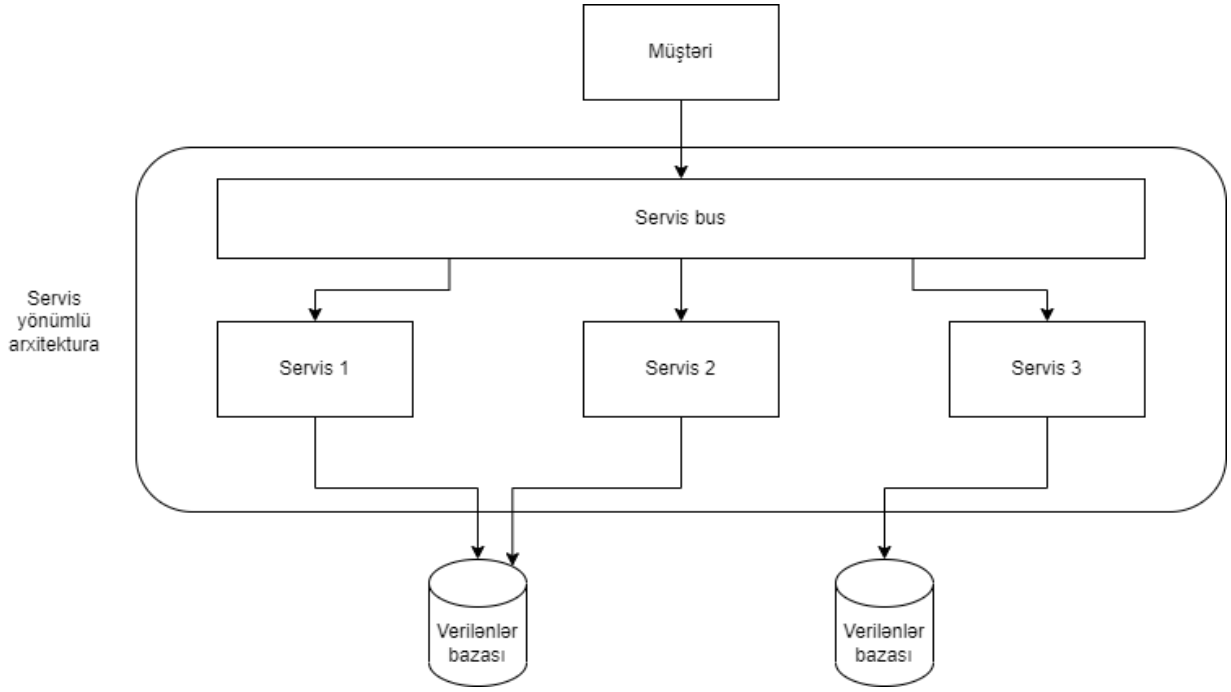
Paylanmış monolit arxitekturasında proqram təminatı bir kod daxildən müxtəlif servislərə bölünür. Bu cür arxitektura çox zaman proqramçılara mənbə kodlarını oxumaqda çətinliyi azaltmaq, proqram təminatına dəyişiklik və əlavələrin edilməsinə ayrılan zamanı azaltmağı hədəfləyir. Monolit arxitektura kimi proqram təminatı məlumatları bir verilənlər bazasında saxlanılır. Bütün servislər bir artifakt daxilində olur (Şəkil 1.2). Deploy zamanı isə qurulmuş mühitə uyğun olaraq ya bütün servislər eyni anda deploy olunmalı ya da hər bir servis ayrılıqda deploy edilə bilər. Amma ki, servislərin hamısının eyni verilənlər bazasını işlətməsi servislərin ayrı-ayrı deploy edilməsi prosesini çətinləşidir.



Şəkil 1.2. Paylanmış monolit arxitektura

- **Servis yönümlü arxitektura**

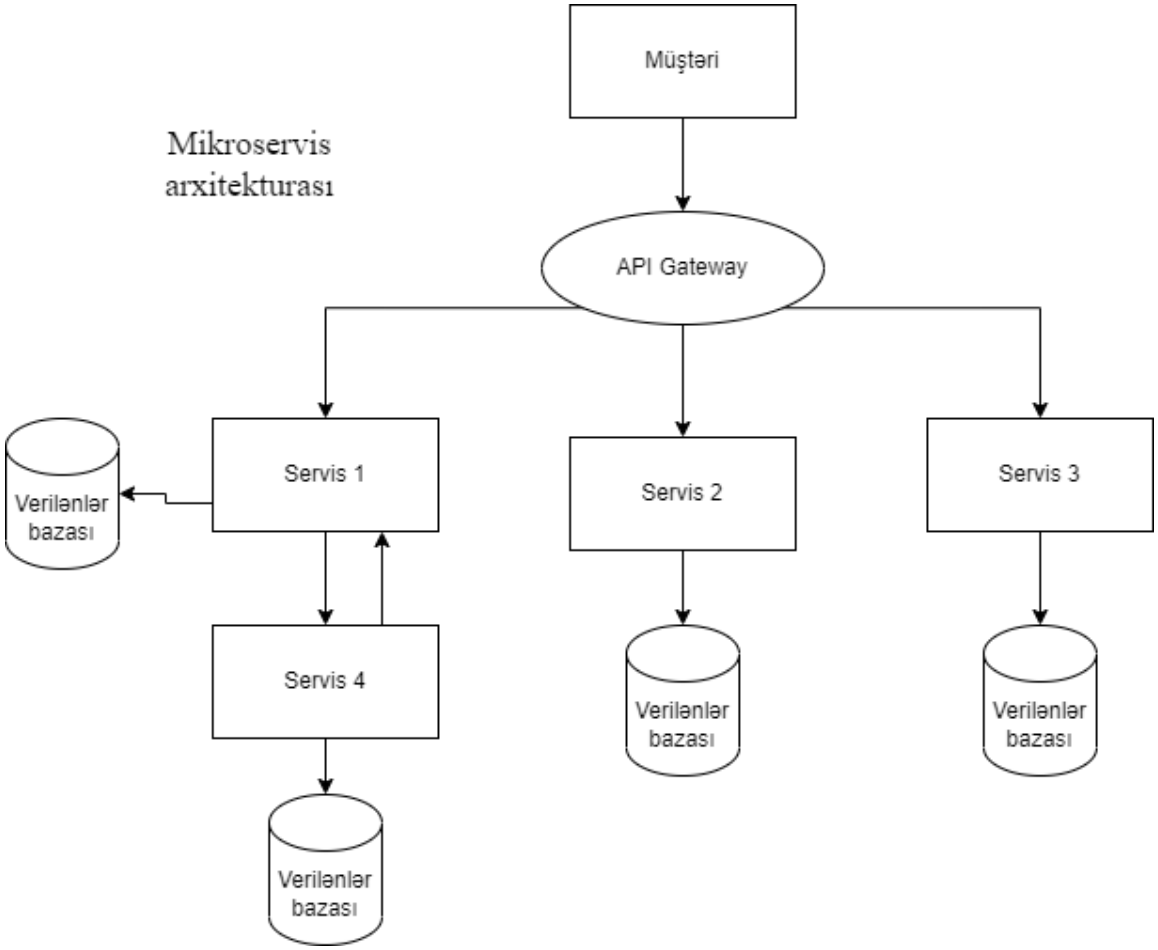
Servis yönümlü arxitekturada böyük sistem və ya proqram təminatı servislər toplusuna bölünür. Bu servislər sistemin komponentləri tərəfindən ortaq istifadə olunur. Hər bir servisin özünü xüsusi bir funksionallığı olur və bu servislər öz aralarında müəyyən protokollar vasitəsilə ilə xəbərləşirlər. Servis yönümlü arxitektura yenidən istifadə oluna bilmə, miqyaslanma və elasktiklik kimi üstünlük verir. Bəzi servislər müxtəlif servislər tərəfindən istifadə oluna bilinir. Bu isə kod bazasında əlavə kodlardan azad olmağa kömək edir. Yarandığı gündən bu günə qədər servis yönümlü arxitektura sürətlə məşhurlaşmağa başladı. Buna baxmayaraq, servis yönümlü arxitekturanın bəzi mənfi cəhətləri vardır. Məsələn, monolitik ilə müqayisədə bu arxitekturanın yaradılması daha çətindir. Həmçinin servislər arasında əlaqədə problemlər yaradır. Çünki bir çox hallarda xəbərləşmə zamanı servisin şəbəkə çıxışı olur ki, təhlükəsizlik məsələlərini önə çıxarır (Şəkil 1.3).



Şəkil 1.3. Servis yönümlü arxitektura

- **Mikroservis arxitektura**

Servis yönümlü arxitektura ilə oxşarlıqları çoxdur. Bəzi ədəbiyyatlarda mikroservis arxitekturası servis yönümlü arxitekturanın xüsusi bir yanaşması sayılır. Bugün çox böyük, qarışıq proqram təminatları, sistemlər üçün mikroservis arxitekturası səmərəli bir həll yolu kimi görünür. Sistem kiçik servislərə bölünür və idarə edilir. Hər bir servisin xüsusi funksionallığı olur və digər servislərlə müəyyən protokollar üzərində xəbərləşirlər (Şəkil 1.4). Sistemin kiçik-kiçik hissələrə bölünməsi, sistemin üzərində daha çox komandanın çalışmasına şərait yaradır. Həmçinin, mikroservis arxitekturasında servislər bir-birindən asılı olmadan inkişaf etdirilməlidir ki, bu da servislərin müstəqil şəkildə - digər servislərdən asılı olmadan deploy olunmasına imkan verir. Bu arxitekturanın da digərləri kimi üstün cəhətləri olduğu kimi, mənfi cəhətləri də vardır. Bu arxitekturada proqram təminatını yaratmaq daha çətindir və daha xərc tələb edir. Servislərin monitorinqi və koordinasiyası xüsusi diqqət tələb edir.



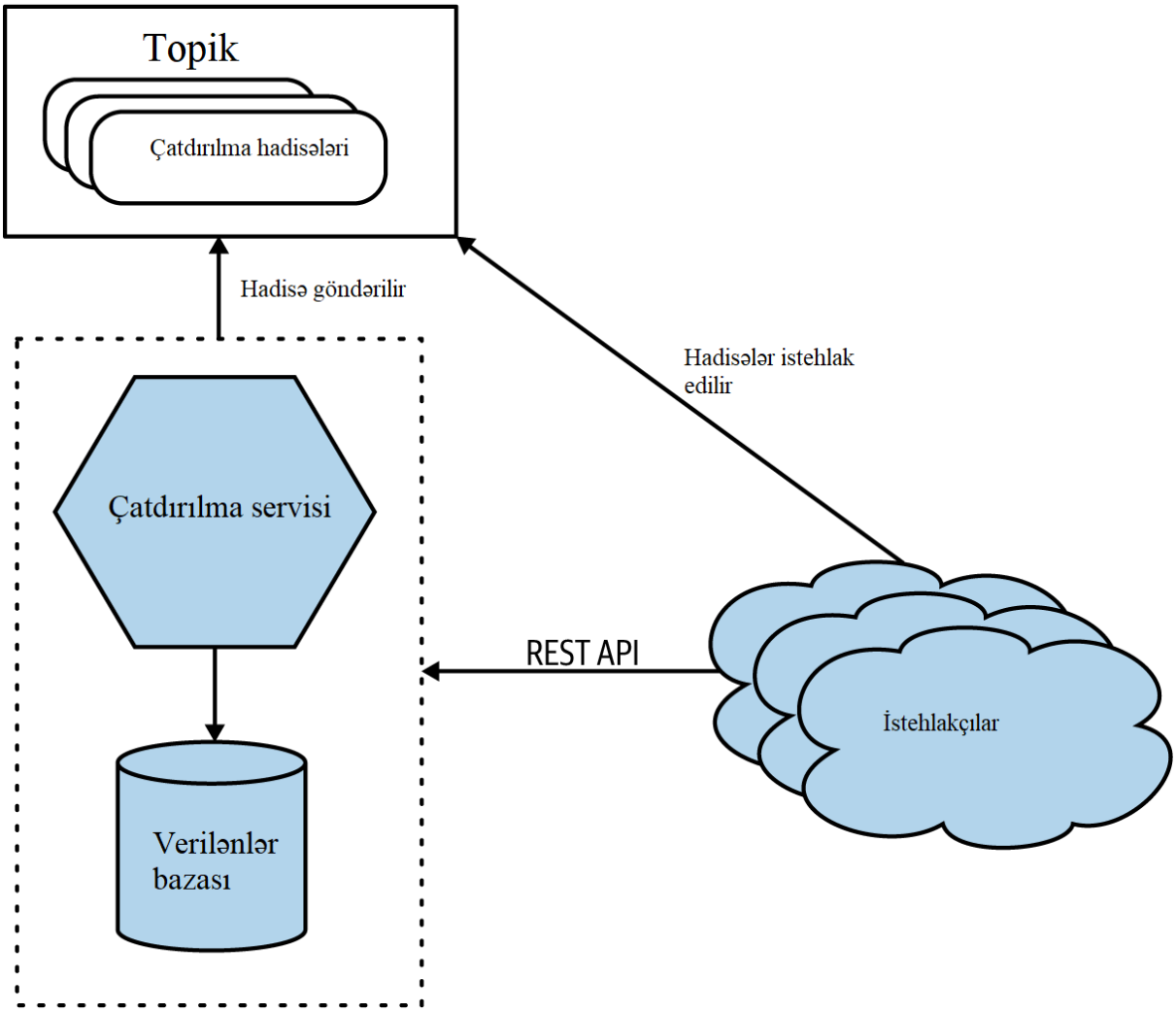
Şəkil 1.4. Mikroservis arxitekturası

1.2. Mikroservis arxitekturası

Mikroservislər biznes sahəsi ətrafında modelləşdirilmiş müstəqil olaraq buraxıla bilən servislərdir. Servis funksionallığı əhatə edir və onu şəbəkələr vasitəsilə digər servislər üçün əlçatan edir - siz bu tikinti bloklarından daha mürəkkəb sistem qurursunuz. Bir mikroservis inventar, başqa bir sifariş idarəetməsi və başqa bir göndərməni təmsil edə bilər, lakin onlar birlikdə bütöv bir ticarət sistemini təşkil edə bilər. Mikroservislər, qarşılaşa biləcəyiniz problemləri həll etmək üçün sizə bir çox variant verməyə yönəlmiş bir arxitektura seçimidir.

Mikroservislər servis sərhədlərinin necə çəkiləcəyi müəyyən edilmiş servis yönümlü arxitekturanın bir növüdür və müstəqil deploy edilə bilinmə onun əsas xüsusiyyətlərindən biridir. Onlar texnoloji aqnostikdirlər ki, bu da onların təklif etdiyi üstünlüklərdən biridir.

Kənardan bir mikroservis qara qutu kimi qəbul edilir. O, bir və ya daha çox şəbəkə son nöqtəsində (məsələn, növbə və ya REST API, Şəkil 1.5-də göstərildiyi kimi) biznes funksiyalarını ən uyğun olan protokollar üzərində saxlayır. İstehlakçılar, istər digər mikroservislər, istərsə də digər proqram növləri olsun, bu şəbəkəyə qoşulmuş son nöqtələr vasitəsilə bu funksiyaya daxil olurlar.



Şəkil 1.5. Mikroservis arxitekturasında kommunikasiya metodları

Mikroservislər məlumat gizlətmə konsepsiyasını əhatə edir. Məlumat gizlətmək komponent daxilində mümkün qədər çox məlumat gizlətmək və xarici interfeyslər vasitəsilə mümkün qədər az ifşa etmək deməkdir. Bu, asanlıqla dəyişə bilən və dəyişdirilməsi daha çətin olanları aydın şəkildə ayırmağa imkan verir. Mikroservisin ifşa etdiyi şəbəkə interfeysləri geriyə uyğun olmayan şəkildə dəyişmədiyi müddətcə,

kənar tərəflərdən gizlədilən tətbiq sərbəst şəkildə dəyişdirilə bilər. Mikroservis sərhədindəki dəyişikliklər (Şəkil 1.1-də göstərildiyi kimi) servisinin istehlakçılarına təsir etməməlidir. Bu, mikroservislərin təcrid olunmuş şəkildə işləməsinə və tələb olunduqda müstəqil şəkildə deploy edilməsinə imkan vermək üçün vacibdir. Servis daxilində hər hansısa reallaşdırma dəyişdikdə dəyişməyən aydın, sabit servis sərhədlərinə malik olmaq, servislər arasında daha boş birləşmə və daha güclü ahəngə malik sistemlərlə nəticələnir.

1.3. Mikroservis arxitekturasının əsas konseptləri

Mikroservisləri araşdırarkən bir neçə əsas ideya başa düşülməlidir. Bəzi aspektlərin tez-tez diqqətdən kənarda qaldığını nəzərə alsaq, mikroservislərin işləməsinə səbəb olanın nə olduğunu başa düşməyinizə kömək etmək üçün bu anlayışları daha da araşdırmaq çox vacibdir.

- Müstəqli deploy edilə bilmə

Müstəqil deploy edilə bilmə, mikroservisə dəyişiklik edə, onu deploy edə və bu dəyişikliyi istifadəçilərimizə hər hansı digər mikroservisləri deploy etmədən nail ola biləcəyimiz ideyasıdır. Daha önəmlisi, təkcə bunu edə bilməyimiz deyil. Bu həmçinin sistemdə deployların necə idarə edilməsi ilə bağlıdır. Bu, icra baxımından mürəkkəb olsa da, sadə bir fikirdir. Bu, mikroservislərin deploy edilməsi nizamıdır.

Müstəqil deploy edə bilmə qabiliyyətini təmin etmək üçün mikroservislərimizin boş birləşdirildiyinə əmin olmalıyıq. Yəni, başqa heç nəyi dəyişmədən bir servisi dəyişdirə bilməliyik. Bu o deməkdir ki, bizə xidmətlər arasında aydın, dəqiq müəyyən edilmiş və sabit müqavilələr lazımdır. Bəzi tətbiq seçimləri bunu çətinləşdirir - məsələn, verilənlər bazalarının paylaşılması xüsusilə problemlidir.

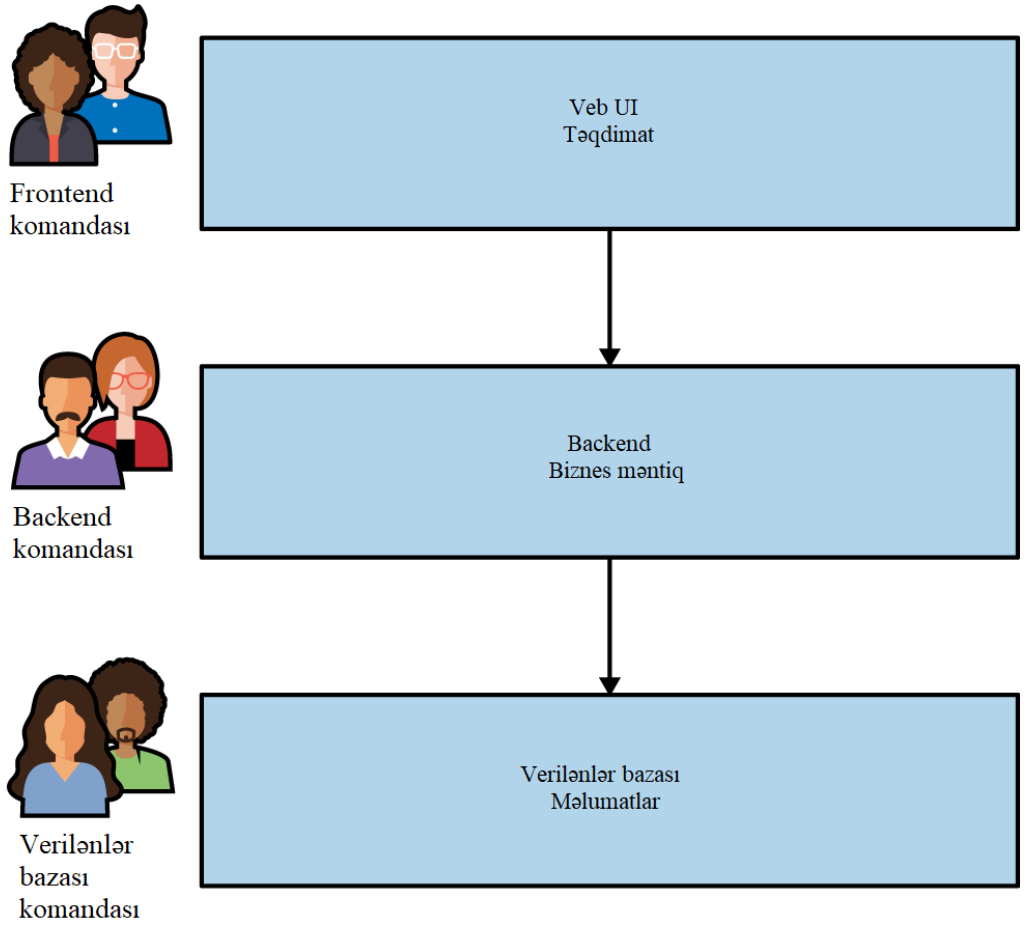
Müstəqil deploy edə bilmə qabiliyyəti özlüyündə inanılmaz dərəcədə dəyərlidir. Ancaq müstəqil deploy etməyə nail olmaq üçün düzgün əldə etməli olduğunuz bir çox başqa şeylər var ki, onların da öz faydaları var. Beləliklə, siz müstəqil deploy etməni diqqəti məcburetmə funksiyası kimi görə bilərsiniz - nəticə olaraq buna diqqət yetirməklə, bir sıra köməkçi üstünlüklərə nail olacaqsınız.

- Biznes domen ətrafında modelləmə

Domenə əsaslanan dizayn kimi texnikalar sizə proqram təminatının işlədiyi real dünya domenini daha yaxşı təmsil etmək üçün kodunuzu strukturlaşdırmağa imkan verə bilər. Mikroservis arxitekturaları ilə biz eyni ideyadan xidmət sərhədlərimizi müəyyən etmək üçün istifadə edirik. Biznes domenləri ətrafında servisləri modelləşdirməklə biz istifadəçilərimizə yeni funksionallıq təqdim etmək üçün yeni funksionallığın tətbiqini və mikroservisləri müxtəlif üsullarla birləşdirməyi asanlaşdıra bilərik.

Birdən çox mikroservisə dəyişiklik tələb edən funksionallığın yayılması baha başa gəlir. Bunun üçün hər bir servis üzrə (və potensial olaraq ayrı-ayrı komandalar arasında) işi koordinasiya etməli və bu xidmətlərin yeni versiyalarının deploy qaydasını diqqətlə idarə edilməlidir. Bu, bir servis daxilində (və ya monolit daxilində) eyni dəyişikliyi etməkdən daha çox iş tələb edir. Beləliklə, servislər arası dəyişiklikləri mümkün qədər az etmək üçün yollar tapmaq istəyi yaranır.

Ənənəvi üsullarla işləyən qurum və ya şirkətlərdə əsasən 3 səviyyəli arxitektura istifadə edilir. Burada arxitekturadakı hər bir səviyyə əlaqəli texniki funksionallığa əsaslanan hər bir servis sərhədi ilə fərqli servis sərhədini təmsil edir (Şəkil 1.6). Bu misalda sadəcə təqdimat qatında dəyişiklik etməliyəmsə, bu, kifayət qədər səmərəli olardı. Bununla belə, təcrübə göstərir ki, funksionallıqdakı dəyişikliklər adətən bu tip arxitekturalarda bir neçə təbəqəni əhatə edir - təqdimat, tətbiq və məlumat səviyyələrində dəyişikliklər tələb olunur. Arxitektura Şəkil 1.6-dakı sadə nümunədən daha çox laylı olarsa, bu problem daha da kəskinləşir. Servislərimizi biznes funksionallığı ilə başa çatdırmaqla, biz arxitekturamızın biznes funksionallığında mümkün qədər səmərəli dəyişikliklər etmək üçün təşkil olunmasını təmin edirik. Şübhəsiz ki, mikroservislərlə biz texniki funksionallığın yüksək uyğunluğundansa, biznes funksionallığının ahəngdar birləşməsinə üstünlük vermək qərarına gəldik.



Şəkil 1.6. Laylı komanda bölgüsü

- Öz məlumatlarına sahiblik

Mikroservislər hər biri öz verilənlər bazasına sahib olmalıdır. Mikroservis arxitekturasının bu xüsusiyyəti öz mövcud monolit sistemlərini miqrasiya etmək və ya sıfırdan mikroservislər yazmaq insanlar üçün çox zaman çətinlik kimi görünür. Mikroservis başqa bir mikroservis tərəfindən saxlanılan məlumatlara daxil olmaq istəyirsə, gedib həmin ikinci mikroservisdən məlumat tələb etməlidir. Bu, mikroservislərə nəyin paylaşıldığını və nəyin gizlədildiyinə qərar vermək imkanı verir ki, bu da bizə sərbəst dəyişə bilən funksionallığı nadir hallarda dəyişmək istədiyimiz funksionallıqdan aydın şəkildə ayırmağa imkan verir.

- Ölçü

Mikroservislərin arxitekturasında servislər kiçik olmaq üçün nəzərdə tutulmuşdur və xüsusi tapşırıq və ya funksiyanı yerinə yetirməyə yönəldilmişdir.

Servisin ölçüsü dəyişə bilər, lakin ümumiyyətlə qəbul edilir ki, mikroservis hələ də nəzərdə tutulan funksiyanı yerinə yetirə bildiyi halda, mümkün qədər kiçik olmalıdır. Servislərin kiçik olmasının səbəbi sistemdə daha çox çeviklik və dəyişkənliyə imkan verməkdir. Kiçik servislərin idarə edilməsi və saxlanması daha asandır və onlar daha böyük servislərə nisbətən daha tez inkişaf etdirilə, sınaqdan keçirilə və deploy edilə bilər. Bu, təşkilatlara dəyişən biznes tələblərinə sürətlə cavab verməyə və müştərilərə daha tez-tez yeni funksiyalar və funksionallıq təqdim etməyə imkan verir.

Praktikada mikroservislərin ölçüsü sistemin konkret məzmunundan və tələblərindən asılı olaraq dəyişə bilər. Bəzi mikroservislər çox kiçik ola bilər və yalnız bir funksiyanı yerinə yetirir, digərləri isə daha böyük və mürəkkəb ola bilər. Bununla belə, ümumiyyətlə xidmətləri mümkün qədər kiçik saxlamaq tövsiyə olunur, ideal olaraq bir neçə min kod sətirindən çox olmamalıdır.

Kiçik olmaqla yanaşı, mikroservislər də müstəqil və sərbəst şəkildə birləşdirilməlidir. Bu o deməkdir ki, hər bir xidmət bir funksiyaya və ya qabiliyyətə görə məsuliyyət daşımalı və sistemdəki digər servislər və ya komponentlərlə sıx bağlı olmamalıdır. Bu, sistemdə daha çox çeviklik və möhkəmliyə imkan verir, çünki hər bir servis sistemin qalan hissəsinə təsir etmədən müstəqil olaraq yenilənə və ya dəyişdirilə bilər.

- Çeviklik

Çeviklik mikroservis arxitekturasının əsas prinsiplərindən biridir. Mikroservislər tam tətbiqi təmin etmək üçün birlikdə işləyən kiçik, müstəqil xidmətlər toplusu kimi proqram sistemlərinin qurulması üsuludur. Bu yanaşmanın əsas üstünlüklərindən biri çevikliyi bir neçə yolla təmin etməsidir.

Mikroservislər texnologiya seçimlərində çevikliyə imkan verir. Hər bir mikroservis müəyyən bir funksiyanı yerinə yetirmək üçün nəzərdə tutulduğundan, tərtibatçılar həmin xüsusi servis üçün ən uyğun olan proqramlaşdırma dilini və texnologiyanı seçmək azadlığına malikdirlər. Bu o deməkdir ki, tərtibatçılar sistemdəki digər xidmətlər üçün edilən seçimlərlə məhdudlaşdırılmadan hər bir xidmət üçün ən uyğun texnologiyadan istifadə edə bilərlər.

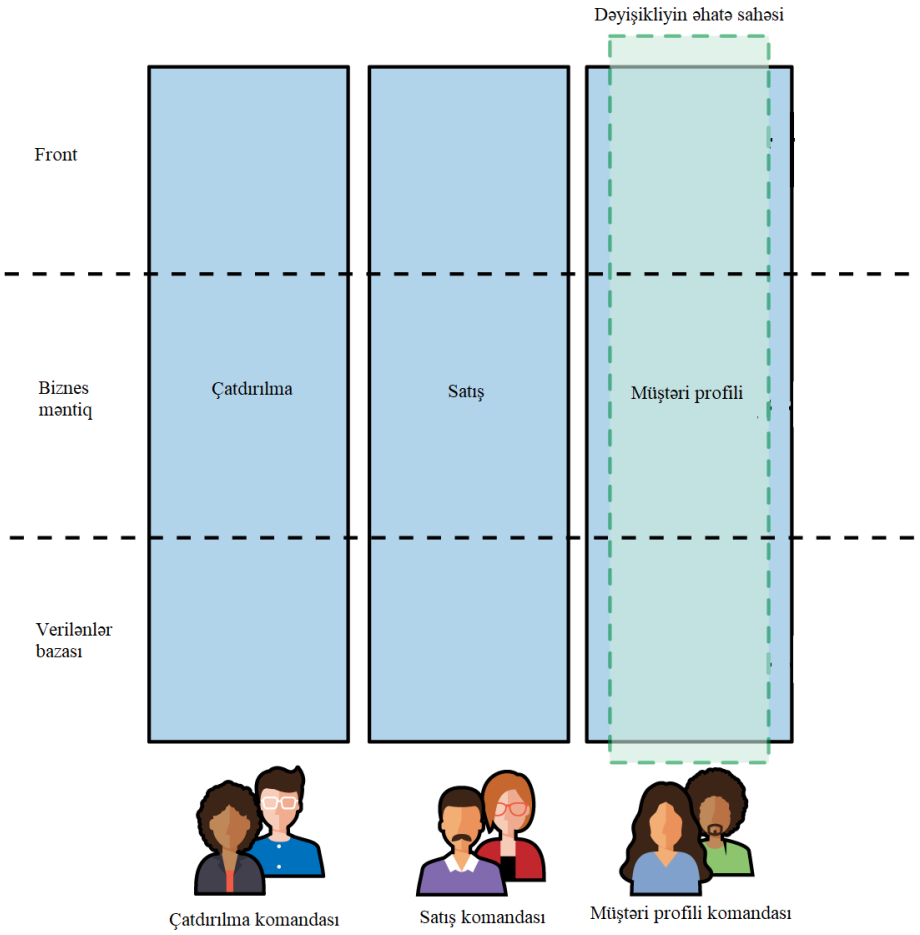
Mikroservislər deploy zamanı da çevikliyə imkan verir. Hər bir mikroservis ayrı bir qurum olduğundan, o, sistemdəki digər servislərdən asılı olmayaraq deploy edilə və miqyaslana bilər. Bu o deməkdir ki, tərtibatçılar sistemin qalan hissəsinə təsir etmədən ayrı-ayrı servislərə yeniləmələri deploy edə və onların xüsusi ehtiyaclarına əsasən fərdi servisləri genişləndirə bilərlər.

Mikroservislər inkişafda çevikliyə imkan verir. Hər bir mikroservis ayrı bir qurum olduğu üçün tərtibatçılar sistemdəki digər servislərdən asılı olmayaraq onlar üzərində işləyə bilərlər. Bu o deməkdir ki, tərtibatçı qrupları digər servislərdən asılılıqlar tərəfindən bloklanmadan eyni vaxtda müxtəlif servislər üzərində işləyə bilər. Bu, daha sürətli inkişaf dövrlərinə imkan verir, çünki bir servisə dəyişikliklər etmək və digər xidmətlərə təsir etmədən tətbiq etmək olar.

Mikroservislər memarlıqda çevikliyə imkan verir. Mikroservislər kiçik və müstəqil olmaq üçün nəzərdə tutulduğundan, müxtəlif arxitekturalar yaratmaq üçün müxtəlif yollarla birləşdirilə bilər. Bu o deməkdir ki, tərtibatçılar monolit arxitektura ilə məhdudlaşdırılmadan tətbiqlərinin tələblərinə ən uyğun olan arxitekturanı seçə bilərlər.

- Təşkilat strukturu

Mikroservislər bir-birindən asılı olmayan kiçik-kiçik servislərdən ibarət olması, bizə ənənəvi 3 səviyyəli təşkilat strukturundan uzaqlaşmağa kömək edir. Təşkilat daxilində komandaları artıq texnoloji alətlərə görə deyil, biznes domenlərə görə bölmək mümkün olur (Şəkil 1.7). Yəni, artıq bizim verilənlər bazası komandası, front komandamız yox, ödəniş komandamız, çatdırılma komandamız və invertar komandamız olacaq. Hər komandada ehtiyaca uyğun olaraq müəyyən sayda verilənlər bazası administratoru, front və back proqramçı, tester ola bilər. Bu cür komanda quruluşu biznesin sürətlə dəyişən tələblərini çevik şəkildə uyğunlaşmağa yardım edir. Şəkil 1.3-də nümunə göstərilmişdir. Müştəri ilə bağlı sistemdə hər hansısa dəyişiklik olacaqsa, bütün dəyişikliklər yalnız müştəri profili servisinə ediləcəkdir və bu dəyişikliyi etmək müştəri profili komandasının məsuliyyəti olacaqdır. Digər servis və komandalar bu dəyişikliklə bağlı narahat edilməyəcəklər.



Şəkil 1.7. Şaquli komanda bölgüsü

II FƏSİL. MİKROSERVİS ARXİTEKTURASINDA SİRLƏRİN İDARƏEDİLMƏSİNDƏKİ PROBLEMLƏR VƏ HƏLLƏR

2.1. Mikroservis arxitekturasındakı təhlükəsizlik problemləri

Mikroservisləri daha az paylanmış arxitekturalarla müqayisə etdikdə maraqlı təzadla qarşılaşdığımız məlum olur. Bir tərəfdən, indi əvvəllər tək bir maşında qalacaq olan şəbəkələr üzərindən daha çox məlumat axınına sahibik və arxitekturamızı idarə edən daha mürəkkəb infrastrukturumuz var - səth sahəsimiz hücum daha böyükdür. Digər tərəfdən, mikroservislər bizə dərindən müdafiə etmək və girişin əhatəsini məhdudlaşdırmaq üçün daha çox imkan verir, Sistemimizin proyeksiyasını potensial olaraq artırır, eyni zamanda hücum baş verərsə təsirini azaldır. Bu görünən paradoks - mikroservislərin sistemlərimizi həm daha az təhlükəsiz, həm də daha təhlükəsiz edə biləcəyi - həqiqətən, sadəcə incə balanslaşdırma aktıdır.

Mikroservis arxitekturasının təhlükəsizliyində düzgün balansı tapmaq üçün biz bəzi məsələləri toxunmalıyıq:

- Əsas prinsiplər

Çox vaxt mikroservis təhlükəsizliyi mövzusu gündəmə gələndə insanlar JWT tokenlərinin istifadəsi və ya qarşılıqlı TLS ehtiyacı kimi ağlabatan təkmil texnoloji məsələlər haqqında danışmaq istəyirlər. Bununla belə, təhlükəsizliklə bağlı problem ondan ibarətdir ki, siz yalnız ən az təhlükəsiz cəhətiniz qədər təhlükəsizsiniz. Bənzətmədən istifadə etmək üçün, əgər evinizi qorumaq istəyirsinizsə, ön qapıya kameralar və işıqlar qoyub, arxa qapını açıq qoymaqla siz evinizi qoruya bilməzsiniz.

Beləliklə, tətbiq təhlükəsizliyinin bəzi fundamental aspektləri var ki, biz onları qısa da olsa, bilməli olduğunuz məsələlərin çoxluğunu vurğulamaq üçün nəzərdən keçirməliyik. Biz bu əsas məsələlərin mikroservislər kontekstində necə daha çox və ya daha az mürəkkəb hala gətirildiyini nəzərdən keçirəcəyik, lakin onlar ümumilikdə proqram təminatının hazırlanmasına da şamil edilməlidir. Bütün bu "yaxşı şeylərə" keçmək istəyənlər üçün, arxa qapını açıq qoyarkən ön qapının təhlükəsizliyinə çox diqqət yetirmədiyinizə əmin olun.

Ən az imtiyaz prinsipi:

Fərdi şəxslərə, xarici və ya daxili sistemlərə, hətta öz mikroservislərimizə proqram girişi verərkən, biz hansı girişi verdiyimizə diqqət yetirməliyik. Ən az imtiyaz prinsipi belə bir fikri təsvir edir ki, giriş icazəsi verərkən tərəfə tələb olunan funksionallığı yerinə yetirmək üçün lazım olan minimum girişi verməliyik və yalnız onlara lazım olan müddət ərzində. Bunun əsas faydası odur ki, etimadnamələr təcavüzkar tərəfindən ələ keçirilərsə, həmin etimadnamələr zərərli tərəfə mümkün qədər məhdud giriş imkanı verəcək.

Əgər mikroservis verilənlər bazasına yalnız oxumaq üçün giriş imkanına malikdirsə, o zaman həmin verilənlər bazası etimadnaməsinə giriş əldə edən təcavüzkar yalnız oxumaq üçün giriş əldə edəcək və yalnız həmin verilənlər bazasına. Əgər verilənlər bazası etimadnaməsi təhlükə altına düşməmişdən əvvəl başa çatırsa, etimadnamə yararsız olur. Bu konsepsiya müəyyən tərəflər tərəfindən hansı mikroservislərlə əlaqə saxlanıla biləcəyini məhdudlaşdırmaq üçün genişləndirilə bilər.

Ən az imtiyazın əsas aspektləri bunlardır:

- Giriş məhdudiyyəti:

Ən az imtiyaz, bilmək lazımdır prinsipinə əsaslanan giriş hüquqlarını və icazələri məhdudlaşdırmağı əhatə edir. İstifadəçilərə yalnız öz iş funksiyalarını yerinə yetirmək üçün tələb olunan xüsusi resurslara, sistemlərə və ya məlumatlara giriş icazəsi verilməlidir. Təhlükəsizlik pozuntusunun və ya imtiyazlardan sui-istifadənin potensial təsirini minimuma endirmək üçün lazımsız imtiyazlar ləğv edilir.

- Minimalizm prinsipi:

Ən az imtiyaz minimalizm prinsipinə uyğundur, yəni bir tapşırığı effektiv şəkildə yerinə yetirmək üçün lazım olan ən az imtiyazların verilməsi deməkdir. Geniş və defolt giriş hüquqlarını təmin etmək əvəzinə, giriş icazələri hər bir hal üzrə təyin edilir və istifadəçilərin yalnız ciddi şəkildə tələb olunanlara çıxış əldə etməsini təmin edir.

- Xırda icazələr:

Ən az imtiyaz icazələr və giriş hüquqları üzərində dəqiq nəzarət tələb edir. İstifadəçi və ya qrup səviyyəsində geniş icazələrin verilməsi əvəzinə, giriş fərdi resurs

səviyyəsində verilir. Bu, təşkilatlara icazəsiz hərəkətlər riskini azaltmaqla daha dəqiq və idarə oluna bilən girişə nəzarət sistemi yaratmağa imkan verir.

- İmtiyazların ayrılması:

Ən az imtiyaz həm də vəzifə və məsuliyyətlərin ayrılmasını təşviq edir. İnzibati tapşırıqlar və ya həssas əməliyyatlar kimi kritik hərəkətlər ayrıca imtiyazlı hesablar və ya şəxslər tərəfindən yerinə yetirilməlidir. İmtiyazları ayırmaqla təşkilatlar zərərli fəaliyyət və ya təsadüfi sui-istifadə riskini minimuma endirirlər.

- Daimi baxış və yeniləmələr:

Ən az imtiyaz birdəfəlik tətbiq deyil, davamlı bir prosesdir. Giriş hüquqları və icazələri iş rolları, məsuliyyətlər və ya layihə tələblərindəki dəyişikliklər əsasında mütəmadi olaraq nəzərdən keçirilməli və yenilənməlidir. Daimi auditlər giriş imtiyazlarının ən az imtiyaz prinsipi ilə uzlaşdığını təmin etməyə kömək edir.

Ən az imtiyazın üstünlükləri:

- Minimallaşdırılmış hücum səthi:

Ən az imtiyaz tətbiq etməklə, təşkilatlar potensial hücumçular üçün imtiyazlı hesabların və giriş nöqtələrinin sayını məhdudlaşdırmaqla hücum səthini azaldır. İstifadəçi hesabı təhlükə altına düşsə belə, potensial zərər müəyyən resurslar dəsti ilə məhdudlaşır.

- Yan hərəkətin qarşısının alınması

Ən az imtiyaz şəbəkə daxilində yanal hərəkətin qarşısını alır. Təcavüzkar məhdud imtiyazları olan istifadəçi və ya sistemə giriş əldə edərsə, onların şəbəkəni keçmək və imtiyazları artırmaq qabiliyyəti əhəmiyyətli dərəcədə məhdudlaşdırılır və bu, qabaqcıl hücumların həyata keçirilməsini çətinləşdirir.

- Daxili təhlükələrin azaldılması:

Ən az imtiyaz daxili təhlükələrin riskini azaltmağa kömək edir. İşçilərə yalnız öz iş funksiyalarını yerinə yetirmək üçün lazım olan girişi verməklə təşkilatlar məlumatların qəsdən və ya qəsdən pozulması və ya imtiyazlardan sui-istifadə ehtimalını azalda bilər.

- Uyğunluq və yoxlanıla bilənlik:

Ən az imtiyaz tənzimləyici uyğunluq tələblərinə uyğun gəlir və auditi asanlaşdırır. Təşkilatlar granüler giriş nəzarətlərini tətbiq etməklə və giriş icazələrini müntəzəm olaraq nəzərdən keçirməklə sənaye standartlarına və tənzimləyici çərçivələrə uyğunluğunu nümayiş etdirə bilər.

- Təkmilləşdirilmiş təhlükəsizlik duruşu:

Ümumilikdə, ən az imtiyaz minimum giriş prinsipini tətbiq etməklə və potensial hücum vektorlarını məhdudlaşdırmaqla təşkilatın təhlükəsizlik vəziyyətini gücləndirir. Bu, icazəsiz giriş, məlumatların pozulması və imtiyazlardan daxili sui-istifadə ehtimalını azaldır.

Yekun olaraq, ən az imtiyaz prinsipinin qəbul edilməsi giriş hüquq və icazələrinin minimum zəruri səviyyəyə məhdudlaşdırılmasını təşviq edən fundamental təhlükəsizlik təcrübəsidir. Ən az imtiyaz tətbiq etməklə, təşkilatlar icazəsiz giriş riskini azalda, təhlükəsizlik insidentlərinin təsirini məhdudlaşdıra və ümumi təhlükəsizlik vəziyyətini yaxşılaşdıra bilər.

Avtomatlaşdırma:

Mikroservis arxitekturası ilə daha çox hərəkət edən hissələrimiz olduğundan, avtomatlaşdırma sistemimizin artan mürəkkəbliyini idarə etməkdə bizə kömək etmək üçün əsas rol oynayır. Eyni zamanda, çatdırılma sürətini artırmağa doğru hərəkətimiz var və burada avtomatlaşdırma vacibdir. Kompüterlər eyni şeyi təkrar-təkrar etməkdə insanlardan qat-qat yaxşıdır - onlar bunu bizdən daha sürətli və daha səmərəli edir (həm də daha az dəyişkənliklə). Onlar həmçinin insan səhvini azalda və ən az imtiyaz prinsipinin həyata keçirilməsini asanlaşdıra bilər – məsələn, biz konkret skriptlərə xüsusi imtiyazlar təyin edə bilərik.

Avtomatlaşdırma hadisədən sonra sistemi yenidən bərpa etməyə kömək edə bilər. Biz ondan təhlükəsizlik açarlarını ləğv etmək və fırlatmaq üçün istifadə edə bilərik, həmçinin potensial təhlükəsizlik problemlərini daha asan aşkar etmək üçün alətlərdən istifadə edə bilərik. Mikroservis arxitekturasının digər aspektlərində olduğu kimi, avtomatlaşdırma mədəniyyətini mənimsəmək təhlükəsizlik məsələsində sizə çox kömək edəcək.

Çatdırılma Prosesində Təhlükəsizlik qurmaq:

Proqram təminatının bir çox digər aspektləri kimi, təhlükəsizlik də çox vaxt sonradan düşünülür. Tarixən heç olmasa sistemin təhlükəsizlik aspektlərinin həlli kod yazıldıqdan sonra edilən bir şeydir və potensial olaraq daha sonra əhəmiyyətli yenidən işləməyə səbəb olur. Təhlükəsizliyə tez-tez proqram təminatının qapıdan çıxarılmasının qarşısını alan bir şey kimi baxılırdı. Son 20 il ərzində sınaq, istifadə imkanları və əməliyyatlarla bağlı oxşar problemlərin şahidi olduq. Proqram təminatının çatdırılmasının bu aspektləri tez-tez kodun əsas hissəsi tamamlandıqdan sonra gizli şəkildə çatdırılırdı. Təbii ki, reallıq ondan ibarətdir ki, istifadə oluna bilməyən, təhlükəsiz olmayan, istehsalda düzgün işləyə bilməyən və səhvlərlə dolu olan proqram təminatı heç bir formada deyil. məliyyat aspektləri (DevOps, kimsə?) və istifadə imkanları ilə etdiyimiz kimi, sınaqları əsas çatdırılma prosesinə çəkməkdə daha yaxşı olduq - təhlükəsizlik fərqli olmamalıdır. Biz təmin etməliyik ki, tərtibatçılar təhlükəsizliklə bağlı narahatlıqlar barədə daha ümumi məlumatlı olsunlar, mütəxəssislər tələb olunduqda çatdırılma qruplarına daxil olmaq üçün bir yol tapsınlar və proqram təminatımızda təhlükəsizliklə bağlı düşüncələr qurmağa imkan verən alətlər təkmilləşsin.

Saytlararası skript hücumlarını axtarmaq kimi sistemlərimizdə zəiflikləri yoxlaya bilən avtomatlaşdırılmış alətlər mövcuddur. Bu alətlərə “Zed Attack Proxy”, Synk, Brakeman kimi proqramları göstərə bilərik.

- Kibertəhlükəsizliyin beş funksiyası

Beynimizdə olan bu əsas prinsiplərlə indi yerinə yetirməli olduğumuz geniş miqyaslı təhlükəsizliklə bağlı fəaliyyətləri nəzərdən keçirək. Daha sonra bu fəaliyyətlərin mikroservis arxitekturası kontekstində necə dəyişdiyini anlamağa davam edəcəyik. Tətbiq təhlükəsizliyi kainatını təsvir etmək üçün üstünlük verdiyim model kibertəhlükəsizliklə bağlı müxtəlif fəaliyyətlər üçün faydalı beş hissədən ibarət modeli əks etdirən ABŞ Milli Standartlar və Texnologiya İnstitutundan (NIST) gəlir:

- **Müəyyən etmək**: Potensial təcavüzkarlarınızın kim olduğunu, hansı hədəfləri əldə etməyə çalışdıqlarını və ən həssas olduğunuz yerləri müəyyənləşdirin.
- **Qorumaq**: Əsas aktivlərinizi potensial hakerlərdən qoruyun.

- **Aşkar etmək**: Ən yaxşı səylərinizə baxmayaraq, hücumun baş verib-vermədiyini aşkar edin.
- **Cavab vermək**: Pis bir şeyin baş verdiyini bildiyiniz zaman cavab verin.
- **Bərpa etmək**: Hadisədən sonra sistemi bərpa edin.

Ağıllı bir təcavüzkar müdafiənizi aşarsa, nə edə biləcəyinizi düşünmədən, əslində hansı təhlükələrlə qarşılaşa biləcəyinizi nəzərə almadan tətbiqinizi qorumaq üçün bütün səylərinizi sərf etmək çox asandır.

Bu funksiyaların hər birini bir az daha dərindən araşdıraq və daha ənənəvi monolit arxitektura ilə müqayisədə mikroservis arxitekturasının bu ideyalara yanaşma tərzinizi necə dəyişə biləcəyinə baxaq.

Müəyyən etmək

Nəyi qorumalı olduğumuzu anlamazdan əvvəl, kimin bizim əşyalarımızın arxasınca düşə biləcəyini və tam olaraq nə axtardıqlarını öyrənməliyik. Özümüzü təcavüzkarın düşüncəsinə salmaq çox vaxt çətindir, lakin səylərimizi düzgün yerə cəmləməyimizi təmin etmək üçün məhz bunu etməliyik. Tətbiq təhlükəsizliyinin bu aspektinə müraciət edərkən ilk diqqət etməli olduğunuz şey təhlükənin modelləşdirilməsidir.

İnsanlar olaraq biz riski başa düşməkdə olduqca pisik. Gözdən uzaq ola biləcək daha böyük problemlərə məhəl qoymadan tez-tez yanlış şeylərə diqqət yetiririk. Bu, əlbəttə ki, təhlükəsizlik sahəsinə də aiddir. Hansı təhlükəsizlik risklərinə məruz qala biləcəyimiz barədə anlayışımız çox vaxt sistemə, bacarıqlarımıza və təcrübələrimizə məhdud baxışımızla rənglənir.

Qorumaq

Ən qiymətli və ən həssas aktivlərimizi müəyyən etdikdən sonra onların lazımi şəkildə qorunduğuna əmin olmalıyıq. Qeyd etdiyim kimi, mikroservis arxitekturaları, şübhəsiz ki, bizə daha geniş hücum səthi sahəsi verir və buna görə də qorunmağa ehtiyacı olan daha çox şeyimiz var, lakin onlar bizə dərindən müdafiə etmək üçün daha çox seçimlər verir.

Aşkar etmək

Mikroservis arxitekturası ilə insidentin aşkarlanması daha mürəkkəb ola bilər. Nəzarət etmək üçün daha çox şəbəkəmiz və nəzarət etmək üçün daha çox maşınımız var. İnformasiya mənbələri xeyli artır, bu da problemlərin aşkar edilməsini daha da çətinləşdirir. Loqların toplanması pis bir şeyin baş verə biləcəyini aşkar etməyə kömək edir. Bunlara əlavə olaraq, pis davranışı aşkar etmək üçün istifadə edə biləcəyiniz müdaxilə aşkarlama sistemləri kimi xüsusi vasitələr var. Sistemlərimizin artan mürəkkəbliyi ilə məşğul olmaq üçün proqram təminatı, xüsusən də Aqua kimi alətlərlə konteyner iş yükləri məkanında təkmilləşdirilir.

Cavab vermək

Ən pisi baş veribsə və siz bundan xəbər tutmusunuzsa, nə etməlisiniz? İnsidentə qarşı effektiv cavab yanaşmasının işlənib hazırlanması pozuntunun vurduğu zərərin məhdudlaşdırılması üçün çox vacibdir. Bu, adətən pozuntunun əhatə dairəsini və hansı məlumatların ifşa olunduğunu anlamaqla başlayır. Əgər ifşa edilən məlumatlar şəxsiyyəti müəyyənləşdirən məlumatları (PII) ehtiva edirsə, onda siz həm təhlükəsizlik, həm də məxfilik insidentlərinə cavab və bildiriş proseslərinə əməl etməlisiniz. Bu, təşkilatınızın müxtəlif hissələri ilə danışmalı olduğunuz anlamına gələ bilər və bəzi hallarda müəyyən növ pozuntular baş verdikdə qanuni olaraq təyin edilmiş məlumatların mühafizəsi üzrə məsul işçini məlumatlandırmaq məcburiyyətində ola bilərsiniz.

Bir çox təşkilat pozuntunun nəticələrini səhv idarə etməklə, onun brendinə və müştərilərlə münasibətlərinə dəyən zərərdən başqa, tez-tez artan maliyyə cəzalarına səbəb olur. Buna görə də, yalnız qanuni və ya uyğunluq səbəblərinə görə nə etməli olduğunuzu deyil, həm də proqram təminatınızın istifadəçilərinə baxmaq baxımından nə etməli olduğunuzu başa düşmək vacibdir. Məsələn, GDPR (General Data Protection Regulation) tələb edir ki, şəxsi məlumatların pozulması barədə 72 saat ərzində müvafiq orqanlara məlumat verilsin - bu, çox ağır görünməyən bir qrafikdir. Bu o demək deyil ki, siz insanların məlumatlarının pozulmasının baş verib-vermədiyini onlara daha əvvəl bildirməyə çalışa bilməyəcəksiniz.

Cavabın xarici ünsiyyət aspektləri ilə yanaşı, daxili işləri necə idarə etdiyiniz də vacibdir. Təqsir və qorxu mədəniyyətinə sahib olan təşkilatlar böyük bir hadisədən

sonra pis nəticələnə bilər. Dərslər öyrənilməyəcək və töhfə verən amillər üzə çıxmayacaq. Digər tərəfdən, açıqlığa və təhlükəsizliyə diqqət yetirən bir təşkilat, oxşar hadisələrin baş vermə ehtimalının daha az olmasını təmin edən dərsləri öyrənmək üçün ən yaxşı şəkildə yerləşdiriləcəkdir.

Bərpa etmək

Bərpa hücumdan sonra sistemi yenidən işə salmaq qabiliyyətimizə, həmçinin problemlərin yenidən baş vermə ehtimalını azaltmaq üçün öyrəndiklərimizi həyata keçirmək qabiliyyətimizə aiddir. Mikroservis arxitekturası ilə daha çox hərəkət edən hissələrə sahibik ki, problem geniş təsir göstərərsə, bərpa prosesini daha mürəkkəb edə bilər.

- Tətbiq təhlükəsizliyinin əsasları

Yaxşı, indi bizdə bəzi əsas prinsiplər və təhlükəsizlik fəaliyyətlərinin əhatə edə biləcəyi geniş dünya anlayışımız var, əgər siz daha təhlükəsiz sistem yaratmaq istəyirsinizsə, mikroservis arxitekturası kontekstində bir neçə təməl təhlükəsizlik mövzusuna baxaq – etimadnamələr , yamaqlar, ehtiyat nüsxələri çıxarın və yenidən qurun.

Etimadnamələr

Geniş şəkildə desək, etimadnamələr şəxsə (və ya kompüterə) hansısa formada məhdud resursa giriş imkanı verir. Bu verilənlər bazası, kompüter, istifadəçi hesabı və ya başqa bir şey ola bilər. Mikroservis arxitekturası ilə, ekvivalent monolit arxitektura ilə müqayisəsi baxımından, bizdə çox güman ki, eyni sayda insan iştirak edir, lakin qarışıqda müxtəlif mikroservisləri, (virtual) maşınları, verilənlər bazalarını və s. . Bu, girişi necə məhdudlaşdırmaq (və ya məhdudlaşdırmamaq) ilə bağlı müəyyən dərəcədə çaşqınlığa səbəb ola bilər və bir çox hallarda geniş imtiyazlara malik olan az sayda etimadnamənin işlərin sadələşdirilməsi cəhdində istifadə edildiyi “tənbəl” yanaşmaya gətirib çıxara bilər. Bu, öz növbəsində, etimadnamələr pozulursa, daha çox problemlərə səbəb ola bilər.

Etimadnamə mövzusunu iki əsas sahəyə bölmək olar. Birincisi, sistemimizin istifadəçilərinin (və operatorlarının) etimadnaməsini əldə edirik. Bunlar çox vaxt

sistemimizin ən zəif nöqtələridir və bir azdan görəcəyimiz kimi, adətən zərərli tərəflər tərəfindən hücum vektoru kimi istifadə olunur. İkincisi, biz sirləri nəzərdən keçirə bilərik - mikroservislərimizi idarə etmək üçün vacib olan məlumat hissələri. Hər iki etimadnamə dəsti arasında biz rotasiya, ləğvetmə və əhatə dairəsinin məhdudlaşdırılması məsələlərini nəzərdən keçirməliyik.

İstifadəçi etimadnamələri

E-poçt və parol birləşmələri kimi istifadəçi etimadnamələri çoxumuzun proqram təminatımızla işləməsi üçün vacib olaraq qalır, lakin sistemlərimizin zərərli tərəflər tərəfindən əldə edilməsi ilə bağlı potensial zəif yerdir. Verizon-un 2020 Məlumat Pozuntu Araşdırmaları Hesabatı [1], haker hücumu nəticəsində yaranan halların 80%-də etimadnamə oğurluğunun bəzi formalarından istifadə edildiyini göstərdi. Bura fişinq hücumları kimi mexanizmlər vasitəsilə etimadnamələrin oğurlandığı və ya parolların zorakılıqla tətbiq edildiyi hallar daxildir.

Şifrələr kimi şeyləri necə düzgün idarə etmək barədə bəzi əla məsləhətlər var - sadə və sadə olmasına baxmayaraq, hələ də kifayət qədər geniş şəkildə qəbul edilmir. Troy Hunt həm NIST, həm də Böyük Britaniyanın Milli Kibertəhlükəsizlik Mərkəzinin ən son məsləhətlərinə əla baxışa malikdir [2]. Bu tövsiyəyə parol menecerlərindən və uzun parollardan istifadə etmək, mürəkkəb parol qaydalarından istifadə etməmək və bir qədər təəccüblü olaraq, müntəzəm parol dəyişikliklərindən qaçmaq üçün tövsiyələr daxildir.

API ilə idarə olunan sistemlərin hazırkı dövründə, etimadnamələrimiz ictimai bulud provayderinizin hesabları kimi üçüncü tərəf sistemləri üçün API açarları kimi şeyləri idarə etməyə də şamil edilir. Zərərli tərəf kök AWS hesabınıza giriş əldə edərsə, məsələn, o hesabda işləyən hər şeyi məhv etməyə qərar verə bilər. Ekstremal bir misalda belə bir hücum Code Spaces adlı şirkətin işdən çıxması ilə nəticələndi – onların bütün resursları bir hesabda, ehtiyat nüsxələrdə və hamısında işləyirdi. “Rock Solid, Secure və Affordable Svn Hosting, Git Hosting və Project Management” təklif edən Code Spaces-in istehzası məndən uzaq deyil.

Kimsə bulud provayderiniz üçün API açarlarınızı ələ keçirsə və qurduğunuz hər şeyi məhv etməyə qərar verməsə belə, fərqinə varmayacağınız ümidi ilə bəzi bitcoin

mədənlərini idarə etmək üçün bəzi bahalı virtual maşınları fırlatmağa qərar verə bilər. . Bu, hesab bağlanmazdan əvvəl kiminsə 10 min dollardan çox pul xərclədiyini aşkar edən müştərilərimdən birinin başına gəldi. Məlum olub ki, təcavüzkarlar da avtomatlaşdırmağı bilirlər – orada sadəcə etimadnamələri skan edəcək və kriptovalyuta mədənçiliyi üçün maşınları işə salmaq üçün onlardan istifadə etməyə çalışacaq botlar var.

Versiyaların yenilənməsi

2017-ci il Equifax məlumatlarının pozulması versiyaların yenilənməsinin vacibliyinə gözəl bir nümunədir. Apache Struts-da məlum zəiflik Equifax tərəfindən saxlanılan məlumatlara icazəsiz giriş əldə etmək üçün istifadə edilmişdir. Equifax kredit bürosu olduğu üçün bu məlumat xüsusilə həssas idi. Sonda məlum olub ki, 160 milyondan çox insanın məlumatı pozularaq oğurlanıb. Equifax 700 milyon dollar ödəməli oldu.

Pozulmadan aylar əvvəl Apache Struts-da zəiflik müəyyən edilmişdi və problemi həll edən texniki qulluqçular tərəfindən yeni buraxılış hazırlanmışdı. Təəssüf ki, hücumdan bir neçə ay əvvəl mövcud olmasına baxmayaraq, Equifax proqram təminatının yeni versiyasına yeniləmədi. Equifax bu proqramı vaxtında yeniləsəydi, ehtimal ki, hücum mümkün olmayacaqdı.

Biz getdikcə daha mürəkkəb sistemləri yerləşdirdikcə, versiyların yenilənməsini prioritet saxlamaq məsələsi daha da mürəkkəbləşir. Bu kifayət qədər əsas konsepsiyanı necə idarə etdiyimizi daha da təkmilləşdirməliyik.

Yedəkləmələr

Məlumatlar həmişəkindən daha qiymətlidir və buna baxmayaraq bəzən düşünürəm ki, texnologiyadakı təkmilləşdirmələr bizə ehtiyat nüsxələrini prioritetləşdirməyə səbəb olubmu? Disklər əvvəlkindən daha etibarlıdır. Verilənlər bazalarında məlumat itkisinin qarşısını almaq üçün daxili replikasiya olma ehtimalı daha yüksəkdir. Bu cür sistemlərlə biz özümüzü ehtiyat nüsxələrə ehtiyacımız olmadığına inandıra bilərik. Bəs fəlakətli bir səhv baş verərsə və bütün Cassandra klasteriniz yox olarsa? Və ya kodlaşdırma səhvi tətbiqinizin həqiqətən qiymətli məlumatları silməsi deməkdirsə? Yedəkləmələr həmişə olduğu kimi vacibdir.

Mikroservislərimizin deploylarınını avtomatlaşdırılmaqla, biz tam maşın ehtiyat nüsxələrini götürməyə ehtiyac duymuruq, çünki mənbə kodundan infrastrukturumuzu yenidən qura bilərik. Beləliklə, biz bütün maşınların vəziyyətini kopyalamağa çalışmırıq; biz bunun əvəzinə ehtiyat nüsxələrimizi ən qiymətli vəziyyətə yönəldirik. Bu o deməkdir ki, ehtiyat nüsxələrə diqqətimiz verilənlər bazalarımızdakı məlumatlar və ya bəlkə də tətbiq qeydlərimiz kimi şeylərlə məhdudlaşır. Düzgün fayl sistemi texnologiyası ilə, xidməti nəzərəçarpacaq dərəcədə kəsmədən verilənlər bazası məlumatlarının blok səviyyəli klonlarını dərhal götürmək mümkündür.

Yedəkləri götürən zaman əmin olmaq lazımdır ki, həmin yedəklər həqiqətən də işləyir. Yedəklərin işləkliyi daima mütəmadi olaraq yoxlanmalıdır. Problemli bir şey baş verdikdən sonra sistemin bərpası zamanı yedəklərin əslində düz götürülmədiyi məlum olması daha da böyük problemlər yaradacaq.

Sıfır etibar

Sıfır etibar mühitində işləyərkən, artıq təhlükəyə məruz qalmış bir mühitdə işlədiyinizi güman etməlisiniz - danışdığınız kompüterlər təhlükə altına düşə bilər, daxil olan əlaqələr düşmən tərəflərdən ola bilər, məlumatlarınınız və yazılarınınız pis insanlar tərəfindən oxuna bilər. Hər şeydən tam şübhə etməlisiniz. Bu sıfır etibar, güvən adlanır.

Sıfır etibar, prinsipcə, bir düşüncə tərzidir. Bu, bir məhsul və ya alətdən istifadə edərək sehrli şəkildə həyata keçirə biləcəyiniz bir şey deyil; bu bir fikirdir və bu fikir ondan ibarətdir ki, əgər siz pis aktyorların artıq mövcud ola biləcəyi düşmən mühitdə fəaliyyət göstərdiyiniz fərziyyəsi altında fəaliyyət göstərirsinizsə, o zaman hələ də təhlükəsiz fəaliyyət göstərə biləcəyinizə əmin olmaq üçün tədbirlər görməlisiniz.

Sıfır etibar kibertəhlükəsizliyə “heç kimə güvənmə, hər şeyi yoxla” prinsipini vurğulayan bir yanaşmadır. O, ənənəvi perimetr əsaslı təhlükəsizlik modelinə meydan oxuyur və hesab edir ki, heç bir istifadəçi, cihaz və ya şəbəkə komponenti mahiyyət etibarilə etibar edilə bilməz. Bunun əvəzinə, sıfır etibar davamlı yoxlamaya və ciddi giriş nəzarətinə diqqət yetirən təhlükəsizlik çərçivəsini təşviq edir.

Sıfır etibar konsepsiyası 2010-cu ildə Forrester Research analitiki Con Kindervaq tərəfindən təqdim edilib, lakin son illərdə İT mühitlərinin mürəkkəbliyi,

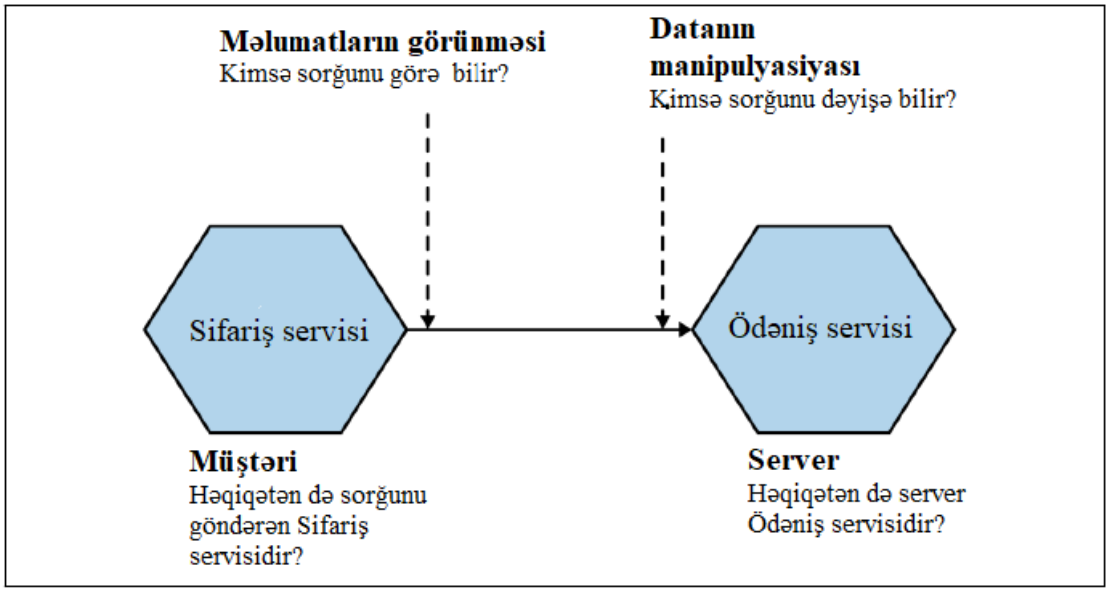
bulud hesablamalarının yüksəlişi və qoşulmuş cihazların çoxalması səbəbindən o, əhəmiyyətli dərəcədə cəlbedicilik qazanıb.

- Məlumatların Təhlükəsizliyinin təmin edilməsi

Monolit proqram təminatımızı mikroservislərə ayırdıqca, verilənlərimiz əvvəlkindən daha çox sistemlərimiz ətrafında hərəkət edir. O, sadəcə şəbəkələr üzərindən axmır; diskdə də oturur. Ehtiyatlı olmasaq, tətbiqimizin təhlükəsizliyini təmin etməyə gəldikdə, daha çox yerdə daha çox dəyərli məlumatların yayılması qorxulu yuxu ola bilər. Gəlin məlumatlarımızı şəbəkələr üzərində hərəkət edərkən və durğun vəziyyətdə ikən necə qoruduğumuza daha ətraflı baxaq.

Transitdə data

Sahib olduğunuz qorunmaların xarakteri əsasən seçdiyiniz rabitə protokollarının xarakterindən asılı olacaq. Məsələn, HTTP-dən istifadə edirsinizsə, HTTP-də TLS-dən istifadə etmək təbii olardı, lakin mesaj brokeri vasitəsilə əlaqə kimi alternativ protokollardan istifadə edirsinizsə, ötürülən məlumatların qorunması üçün həmin texnologiyanın dəstəyinə baxmalı ola bilərsiniz. Bu məkanda geniş çeşidli texnologiyanın təfərrüatlarına baxmaq əvəzinə, məncə, tranzit məlumatların təhlükəsizliyinə gəldikdə daha çox ümumi maraq doğuran dörd əsas sahəni nəzərdən keçirmək və bu narahatlıqların necə ola biləcəyinə baxmaq vacibdir.



Şəkil 2.1. Məlumatın tranzit zamanı qorunması

Şəkil 2.1-də tranzit zamanı məlumatların dörd əsas məsələni görə bilərik. Hər biri haqqında ətraflı məlumat verək:

- Server

Yoxlamaq üçün ən sadə şeylərdən biri, danışdığınız serverin tam olaraq iddia etdiyi şəxs olmasıdır. Bu vacibdir, çünki zərərli tərəf nəzəri olaraq son nöqtəni təqlid edə bilər və onu göndərdiyiniz istənilən faydalı məlumatları dəyişə bilər. Server identifikasiyasının təsdiqlənməsi ictimai internetdə uzun müddətdir narahatlıq doğurur və HTTPS-in daha geniş istifadəsinə təkan verir və biz daxili internetin idarə edilməsinə gəldikdə ictimai internetin təhlükəsizliyini təmin etmək üçün görülən işlərdən müəyyən dərəcədə faydalana bilirik.

- Müştəri

Bu kontekstdə müştəri şəxsiyyətinə istinad etdikdə, zəng edən mikroservisə istinad edirik, ona görə də yuxarı mikroservisin kimliyini təsdiqləməyə və autentifikasiya etməyə çalışırıq.

Biz müştərinin şəxsiyyətini bir neçə yolla yoxlaya bilərik. Müştəridən kim olduqlarını bildirən sorğuda bizə bəzi məlumat göndərməsini xahiş edə bilərik. Məsələn, sorğunu imzalamaq üçün bir növ paylaşılan sirrdən və ya müştəri tərəfi sertifikatından istifadə etmək olar.

Bu situasiyada server və müştəri arasında qarşılıqlı autentikasiya məlumatların daha təhlükəsiz ötürülməsinə şərait yaradır. Qarşılıqlı autentikasiya üçün qarşılıqlı TLS-dən istifadə edilir. Qarşılıqlı TLS-də həm server, həm də müştəri tərəfi sertifikatlardan istifadə edir.

- Məlumatların görünməsi

Biz bir mikroservisdən digərinə məlumat göndərdiyimiz zaman kimsə məlumatı görə bilərmi? Servislər arası məlumat axınında bəzi məlumatlar ictimai məlumatlardır, onlar qorunsa və ya qorunmasa da hər kəs tərəfindən bilinir, amma bəzi məlumatlar gizli məlumatlardır. Məsələn, müştərinin kart məlumatları. Bu məlumatların heç bir üçüncü şəxs tərəfindən görünməsi düzgün deyildir. Sadə köhnə HTTPS və ya qarşılıqlı TLS-dən istifadə etdiyiniz zaman məlumatlar ara tərəflər üçün görünməyəcək - bunun

səbəbi TLS-in göndərilən məlumatları şifrələməsidir. Məlumatın açıq şəkildə göndərilməsini istəsəniz, bu problemli ola bilər - məsələn, Squid və ya Varnish kimi əks proksilər HTTP cavablarını keşləyə bilir, lakin HTTPS ilə bu mümkün deyil.

- Datanın manipulyasiyası

Göndərilən məlumatların manipulyasiya edilməsinin pis ola biləcəyi bir sıra vəziyyətləri təsəvvür edə bilərik - məsələn, göndərilən pulun miqdarını dəyişdirmək. Beləliklə, Şəkil 8-də potensial təcavüzkarın Sifarişdən Ödənişə göndərilən sorğunu dəyişdirə bilməyəcəyinə əmin olmalıyıq.

Tipik olaraq, məlumatları görünməz edən qorunma növləri də məlumatların manipulyasiya edilə bilməyəcəyini təmin edəcək (məsələn, HTTPS bunu edir). Bununla belə, biz məlumatları açıq şəkildə göndərmək qərarına gələ bilərik, lakin yenə də onların manipulyasiya edilə bilməyəcəyinə əmin olmaq istəyirik. HTTP üçün belə yanaşmalardan biri göndərilən məlumatları imzalamaq üçün hash-əsaslı mesaj identifikasiyası kodundan (HMAC) istifadə etməkdir. HMAC ilə hash yaradılır və verilənlərlə birlikdə göndərilir və qəbuledici məlumatların dəyişdirilmədiyini təsdiqləmək üçün hashı verilənlərlə yoxlaya bilər.

Data durğun vəziyyətdə ikən

Yalan məlumat, xüsusilə həssasdırsa, məsuliyyətdir. Ümid edirik ki, təcavüzkarların şəbəkəmizi poza bilməməsi, həmçinin əsas məlumatlara giriş əldə etmək üçün tətbiqlərimizi və ya əməliyyat sistemlərimizi poza bilməmələri üçün əlimizdən gələni etdik. Bununla belə, onların baş verəcəyi təqdirdə hazır olmalıyıq - dərindən müdafiə əsasdır.

Eşitdiyimiz yüksək profilli təhlükəsizlik pozuntularının bir çoxu, təcavüzkar tərəfindən əldə edilən sabit vəziyyətdə və həmin məlumatların təcavüzkar tərəfindən oxuna bilməsi ilə bağlıdır. Bu, ya verilənlərin şifrələnməmiş formada saxlanması və ya məlumatların qorunması üçün istifadə edilən mexanizmin qüsuru olması səbəbindən baş verir.

Durğun vəziyyətdə məlumatların qorunması mexanizmləri çox və müxtəlifdir, lakin nəzərə alınmalı olan bəzi ümumi şeylər var.

Yaxşı bilinənlə davam etmək

Bəzi hallarda, məlumatların şifrələnməsi işini mövcud proqram təminatına yükləyə bilərsiniz, məsələn, verilənlər bazanızın şifrələmə üçün daxili dəstəyindən istifadə etməklə. Bununla belə, öz sisteminizdə məlumatları şifrələmək və deşifrə etmək ehtiyacı tapırsınızsa, tanınmış və sınaqdan keçirilmiş tətbiqlərdən istifadə etdiyinizə əmin olun. Qarışdırmağın ən asan yolu məlumat şifrələməsi öz şifrələmə alqoritmlərinizi həyata keçirməyə çalışmaq və ya hətta başqasının alqoritmlərini həyata keçirməyə çalışmaqdır. İstifadə etdiyiniz proqramlaşdırma dilindən asılı olmayaraq, yaxşı hesab edilən şifrələmə alqoritmlərinin nəzərdən keçirilmiş, müntəzəm yamaqlanmış tətbiqlərinə giriş əldə edəcəksiniz. Bunlardan istifadə edin! Seçdiyiniz texnologiya üçün poçt siyahılarına/məsləhət siyahılarına abunə olun ki, boşluqlar aşkar edildikdə onlardan xəbərdar olun, beləliklə onları yamaqlı və yeni saxlaya bilərsiniz.

Şifrələrin təhlükəsizliyini təmin etmək üçün siz mütləq duzlu şifrə hashing adlı texnikadan istifadə etməlisiniz. Bu, şifrələrin heç vaxt düz mətndə saxlanmamasını və təcavüzkar bir haşlanmış şifrəni kobud şəkildə zorlasa belə, digər parolları avtomatik oxuya bilməyəcəyini təmin edir. Səhv şəkildə həyata keçirilən şifrələmə heç birinin olmamasından daha pis ola bilər, çünki səhv şifrələmə sizdə hər şeyin təhlükəsiz olması hissi yarada və hər şeyi təhlükə altında qoya bilər.

Hər şeyin şifrələndiyini düşünmək hər şeyi bir qədər sadələşdirə bilər. Nəyin qorunmalı və ya qorunmaması barədə heç bir fərziyyə yoxdur. Bununla belə, problemin müəyyənləşdirilməsinə kömək etmək üçün hansı məlumatların logfayllara yerləşdirilə biləcəyi barədə hələ də düşünməli olacaqsınız və hər şeyi şifrələmək üçün hesablama yükü olduqca çətin ola bilər və nəticədə daha güclü avadanlıq tələb edə bilər. Refaktorinq sxemlərinin bir hissəsi kimi verilənlər bazası köçürmələrini tətbiq edərkən bu daha da çətinləşir. Edilən dəyişikliklərdən asılı olaraq, məlumatların şifrəsini açmaq, köçürmək və yenidən şifrələmək lazım ola bilər.

Sisteminizi daha incə xidmətlərə bölməklə, topdansatış şifrələnə bilən bütün məlumat anbarını müəyyən edə bilərsiniz, lakin bu, mümkün deyil. Bu şifrələməni məlum cədvəllər dəsti ilə məhdudlaşdırmaq məntiqli yanaşmadır.

Disk sahəsi ucuzlaşdıqca və verilənlər bazalarının imkanları yaxşılaşdıqca, böyük həcmdə məlumatların tutulması və saxlanılması asanlığı sürətlə yaxşılaşır. Bu

məlumatlar dəyərlidir – təkcə məlumatları getdikcə daha çox qiymətli aktiv kimi görən müəssisələrin özləri üçün deyil, eyni zamanda öz məxfiliyini qiymətləndirən istifadəçilər üçün də dəyərlidir. Fərdi şəxsə aid olan və ya şəxs haqqında məlumat əldə etmək üçün istifadə edilə bilən məlumatlar bizim ən diqqətli olduğumuz məlumatlar olmalıdır.

Ancaq həyatımızı bir az asanlaşdırsaq nə olacaq? Niyə şəxsiyyəti müəyyən edə bilən mümkün qədər çox məlumatı silməyə və bunu mümkün qədər tez etməyəsən? İstifadəçidən sorğu daxil edərkən bütün IP ünvanını əbədi saxlamalıyıq, yoxsa son bir neçə rəqəmi x ilə əvəz edə bilərik? Məhsul təklifləri ilə təmin etmək üçün kiminsə adını, yaşını, cinsini və doğum tarixini saxlamalıyıq, yoxsa onların yaş aralığı və poçt kodu kifayət qədər məlumatdır?

Məlumat toplamaqda qənaətcil olmağın üstünlükləri çoxşaxəlidir. Birincisi, onu saxlamasanız, heç kim onu oğurlaya bilməz. İkincisi, onu saxlamasanız, heç kim (məsələn, dövlət qurumu) da bunu tələb edə bilməz!

Şifrələmənin əksər formaları şifrələnmiş məlumat yaratmaq üçün uyğun bir alqoritmlə birlikdə bəzi açarlardan istifadə etməyi əhatə edir. Məlumatın oxunması üçün şifrəni açmaq üçün səlahiyyətli tərəflər açara giriş tələb edəcəklər - ya eyni açara, ya da fərqli açara (açıq açarla şifrələmə vəziyyətində). Bəs açarlarınız harada saxlanılır? İndi, əgər kiminsə bütün verilənlər bazamı oğurlayacağından narahat olduğum üçün məlumatımı şifrələyirəmsə və istifadə etdiyim açarı eyni verilənlər bazasında saxlayıramsa, deməli, çox şey əldə etməmişəm. Buna görə də, açarları başqa yerdə saxlamalıyıq.

Həllərdən biri məlumatı şifrələmək və deşifrə etmək üçün ayrıca təhlükəsizlik cihazından istifadə etməkdir. Digəri, xidmətinizin açara ehtiyacı olduqda daxil ola biləcəyi ayrıca açar anbarından istifadə etməkdir. Açarların həyat dövrünün idarə edilməsi (və onları dəyişdirmək imkanı) mühüm əməliyyat ola bilər və bu sistemlər bunu sizin üçün idarə edə bilər. HashiCorp-un Vault da çox lazımlı ola biləcəyi yer budur.

Bəzi verilənlər bazaları hətta SQL Serverin Şəffaf Məlumat Şifrələməsi kimi şifrələmə üçün daxili dəstəyi də ehtiva edir ki, bu da bunu şəffaf şəkildə idarə etməkdir.

Seçdiyiniz verilənlər bazası belə dəstəyi ehtiva etsə belə, açarların necə idarə olunduğunu araşdırın və qoruduğunuz təhlükənin həqiqətən azaldılıb-azaldılmadığını anlayın.

Yedəkləmələr yaxşıdır. Biz vacib məlumatlarımızın ehtiyat nüsxəsini çıxarmaq istəyirik. Və bu, açıq bir nöqtə kimi görünə bilər, lakin əgər məlumat kifayət qədər həssasdırsa, onun işləyən istehsal sistemimizdə şifrələnməsini istəyiriksə, o zaman biz də yəqin ki, eyni məlumatların ehtiyat nüsxələrinin də şifrələndiyinə əmin olmaq istəyəcəyik.

- Autentifikasiya və avtorizasiya

Sistemimizlə qarşılıqlı əlaqədə olan insanlara və əşyalara gəldikdə identifikasiya və avtorizasiya əsas anlayışlardır. Təhlükəsizlik kontekstində autentifikasiya tərəfin dedikləri şəxs olduğunu təsdiqlədiyimiz prosesdir. Biz adətən insan istifadəçini istifadəçi adı və parolunu daxil etməklə təsdiq edirik. Biz güman edirik ki, yalnız faktiki istifadəçi bu məlumatı əldə edə bilər və buna görə də bu məlumatı daxil edən şəxs onlar olmalıdır. Əlbəttə ki, başqa, daha mürəkkəb sistemlər də mövcuddur. Telefonlarımız indi bizə dediyimiz kimi olduğumuzu təsdiqləmək üçün barmaq izimizdən və ya üzümüzdən istifadə etməyə imkan verir. Ümumiyyətlə, kimin və ya nəyin autentifikasiyası haqqında mücərrəd şəkildə danışarkən, biz həmin tərəfi əsas olaraq adlandırırıq.

Avtorizasiya, bir rəhbərdən onlara icazə verdiyimiz hərəkətə keçmək mexanizmidir. Çox vaxt, direktor təsdiqləndikdə, bizə onlar haqqında məlumat veriləcək ki, bu da onlara nə etməli olduğumuzu qərara almağa kömək edəcək. Bizə, məsələn, onların hansı şöbədə və ya ofisdə işlədikləri barədə məlumat verilə bilər - sistemimizin direktorun nə edə və nəyi edə bilməyəcəyinə qərar vermək üçün istifadə edə biləcəyi bir məlumat.

İstifadə rahatlığı vacibdir - biz istifadəçilərimizin sistemimizə daxil olmasını asanlaşdırmaq istəyirik. İstəmirik ki, hər kəs üçün fərqli istifadəçi adı və parol istifadə edərək müxtəlif mikroservislərə daxil olmaq üçün ayrıca daxil olmaq məcburiyyətində qalsın. Beləliklə, biz mikroservislər mühitində tək girişi (SSO) necə həyata keçirə biləcəyimizə də baxmalıyıq.

Servisdən-servisə autentifikasiya

Daha əvvəl biz qarşılıqlı TLS-ni müzakirə etdik, bu, tranzit zamanı məlumatların qorunması ilə yanaşı, həm də autentifikasiya formasını həyata keçirməyə imkan verir. Müştəri qarşılıqlı TLS-dən istifadə edərək serverlə danışdıqda, server müştərinin autentifikasiyasını həyata keçirə bilir və müştəri serveri autentifikasiya edə bilir – bu xidmətdən xidmətə doğrulamanın bir formasıdır. Qarşılıqlı TLS-dən başqa digər autentifikasiya sxemləri də istifadə edilə bilər. Ümumi nümunə API açarlarının istifadəsidir, burada müştəri sorğunu elə bir şəkildə hash etmək üçün açardan istifadə etməlidir ki, server müştərinin etibarlı açardan istifadə etdiyini yoxlaya bilsin.

Biz insanların tanış istifadəçi adı və parol kombinasiyası ilə öz autentifikasiyasına öyrəşmişik. Bununla belə, getdikcə daha çox bu, çoxfaktorlu autentifikasiya yanaşmasının bir hissəsi kimi istifadə olunur, burada istifadəçinin özünü təsdiqləmək üçün birdən çox biliyə ehtiyacı ola bilər. Ən çox bu, birdən çox faktora ehtiyac duyulan çoxfaktorlu autentifikasiya (MFA - multifactor authentication) formasını alır. “MFA” ən azı bir əlavə amil təmin etməklə yanaşı, adətən normal istifadəçi adı və parol kombinasiyasının istifadəsini nəzərdə tutur.

Son illərdə müxtəlif növ autentifikasiya faktorları artıb - SMS və e-poçt vasitəsilə göndərilən kodlardan tutmuş xüsusi mobil proqramlara və ya NFC aparatlara gələn keçidlərə qədər. İstifadəçilərin barmaq izi və ya üz tanıma kimi funksiyaları dəstəkləyən aparatlara daha çox çıxışı olduğu üçün biometrik amillər də indi daha çox istifadə olunur. “MFA” özünü ümumi yanaşma kimi daha təhlükəsiz göstərsə də və bir çox dövlət xidmətləri bunu dəstəkləsə də, mən bunun dəyişəcəyini gözləsəm də, kütləvi bazarın autentifikasiyası sxemi kimi qəbul olunmayıb. Proqramınızın işləməsi üçün həyati əhəmiyyət kəsb edən və ya xüsusilə həssas məlumatlara (məsələn, mənbə koduna giriş) çıxışa imkan verən əsas xidmətlərin autentifikasiyasını idarə etmək üçün mən MFA-dan istifadəni zəruri hesab edərdim.

Sirlər

Geniş şəkildə desək, sirlər mikroservisin işləməsi üçün lazım olan kritik məlumat parçalarıdır və həm də zərərli tərəflərdən qorunmağı tələb edən kifayət qədər həssasdır. Mikroservisə lazım ola biləcək sirlərə aşağıdakı nümunələr daxildir:

- TLS üçün sertifikatlar
- SSH açarları
- Açıq/gizli API açar cütləri
- Verilənlər bazasına daxil olmaq üçün etimadnamələr

Bir sirrin həyat dövrünü nəzərdən keçirsək, müxtəlif təhlükəsizlik ehtiyaclarını tələb edə biləcək sirlərin idarə edilməsinin müxtəlif aspektlərini ayırmağa başlaya bilərik:

- Yaradılış
  İlk növbədə sirri necə yaradırıq?

- Paylanma
  Sirr yaradıldıqdan sonra onun doğru yerə (və yalnız doğru yerə) çatdığından necə əmin ola bilərik?

- Saxlama
  Sirr yalnız səlahiyyətli şəxslərin ona daxil olmasını təmin edəcək şəkildə saxlanılırmı?

- Monitorinq
  Bu sirrin necə istifadə olunduğunu bilirikmi?

- Yeniləmə
  Problem yaratmadan sirri dəyişə bilirikmi?

Hər biri müxtəlif sirlər dəstini istəyə biləcək bir sıra mikroservislərimiz varsa, bütün bunları idarə etmək üçün alətlərdən istifadə etməli olacağıq.

Kubernetes daxili sirr həllini təmin edir. O, funksionallıq baxımından bir qədər məhduddur, lakin əsas Kubernetes quraşdırmasının bir hissəsi kimi gəlir, ona görə də bir çox istifadə halları üçün kifayət qədər yaxşı ola bilər.

Bu məkanda daha mürəkkəb alət axtarırsınızsa, Hashicorp's Vault baxmağa dəyər. Mövcud kommersiya variantları olan açıq mənbə aləti, sirlərin yayılmasının əsas aspektlərindən tutmuş verilənlər bazası və bulud platformaları üçün vaxt məhdud etimadnamələrin yaradılmasına qədər hər şeyi idarə edən əsl İsveçrə Ordusunun sirləri idarə edən bıçağıdır. Vault-un əlavə üstünlüyü var ki, dəstəkləyən konsul şablonu aləti normal konfiqurasiya faylındakı sirləri dinamik şəkildə yeniləyə bilir. Bu o deməkdir ki, sisteminizin yerli fayl sistemindən sirləri oxumaq istəyən hissələrinin sirri idarəetmə alətini dəstəkləmək üçün dəyişdirilməsinə ehtiyac yoxdur. Vault-da sirr dəyişdirildikdə, konsul şablonu konfiqurasiya faylındakı bu qeydi yeniləyə bilər və mikroservislərinizə istifadə etdikləri sirləri dinamik şəkildə dəyişməyə imkan verir. Bu, etimadnamələri miqyasda idarə etmək üçün əladır.

Bəzi ictimai bulud provayderləri də bu məkanda həllər təklif edir; məsələn, AWS Secrets Manager və ya Azure-un Key Vault yadıma gəlir. Bəzi insanlar kritik məxfi məlumatların bu kimi ictimai bulud xidmətində saxlanması ideyasını bəyənmirlər. Yenə də bu, təhdid modelinizə düşür. Əgər bu ciddi narahatlıq doğurursa, seçdiyiniz ictimai bulud provayderinizdə Vault-u işə salmağa və bu sistemi özünüz idarə etməyə heç nə mane ola bilməz. İstirahət halında olan məlumatlar bulud provayderində saxlansa belə, müvafiq yaddaş dəstəyi ilə siz məlumatların elə şifrələndiyinə əmin ola bilərsiniz ki, kənar tərəf məlumatı tutsa belə, onunla heç nə edə bilməyəcək.

İdeal olaraq, etimadnamələrə giriş əldə etdikdə kiminsə edə biləcəyi zərəri məhdudlaşdırmaq üçün etimadnamələri tez-tez dəyişmək istəyirik. Zərərli tərəf AWS API açıq/gizli açar cütünüzə giriş əldə edərsə, lakin həmin etimadnamə həftədə bir dəfə dəyişdirilirsə, onların etimadnamələrdən istifadə etmək üçün cəmi bir həftə vaxtı var. Onlar, əlbəttə ki, bir həftə ərzində hələ də çox zərər verə bilərlər, amma siz fikirləşirsiniz. Təcavüzkarların bəzi növləri sistemlərə giriş əldə etməyi və sonra aşkarlanmamağı xoşlayır ki, bu da onlara zamanla daha qiymətli məlumat toplamaq və sisteminizin digər hissələrinə daxil olmaq yollarını tapmaq imkanı verir. Əgər onlar giriş əldə etmək üçün oğurlanmış etimadnamələrdən istifadə etmişlərsə, istifadə etdikləri etimadnamələr onlardan çox istifadə etmədən əvvəl başa çatarsa, siz onları izlərində dayandıra bilərsiniz.

Operator etimadnamələri üçün yenilənmənin əla nümunəsi AWS-dən istifadə üçün vaxtı məhdud API açarlarının yaradılması ola bilər. Bir çox təşkilatlar indi öz işçiləri üçün operativ olaraq API açarlarını yaradırlar, ictimai və şəxsi açar cütləri yalnız qısa müddət ərzində – adətən bir saatdan az müddət ərzində etibarlıdır. Bu, sizə lazım olan istənilən əməliyyatı yerinə yetirmək üçün lazım olan API açarlarını yaratmağa imkan verir, belə ki, zərərli tərəf sonradan bu açarlara giriş əldə etsə belə, onlardan istifadə edə bilməyəcək. Təsadüfən o klaviaturanı ictimai GitHub-da yoxlasanız belə, müddəti bitdikdən sonra heç kimə faydası olmayacaq.

Vaxt məhdudiyyəti olan etimadnamələrin istifadəsi sistemlər üçün də faydalı ola bilər. Hashicorp's Vault verilənlər bazası üçün vaxt məhdud etimadnaməsini yarada bilər. Mikroservis instansiyanızın verilənlər bazası bağlantısı təfərrüatlarını konfiqurasiya mağazasından və ya mətn faylından oxumaq əvəzinə, onlar mikroservisinizin konkret nümunəsi üçün tez yaradıla bilər.

Açarlar kimi etimadnamələrin tez-tez fırlanması prosesinə keçmək ağrılı ola bilər. Açarların dəyişdirilməsi nəticəsində sistemlərin işləməyi dayandırdığı şirkətlərlə danışdım. Bu, çox vaxt müəyyən bir etimadnamənin nədən istifadə olunduğuna dair qeyri-müəyyən ola bilməsi ilə əlaqədardır. Etibarnamənin əhatə dairəsi məhduddursa, yenilənmənin potensial təsiri əhəmiyyətli dərəcədə azalır. Lakin etimadnamənin geniş istifadəsi varsa, dəyişikliyin təsirini öyrənmək çətin ola bilər. Bu, sizi rotasiyadan yayındırmaq üçün deyil, sadəcə sizi potensial risklərdən xəbərdar etmək üçündür və mən bunun düzgün iş olduğuna əminəm. İrəliləmənin ən ağlabatan yolu, çox güman ki, bu prosesi avtomatlaşdırmağa kömək edəcək alətləri qəbul etmək və eyni zamanda hər bir etimadnamə dəstinin əhatə dairəsini məhdudlaşdırmaqdır.

2.2. Sirlərin saxlanılması problemləri

2022-ci ilin mart ayında GitGuardian açıqladı ki, ictimai GitHub repozitoriyalarında ifşa olunan sirlərin sayı 2020-ci ilə nisbətən 2021-ci ildə iki dəfə artaraq cəmi altı milyon sirrə çatdı [3]. Proqram təminatı əsas funksionallıq üçün xarici veb xidmətlərindən istifadə edir. Ödəniş sistemləri, məkan xidmətləri və sosial şəbəkə

platforması inteqrasiyası üçün API-lər, bir neçəsini desək, hamısı xarici veb xidmətlərinin nümunələridir. Sistem inteqrasiyasının bir hissəsi kimi proqram artefaktları arasında autentifikasiyanı həyata keçirmək üçün proqram tərtibatçıları sirlərə (verilənlər bazası etimadnamələri, API açarları və digər etimadnamələr) ehtiyac duyurlar. Proqram təminatının hazırlanması zamanı bu sirlərin komandada işləyən tərtibatçılar tərəfindən paylaşılması və yerləşdirmədən sonra proqramlara paylanması tələb oluna bilər.

2019-cu ildə Meli et al. ictimai GitHub repozitoriyalarının 13%-ni tədqiq etdi və depolarda yoxlanılan 200K-dan çox API açarı və token tapdı [4]. Sirlər yalnız tərtibatçılar tərəfindən versiyaya nəzarət sistemi (VNS) depolarına daxil edilmir, həm də Android və iOS proqram paketlərində saxlanılır. Təxminən 16 min Android tətbiqini tərsinə çevirən bir təhlükəsizlik araşdırması şirkətinin məlumatına görə, hər 200 Android proqramından biri Twitter və AWS API açarları [5] kimi həssas məlumatları sızdırır.

VNS repozitoriyalarında proqram sirrinin olması təhlükəsiz inkişaf üçün adekvat məxfi idarəetmə təcrübələrinin inteqrasiyasını tələb edir. Bununla belə, sirlərin idarə edilməsi ilə bağlı hərtərəfli təcrübələr toplusunun olmaması səbəbindən belə inteqrasiya çətin ola bilər. Məsələn, tərtibatçılar sirləri saxlamaq üçün ən yaxşı təcrübələri tapmaq üçün onlayn forumlara müraciət edirlər. Təcrübəçilərə sirlərin ifşasını məhdudlaşdırmaqda kömək etmək üçün gizli idarəetmə təcrübələri sistematik şəkildə əldə edilə bilər. Bundan əlavə, əldə edilmiş təcrübələr toplusu praktikantlar tərəfindən mövcud gizli idarəetmə təcrübələri üçün müqayisə nöqtəsi kimi istifadə edilə bilər.

Mikroservis arxitekturasında, nəhəng proqram təminatı çox kiçik müstəqil hissələrə bölünür və bu şəkildə reallaşdırılır. Miqyasalana bilmək və hər zaman əlçatanlığı təmin etmək üçün bu servislər çox sayda olur və müxtəlif data mərkəzlərində eyni anda işləyə bilir. Miqyaslanmada və əlçatanlıqda mikroservisin köməyə gələn bu xüsusiyyəti başqa problem üçün qapı aralayır. Belə ki, bu xüsusiyyət, yəni, çox saylı servislər gizli məlumatların paylaşılmasında çətinliklər yaşayır. Burada gizli məlumatlar dedikdə şifrələr, API açarları və digər həssas məlumatlar aiddir.

Gizli məlumatların idarə olunması istənilən təhlükəsiz sistemin vacib aspektidir. Mikroservis arxitekturasında gizli məlumatlar adətən gizli məlumatların idarə edilməsi sistemi kimi tanınan mərkəzi depoda saxlanılır. Gizli məlumatların idarə olunması sistemi gizli məlumatların etibarlı şəkildə saxlanmasına, idarə olunmasına və müvafiq mikroservislərə paylanmasına cavabdehdir.

Gizli məlumatların idarə olunması sistemində ən böyük problemlərdən biri yalnız səlahiyyətli mikroservislərin ehtiyac duyduqları gizli məlumatlara çıxışının təmin edilməsidir. Bunun üçün mikroservislərin kimliyini yoxlaya bilən və gizli məlumatalara girişlə bağlı siyasətləri tətbiq edə bilən möhkəm girişə nəzarət mexanizmi tələb olunur. Məsələn, bəzi mikroservislər gizli məlumatalara yalnız oxumaq üçün giriş tələb edə bilər, digərləri isə oxumaq-yazmaq imkanı tələb edə bilər.

Başqa bir problem gizli məlumatların təsadüfən sızmaması və ya ifşa olunmamasını təmin etməkdir. Mikroservislər müxtəlif maşınlarda və mühitlərdə yerləşdirilə bilər və onlar arasında gizli məlumatların paylaşılması tələb oluna bilər. Bu, səhv konfiqurasiya və ya insan səhvi nəticəsində gizli məlumatın təsadüfən ifşa olunması riskini artırır. Bu riski azaltmaq üçün gizli məlumatlar həm ötürülmə, həm də istirahət zamanı şifrələnməlidir və onlara giriş yoxlanılmalı və nəzarət edilməlidir.

Gizli məlumatların mütəmadi olaraq dəyişdirilməsi gizli məlumatların idarə edilməsi prosesinin başqa çox vacib aspektlərindən biridir. Zaman keçdikcə gizli məlumatlar hər hansı məlumat sızıntısı və ya başqa bir təhlükəsizlik insidentinə görə pozula və ya oğurlana bilər. Buna görə də, bu cür hadisələrin təsirini azaltmaq üçün gizli məlumatları mütəmadi olaraq dəyişdirmək vacibdir. Bunun üçün yeni gizli məlumatların yaradılması və müvafiq mikroservislərdə yenilənməsi üçün etibarlı mexanizm tələb olunur.

2.3. Mikroservis arxitekturasında gizli məlumatların idarəedilməsi həlləri

Gizli məlumatların idarə edilməsi mikroservis arxitekturasının çox mühüm aspektlərindən biridir. Mikroservislər bir-biriləri ilə şəbəkə üzərindən əlaqə saxladığından, sistemin məxfiliyini və bütövlüyünü təmin etmək üçün güclü bir həllə

ehtiyac vardır. Mikroservis arxitekturasında gizli məlumatların idarəedilməsində bəzi həllər mövcuddur:

- Mərkəzləşdirilmiş gizli məlumat idarəetmə sistemi:

Mərkəzləşdirilmiş gizli məlumat idarəetmə sistemi bütün mikroservislərdə sirləri idarə etmək üçün mərkəzi yer təmin edən xüsusi alət və ya xidmətdir. Bu yanaşma gizli məlumatlara nəzarətin və idarə olunmasının mərkəzləşdirilməsi üstünlüyünü təklif edir, saxlamağı və yeniləməyi asanlaşdırır. Mərkəzləşdirilmiş gizli məlumat idarəetmə sistemlərinə misal olaraq "HashiCorp Vault" və "AWS Secrets Manager" daxildir.

- Gizli məlumatlar servis kimi:

Bir xidmət olaraq "Gizli məlumatlar servis kimi" xidməti (SaaS) gizli məlumatları idarə etmək üçün təhlükəsiz və genişlənə bilən bir yol təqdim edən bulud əsaslı bir həlldir. Bu yanaşma gizli idarəetmə prosesinin üçüncü tərəf provayderinə verilməsini nəzərdə tutur. Bu kimi xidmətlərə misal olaraq "AWS Secrets Manager" və "Google Cloud Key Management Service" daxildir.

- Konteyner əsaslı gizli məlumat idarəetmə:

Konteyner əsaslı gizli məlumat idarəetmə gizli məlumatları konteynerin içərisində və ya paylaşılan həcmdə saxlamağı və idarə etməyi əhatə edir. Bu yanaşma sirləri proqram kodundan və konfiqurasiya fayllarından ayrı saxlamağın üstünlüyünü təklif edir, sirləri idarə etməyi və döndərməyi asanlaşdırır. Açıq mənbəli konteyner orkestrasiyası vasitəsi olan Kubernetes, daxili konteynerə əsaslanan gizli idarəetmə sistemini təmin edir.

- Gizli məlumat olmayan arxitektura:

Gizli məlumat olmayan arxitektura sirlərin saxlanması ehtiyacını tamamilə aradan qaldıran yeni bir tendensiyadır. Bunun əvəzinə, həssas resurslara girişi idarə etmək üçün mərkəzləşdirilmiş autentifikasiya və avtorizasiya sisteminə əsaslanır. Bu yanaşma hücum səthinin azaldılması və gizli məlumat idarəetmə prosesinin sadələşdirilməsi üstünlüyü təklif edir. Bu arxitekturaya misal olaraq "OAuth2" və "OpenID Connect" kimi texnologiyaları göstərə bilərik.

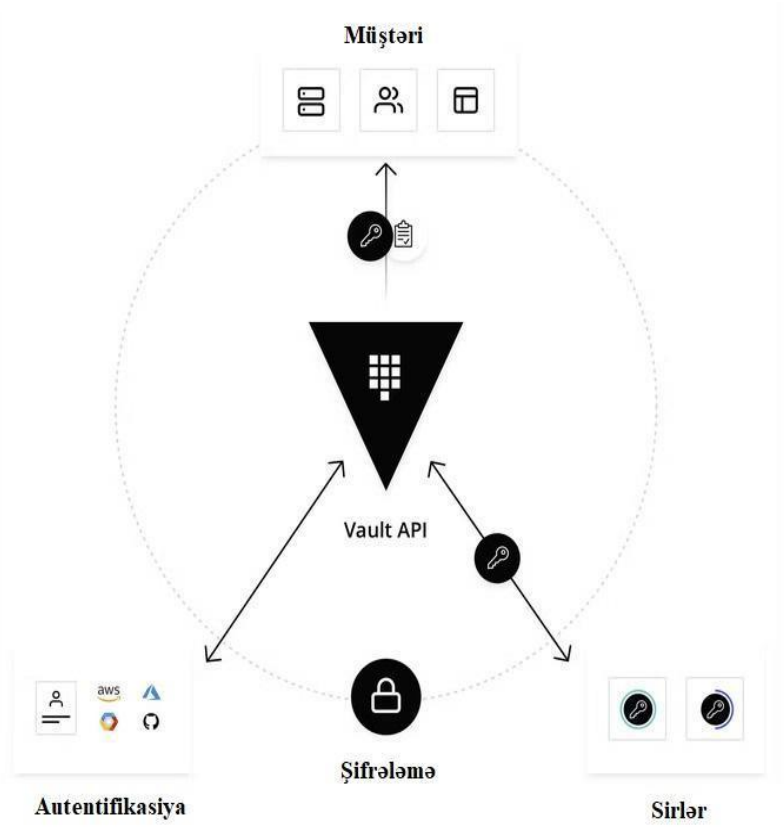
Gizli məlumatların idarə edilməsi güclü və təhlükəsiz həll tələb edən mikroservis arxitekturasının kritik aspektidir. Yuxarıda müzakirə edilən həllər mikroservislərdə sirləri idarə etmək üçün müxtəlif yanaşmalar təqdim edir və hər birinin öz üstünlükləri və mənfi cəhətləri var. Hər hansı bir təhlükəsizlik həlli kimi, sistemin xüsusi ehtiyaclarını qiymətləndirmək və bu tələblər əsasında uyğun həlli seçmək vacibdir.

Hashicorp Vault proqramı

HashiCorp Vault şəxsiyyətə əsaslanan sirlər və şifrələmə idarəetmə sistemidir. Sirr, API şifrələmə açarları, parollar və sertifikatlar kimi girişi ciddi şəkildə idarə etmək istədiyiniz hər şeydir. Vault identifikasiya və avtorizasiya üsulları ilə bağlanan şifrələmə xidmətləri təqdim edir. Vault UI, CLI və ya HTTP API istifadə edərək, sirlərə və digər həssas məlumatlara giriş təhlükəsiz şəkildə saxlanıla və idarə oluna, ciddi şəkildə idarə oluna (məhdudlaşdırıla) və yoxlanıla bilər.

Müasir sistem verilənlər bazası etimadnamələri, xarici xidmətlər üçün API açarları, xidmət yönümlü arxitektura kommunikasiyası üçün etimadnamələr və s. daxil olmaqla çoxlu sirlərə giriş tələb edir. Xüsusilə bu, platforma ola biləcəyi üçün kimin hansı sirlərə daxil olduğunu anlamaq çətin ola bilər. spesifik. Xüsusi bir həll olmadan açarların yuvarlanması, təhlükəsiz saxlanması və təfərrüatlı audit qeydlərinə əlavə etmək demək olar ki, mümkün deyil. Vault işə girdiyi yerdir.

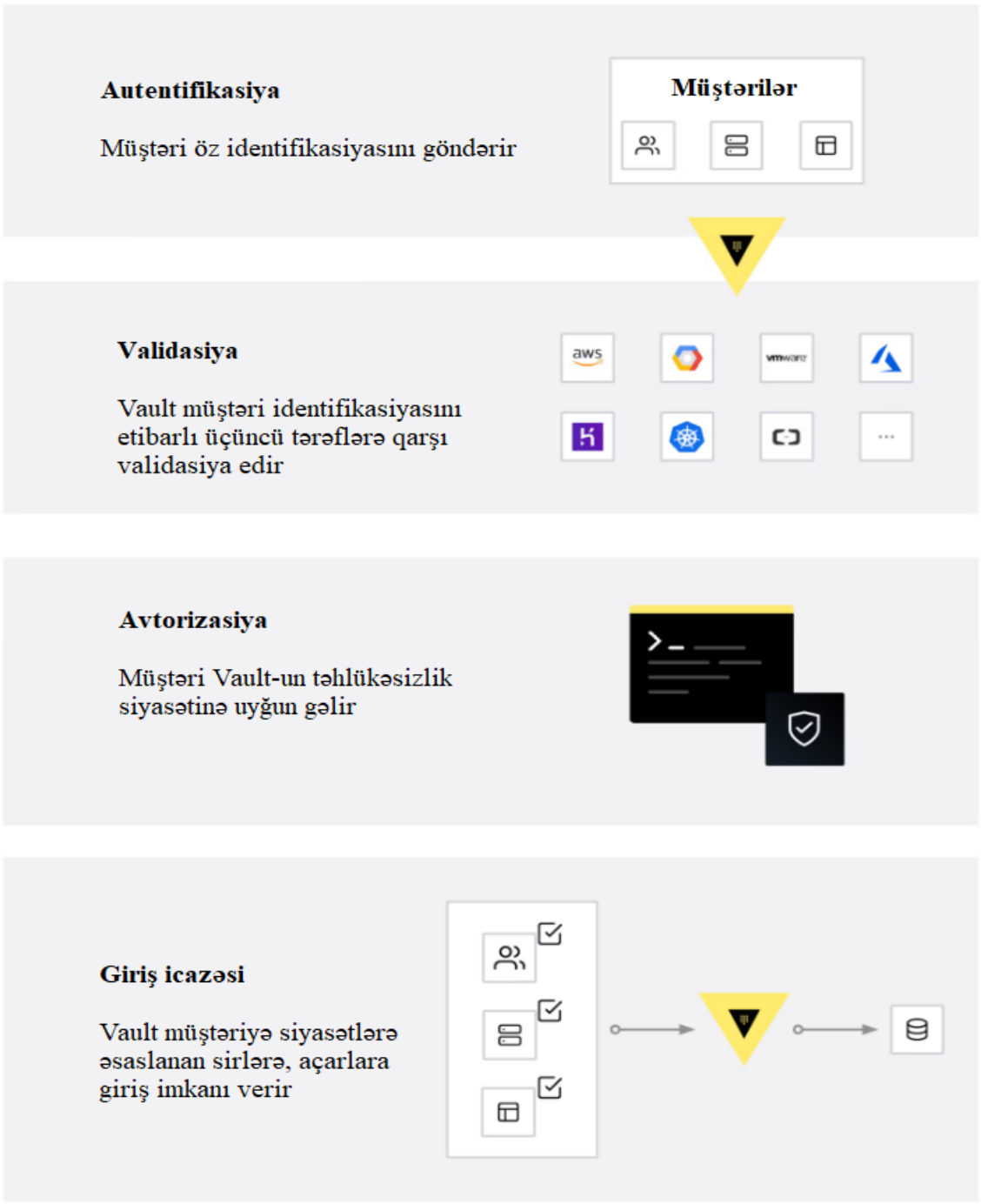
Vault müştərilərə (istifadəçilər, maşınlar, proqramlar) sirrlərə və ya saxlanılan həssas məlumatlara girişi təmin etməzdən əvvəl onları yoxlayır və onlara icazə verir (Şəkil 2.2).



Şəkil 2.2. Hashicorp Vault və imkanları

Vault necə işləyir?

Vault əsasən tokenlərlə işləyir və token müştərinin siyasəti ilə əlaqələndirilir. Hər bir siyasət yola əsaslanır və siyasət qaydaları hər bir müştəri üçün hərəkətləri və yollara əlçatanlığı məhdudlaşdırır. Vault ilə siz tokenləri əl ilə yarada və onları müştərilərinizə təyin edə bilərsiniz və ya müştərilər daxil olub nişan ala bilərlər. Aşağıdakı təsvir Vault-un əsas iş prosesini göstərir. (Şəkil 2.3)



Şəkil 2.3. Vault-un əsas iş prinsipi

Əsas Vault iş axını dörd mərhələdən ibarətdir:

Autentifikasiya: Vault-da identifikasiya müştərinin Vault-un dediklərinə uyğun olub-olmadığını müəyyən etmək üçün istifadə etdiyi məlumatı təqdim etdiyi prosesdir. Müştəri autentifikasiya metodu ilə təsdiqləndikdən sonra bir token yaradılır və siyasətlə əlaqələndirilir.

Validasiya: Vault müştərini Github, LDAP, AppRole və s. kimi üçüncü tərəfin etibarlı mənbələrinə qarşı doğrulayır.

Avtorizasiya: Müştəri Vault təhlükəsizlik siyasətinə uyğunlaşdırılıb. Bu siyasət, müştərinin Vault nişanı ilə hansı API son nöqtələrinə daxil ola biləcəyini müəyyən edən qaydalar toplusudur. Siyasətlər Vault-da müəyyən yollara və əməliyyatlara giriş icazəsi vermək və ya qadağan etmək üçün deklarativ üsul təqdim edir.

Giriş icazəsi: Vault, müştərinin şəxsiyyəti ilə əlaqəli siyasətlərə əsaslanan nişan verməklə sirrlərə, açarlara və şifrələmə imkanlarına giriş imkanı verir. Müştəri daha sonra gələcək əməliyyatlar üçün Vault tokenindən istifadə edə bilər.

Niyə Vault?

Bu gün əksər müəssisələrin təşkilatları arasında yayılan etimadnamələri var. Parollar, API açarları və etimadnamələr düz mətndə, proqram mənbə kodunda, konfiqurasiya fayllarında və digər yerlərdə saxlanılır. Bu etimadnamələr hər yerdə yaşadığı üçün, yayılma kimin nəyə giriş və icazəyə malik olduğunu həqiqətən bilmək çətin və qorxulu edə bilər. Düz mətndə etimadnamələrin olması həm daxili, həm də xarici təcavüzkarlar tərəfindən zərərli hücumların potensialını artırır.

Vault bu problemlər nəzərə alınmaqla hazırlanmışdır. Vault bu etimadnamələrin hamısını götürür və onları bir məkanda müəyyən etmək üçün mərkəzləşdirir ki, bu da etimadnamələrə arzuolunmaz məruz qalmağı azaldır. Lakin Vault istifadəçilərin, tətbiqlərin və sistemlərin autentifikasiyasına və resurslara daxil olmaq üçün açıq şəkildə icazə verildiyinə əmin olmaqla, eyni zamanda müştərilərin hərəkətlərinin tarixini tutan və saxlayan audit izi təqdim etməklə bunu bir neçə addım irəli aparır.

Vault-un əsas xüsusiyyətləri bunlardır:

Təhlükəsiz Gizli Saxlama: İxtiyari açar/dəyər sirləri Vault-da saxlanıla bilər. Vault bu sirləri davamlı yaddaşa yazmazdan əvvəl şifrələyir, beləliklə, xam yaddaşa giriş əldə etmək sirlərinizə daxil olmaq üçün kifayət deyil. Vault diskə, Konsula və s. yaza bilər.

Dinamik sirlər: Vault can generate secrets on-demand for some systems, such as AWS or SQL databases. For example, when an application needs to access an S3 bucket, it

asks Vault for credentials, and Vault will generate an AWS keypair with valid permissions on demand. After creating these dynamic secrets, Vault will also automatically revoke them after the lease is up.

Məlumatların Şifrələnməsi: Vault məlumatları saxlamadan şifrələyə və deşifrə edə bilər. Bu, təhlükəsizlik qruplarına şifrələmə parametrlərini müəyyən etməyə və tərtibatçılara şifrələnmiş məlumatları öz şifrələmə üsullarını dizayn etmədən SQL verilənlər bazası kimi bir yerdə saxlamağa imkan verir.

Lizinq və Yeniləmə: Vault-dakı bütün sirlərin onlarla əlaqəli icarəsi var. İcarənin sonunda Vault avtomatik olaraq həmin sirri ləğv edəcək. Müştərilər daxili yeniləmə API-ləri vasitəsilə icarələri yeniləyə bilərlər.

Ləğv: Vault gizli ləğv üçün daxili dəstəyə malikdir. Vault yalnız bir sirri deyil, bir sirr ağacını, məsələn, müəyyən bir istifadəçi tərəfindən oxunan bütün sirləri və ya müəyyən bir növün bütün sirlərini ləğv edə bilər. Ləğvetmə açarın yuvarlanmasına, eləcə də müdaxilə halında sistemlərin bağlanmasına kömək edir.

Bəzi Vault əmrləri

Vault sirləri idarə etmək və həssas məlumatları qorumaq üçün güclü bir vasitədir. Bu, bir çox proqram və xidmətlər arasında sirləri idarə etmək üçün təhlükəsiz və mərkəzləşdirilmiş bir yol təqdim edir. Başlamağınıza kömək edəcək bəzi əsas Vault əmrləri buradadır

Vault-un işə salınması: Vault-dan istifadə etməyə başlamazdan əvvəl onu işə salmalısınız. Bu, Vault ilə autentifikasiya üçün istifadə olunacaq ilkin kök nişanını yaradır. Vault-u işə salmaq əmri:

‘vault operator init’

Vault-un möhürünün açılması: Vault-u işə saldıqdan sonra onu açmalısınız. Vault-un möhürlənməsi başlatma prosesi zamanı yaradılan çoxsaylı açarları tələb edən bir prosesdir. Anbarın möhürünü açmaq əmridir.

‘vault operator unseal’

Vault-a daxil olmaq: Vault möhürləndikdən sonra, işə salma zamanı yaradılan kök işarəsindən istifadə edərək ona daxil olmalısınız. Vault-a daxil olmaq üçün əmr:
‘vault login’

Vault-a sirlərin yazılması: Vault write əmrindən istifadə edərək, Vault-a sirlər yaza bilərsiniz. Məsələn, Vault-a istifadəçi adı və parol yazmaq üçün aşağıdakı əmrdən istifadə edərdiniz:
‘vault write secret/myapp username=admin password=secret’

Vaultdan sirlərin oxunması: Siz "vault read" əmrindən istifadə edərək Vaultdan sirləri oxuya bilərsiniz. Məsələn, əvvəlki nümunədə Vault-a yazılmış istifadəçi adı və şifrəni oxumaq üçün aşağıdakı əmrdən istifadə edərdiniz:
‘vault read secret/myapp’

Vaultda sirlərin yenilənməsi: Siz "vault write" əmrindən istifadə edərək Vaultdakı sirləri yeniləyə bilərsiniz. Məsələn, əvvəlki nümunədə Vault-a yazılmış parolu yeniləmək üçün aşağıdakı əmrdən istifadə edərdiniz:
‘vault write secret/myapp password=newsecret’

Səhnədən sirlərin silinməsi: Siz "vault delete" əmrindən istifadə edərək, Vaultdan sirləri silə bilərsiniz. Məsələn, əvvəlki nümunələrdə Vault-a yazılmış sirləri silmək üçün aşağıdakı əmrdən istifadə edərdiniz:
‘vault delete secret/myapp’

Bunlar İT mühitində sirləri idarə etmək üçün istifadə edə biləcəyiniz əsas Vault əmrlərindən yalnız bir neçəsidir. İdentifikasiya üsullarını konfiqurasiya etmək, siyasətləri idarə etmək və dinamik sirləri yaratmaq və idarə etmək kimi əlavə funksionallıq təmin edən bir çox başqa əmrlər mövcuddur. Bu əmrlərlə tanış olmaq və sirlərinizin təhlükəsiz və yaxşı qorunduğunu təmin etmək üçün onlardan necə səmərəli istifadə edəcəyinizi başa düşmək vacibdir.

Təhlükəsizlik Modeli

Vault xüsusiyyətinə və idarə etdiyi məlumatların məxfiliyinə görə Vault təhlükəsizlik modeli çox vacibdir. Vault-un təhlükəsizlik modelinin ümumi məqsədi məxfilik, bütövlük, əlçatanlıq, hesabatlılıq, autentifikasiyanı təmin etməkdir.

Bu o deməkdir ki, istirahətdə və tranzitdə olan məlumatlar dinləmə və ya saxtakarlıqdan qorunmalıdır. Müştərilər müvafiq surətdə autentifikasiya olunmalı və məlumatlara daxil olmaq və ya siyasətləri dəyişdirmək səlahiyyətinə malik olmalıdırlar. Bütün qarşılıqlı əlaqə yoxlanıla bilən və mənşəli müəssisəyə qədər izlənilə bilən olmalıdır və sistem onun hər hansı giriş nəzarətindən yan keçmək cəhdlərinə qarşı möhkəm olmalıdır.

Təhdid Modeli

Vault təhlükəsi modelinin müxtəlif hissələri aşağıdakılardır:

- İstənilən Vault rabitəsini dinləmək. Vault ilə müştəri əlaqəsi dinləmədən, həmçinin Vaultdan onun saxlama arxa hissəsinə və ya Vault klaster qovşaqları arasında ünsiyyətdən təhlükəsiz olmalıdır.
- İstirahət və ya tranzit zamanı məlumatların dəyişdirilməsi. İstənilən saxtakarlıq aşkar edilə bilər və Vault-un əməliyyatın işlənməsini dayandırmasına səbəb olmalıdır.
- Doğrulama və ya icazə olmadan məlumatlara və ya nəzarətlərə giriş. Bütün sorğular müvafiq təhlükəsizlik siyasətləri ilə icra edilməlidir.
- Məlumata və ya nəzarətə cavabdehlik olmadan daxil olmaq. Əgər audit qeydi aktivləşdirilibsə, müştəri hər hansı məxfi material almamışdan əvvəl sorğular və cavablar qeyd edilməlidir.
- Saxlanılan sirlərin məxfiliyi. Vault-u yaddaşın arxa tərəfində saxlamaq üçün tərk edən hər hansı məlumat dinləmələrə qarşı təhlükəsiz olmalıdır. Praktikada bu o deməkdir ki, istirahətdə olan bütün məlumatlar şifrələnməlidir.
- Uğursuzluq qarşısında gizli materialın mövcudluğu. Vault əlçatanlığın itirilməsinin qarşısını almaq üçün yüksək əlçatan konfiqurasiyada işləməyi dəstəkləyir.

Aşağıdakılar Vault təhlükəsi modelinin bir hissəsi hesab edilmir:

- Yaddaş arxa ucunun ixtiyari nəzarətindən qorunma. Yaddaşın arxa hissəsinə qarşı ixtiyari əməliyyatlar həyata keçirə bilən təcavüzkar qorunmaq çətin və ya qeyri-mümkün olan istənilən sayda yolla təhlükəsizliyi poza bilər. Nümunə olaraq, təcavüzkar yaddaşın bütün məzmununu silə və ya korlaya bilər və Vault üçün ümumi məlumat itkisinə səbəb ola bilər. Oxunmalara nəzarət etmək qabiliyyəti təcavüzkarın tanınmış vəziyyətdə snapshot almasına və bu, onlar üçün faydalı olarsa, geri qaytarma vəziyyətini dəyişməyə imkan verəcəkdir.
- Gizli materialın mövcudluğunun sızmasına qarşı qorunma. Yaddaşın arxa hissəsindən oxuya bilən təcavüzkar, məxfi saxlanılsa belə, gizli materialın mövcud olduğunu və saxlanıldığını müşahidə edə bilər.
- Çalışan Vault yaddaş analizindən qorunur. Təcavüzkar işləyən Vault instansiyasının yaddaş vəziyyətini yoxlaya bilirsə, məlumatların məxfiliyi pozula bilər.
- Zərərli plaginlərdən və ya əsas hostda kod icrasından qorunma. Təcavüzkar kod icrasını əldə edə və ya əsas hosta yazma imtiyazları əldə edə bilsə, məlumatların məxfiliyi və ya bütövlüyü pozula bilər.
- Vault-a daxil olan müştərilər və ya sistemlərdə qüsurlardan qorunmaq. Təcavüzkar Vault müştərisini (məsələn, sistem, brauzer) ələ keçirə və bu müştərinin Vault etimadnaməsini əldə edə bilsə, bu müştəri ilə əlaqəli imtiyaz səviyyəsi ilə Vault-a daxil ola bilər.

Vault-un istifadə halları

HashiCorp Vault şəxsiyyətə əsaslanan sirlər və şifrələmə idarəetmə sistemidir. Vault müştərilərə (istifadəçilər, maşınlar, proqramlar) sirrlərə və ya saxlanılan həssas məlumatlara girişi təmin etməzdən əvvəl onları yoxlayır və onlara icazə verir.

1. **Ümumi sirr anbarı (General secret storage)**:

İş yükləri getdikcə efemer və qısamüddətli olduqca, uzunmüddətli statik etimadnaməyə malik olmaq böyük təhlükəsizlik təhdid vektoru yaradır. Etibarnamələr təsadüfən sızarsa və ya işçi postunu tərk edərsə, AWS giriş açarını ehtiva edən qeydlər

edərsə və ya kimsə ictimai GH repo-ya S3 giriş nişanını yoxlayırsa? Vault ilə siz qısamüddətli, tam vaxtında etimadnamələr yarada bilərsiniz ki, onların müddəti bitdikdə avtomatik olaraq ləğv edilir. Bu o deməkdir ki, istifadəçilər və təhlükəsizlik qrupları bu etimadnamələri əl ilə ləğv etmək və ya dəyişdirməkdən narahat olmayacaqlar.

- Statik sirlər

Etibarnamələr uzunmüddətli və statik ola bilər, burada onlar dəyişmir və ya nadir hallarda dəyişir. Vault bu sirləri kriptoqrafik maneənin arxasında saxlaya bilər və müştərilər onlardan öz tətbiqlərində istifadə etməyi tələb edə bilərlər.

- Dinamik sirlər

Sirr saxlama ilə əsas dəyər etimadnamələri dinamik şəkildə yaratmaq qabiliyyətidir. Bu etimadnamələr müştərilərin onlara ehtiyacı olduqda yaradılır. Vault həmçinin bu etimadnamələrin həyat dövrünü idarə edə bilər, o cümlədən, lakin bununla məhdudlaşmayaraq, müəyyən müddətdən sonra onları silmək.

Verilənlər bazası etimadnaməsini idarə etməklə yanaşı, Vault Active Directory hesablarınızı, SSH açarlarınızı, PKI sertifikatlarınızı və s. idarə edə bilər.

2. Məlumatların Şifrələnməsi

Bir çox təşkilatlar bulud və ya çox məlumat mərkəzi mühitində proqram məlumatlarını şifrələmək/şifrəsini açmaq üçün həll yolları axtarır; kriptoqrafiyanın tətbiqi və mürəkkəb açar idarəetmə infrastrukturunun saxlanması bahalı və inkişaf etdirilməsi çətin ola bilər. Vault, tranzit zamanı və buludlarda və məlumat mərkəzlərində saxlanılan məlumatların şifrələnməsini sadələşdirmək üçün mərkəzləşdirilmiş açar idarəetməsi ilə bir xidmət kimi şifrələməni təmin edir. Vault başqa yerdə saxlanılan məlumatları şifrələyə/şifrəni açа bilər ki, bu da proqramlara öz məlumatlarını ilkin məlumat anbarında saxlayarkən şifrələməyə imkan verir. Vault-un təhlükəsizlik komandası Vault mühitində məlumatların şifrələnməsinin məsuliyyətini idarə edir və saxlayır, tərtibatçılara lazım olduqda yalnız məlumatların şifrələnməsi/şifrəsinin açılmasına diqqət yetirməyə imkan verir.

3. Şəxsiyyət Əsaslı Giriş

Təşkilatlar müxtəlif buludların, xidmətlərin və sistemlərin yayılması ilə şəxsiyyət genişlənməsini idarə etmək üçün bir yola ehtiyac duyur - hamısı öz şəxsiyyət təminatçıları ilə. Çoxsaylı bulud platformalarında vahid məntiqi identikliyi birləşdirmək üçün həllər həyata keçirməyə çalışarkən təşkilatlar çoxsaylı şəxsiyyət idarəetmə sistemlərini idarə etmək məcburiyyətində qaldıqları üçün təşkilatın təhlükəsizlik infrastrukturunu pozma riski artır. Fərqli platformalar identifikasiya üçün müxtəlif üsulları və konstruksiyaları dəstəkləyir, bu da istifadəçini və ya şəxsiyyəti çoxsaylı etimadnamə formalarında tanımağı çətinləşdirir. Vault sistemlərə və sirlərə broker girişi üçün vahid ACL sistemindən istifadə etməklə bu problemi həll edir və provayderlər arasında şəxsiyyətləri birləşdirir. Şəxsiyyətə əsaslanan giriş ilə təşkilatlar sistem və proqram girişini, müxtəlif buludlar, sistemlər və son nöqtələr arasında autentifikasiyanı tənzimləmək və idarə etmək üçün istənilən etibarlı resurs kimliyindən istifadə edə bilər.

4. **Açar İdarəetmə**

Bulud provayderləri ilə işləmək üçün siz onların təhlükəsizlik xüsusiyyətlərindən istifadə etməyi tələb edir, hansı ki, provayder tərəfindən öz açar idarəetmə sistemində (KMS) buraxılmış və saxlanılan şifrələmə açarları daxildir. Həm bulud daxilində, həm də kənarda şifrləmə açarının həyat dövrünə etibar və nəzarət kökünü saxlamaq tələbiniz də ola bilər. Vault Açar İdarəetmə Sirləri Mühərriki bulud provayderi açarlarının paylanması və həyat dövrünün idarə edilməsi üçün ardıcıl iş axını təmin edir, təşkilatlara KMS provayderlərinə məxsus kriptoqrafik imkanlardan istifadə etməklə Vault-da öz açarlarına mərkəzləşdirilmiş nəzarəti saxlamağa imkan verir.

Möhürləmə/açma

Vault serveri işə salındıqda, o, möhürlənmiş vəziyyətdə başlayır. Bu vəziyyətdə Vault fiziki yaddaşa haradan və necə daxil olmağı bilmək üçün konfiqurasiya edilib, lakin onun heç birinin şifrəsini necə açacağını bilmir.

Açma, Vault-a daxil olmağa imkan verən məlumatın şifrəsini açmaq üçün deşifrə açarını oxumaq üçün lazım olan açıq mətn kök açarının əldə edilməsi prosesidir.

Açılışdan əvvəl Vault ilə demək olar ki, heç bir əməliyyat mümkün deyil. Məsələn, autentifikasiya, montaj cədvəllərini idarə etmək və s. hamısı mümkün deyil. Mümkün olan yeganə əməliyyatlar Anbarın möhürünü açmaq və möhürün vəziyyətini yoxlamaqdır.

Vault tərəfindən saxlanılan məlumatlar şifrələnir. Vault məlumatların şifrəsini açmaq üçün şifrələmə açarına ehtiyac duyur. Şifrələmə açarı həmçinin verilənlərlə (açarlıqda) saxlanılır, lakin kök açar kimi tanınan başqa şifrələmə açarı ilə şifrələnir.

Buna görə də, verilənlərin şifrəsini açmaq üçün Vault kök açarı tələb edən şifrələmə açarının şifrəsini açmalıdır. Açılış bu kök açara daxil olmaq prosesidir. Kök açar bütün digər Vault məlumatları ilə birlikdə saxlanılır, lakin başqa bir mexanizmlə şifrələnir: möhürü açma açarı.

Vault məlumatlarının əksəriyyəti açarlıqdakı şifrələmə açarından istifadə etməklə şifrələnir; açarlıq kök açarı ilə şifrələnir; və kök açar açar açarı ilə şifrələnir.

Vault zavod konfiqurasiyası Şamir möhüründən istifadə edir. Açar açarı operatora tək açar kimi paylamaq əvəzinə, Vault açarı səhmlərə bölmək üçün Şamirin Gizli Paylaşımı (Shamir’s Secret Sharing) kimi tanınan alqoritmdən istifadə edir. Açılan açarın yenidən qurulması üçün müəyyən bir səhm həddi tələb olunur, bu da kök açarın şifrəsini açmaq üçün istifadə olunur.

Bu, möhürün açılması prosesidir: açarı yenidən qurmaq və kök açarın şifrəsini açmaq üçün kifayət qədər səhmlər mövcud olana qədər səhmlər bir-bir əlavə edilir (istənilən qaydada).

- Açma

Açma prosesi "vault operator unseal" əmri ilə və ya API vasitəsilə həyata keçirilir. Bu proses statistik xarakter daşıyır: hər bir açar birdən çox müştəri maşınından bir neçə mexanizm vasitəsilə daxil edilə bilər və o işləyəcək. Bu, daha yaxşı təhlükəsizlik üçün kök açarın hər bir paylaşımının fərqli müştəri maşınında olmasına imkan verir.

Qeyd edək ki, Şamir möhürünü çoxsaylı qovşaqlarla istifadə edərkən, hər bir qovşaq tələb olunan səhm həddi ilə möhürlənməlidir. Hər bir qovşağın qismən açılması klaster üzrə paylanmır.

Bir dəfə Vault möhür açma prosesi baş verdikdən sonra bu hallar baş verənədək Vault möhürü açıq vəziyyətdə qalacaq:

- ❖ API vasitəsilə yenidən möhürlənir
- ❖ Server yenidən işə salınır.
- ❖ Vault-un yaddaş qatı bərpa olunmayan xəta ilə qarşılaşır.

- Möhürləmə

Vault-u möhürləmək üçün API də var. Bu, kök açarı yaddaşa atacaq və onu bərpa etmək üçün başqa bir möhürləmə prosesi tələb olunacaq. Mühürləmə yalnız kök imtiyazları olan bir operator tələb edir.

Beləliklə, aşkar edilmiş müdaxilə olarsa, zərərləri minimuma endirmək üçün Vault məlumatları tez bir zamanda kilidlənə bilər. Kök açar paylaşımlarına giriş olmadan ona yenidən daxil olmaq mümkün deyil.

- Avtomatik açma

Avtomatik açma möhürün açarını təhlükəsiz saxlamağın əməliyyat mürəkkəbliyini azaltmağa kömək etmək üçün hazırlanmışdır. Bu funksiya istifadəçilərdən möhür açma açarının təhlükəsizliyini etibarlı cihaz və ya xidmətə həvalə edir. Başlanğıcda Vault möhürü həyata keçirən cihaz və ya xidmətə qoşulacaq və ondan yaddaşdan oxunan Vault əsas açarının şifrəsini açmağı xahiş edəcək.

Vault-da möhürlənmədən başqa müəyyən əməliyyatlar var ki, onların yerinə yetirməsi üçün istifadəçilərin kvorumunu tələb edir, məs. kök işarəsi yaratmaq. Şamir möhüründən istifadə edərkən bu əməliyyatlara icazə vermək üçün möhür açma açarları təmin edilməlidir. Avtomatik açma istifadə edərkən bu əməliyyatlar əvəzinə bərpa açarları tələb olunur.

Necə ki, Şamir möhürü ilə işə salma prosesi möhürü açma düymələrini verir, avtomatik açma ilə başlatma isə bərpa açarlarını verir.

API-dən istifadə edərək Vault qovşağını möhürləmək hələ də mümkündür. Bu halda Vault yenidən işə salınana qədər möhürlənmiş qalacaq və ya möhürdən çıxarma API-si istifadə olunana qədər, Avtomatik Açıqlama ilə Şamir ilə təmin ediləcək açar fraqmentlərinin əvəzinə bərpa açarı fraqmentləri tələb olunur. Proses eyni olaraq qalır.

- Bərpa açarı

HSM və ya KMS-dən istifadə edərkən Vault işə salındıqda, möhürlənmə açarlarının operatora qaytarılması əvəzinə bərpa açarları qaytarılır. Bunlar HSM və ya KMS olmadan işləyərkən Vault-un möhürləmə açarlarına münasibətdə olduğu kimi, Şamirin Gizli Paylaşımı vasitəsilə bölünən daxili bərpa açarından yaradılıb.

Başlama və yenidən açarla bağlı təfərrüatlar aşağıdakılardır. "generate-root" kimi bərpa açarlarından istifadə edən əməliyyatı yerinə yetirərkən, maneəni açmaq düymələri deyil, bu məqsəd üçün bərpa düymələrinin seçilməsi avtomatikdir.

- Yenidən açar

Açma açarı: Vault-un möhürləmə açarı CLI-dən və ya uyğun API çağırışlarından normal kassa operatoru yenidən açar əməliyyatından istifadə etməklə yenidən açıla bilər. Yenidən açar əməliyyatı bərpa açarları həddinə çatmaqla icazə verilir. Yenidən açardan sonra yeni maneə açarı HSM və ya KMS tərəfindən sarılır və əvvəlki açar kimi saxlanılır; bərpa açarlarını təqdim etmiş istifadəçilərə qaytarılmır.

Bərpa açarı: Bərpa açarı səhmlərin sayını dəyişdirmək və ya müxtəlif PGP açarları vasitəsilə müxtəlif açar sahiblərini hədəfləmək üçün yenidən daxil edilə bilər. Vault CLI-dən istifadə edərkən bu, "vault operator rekey" üçün "-target=recovery" bayrağından istifadə etməklə həyata keçirilir.

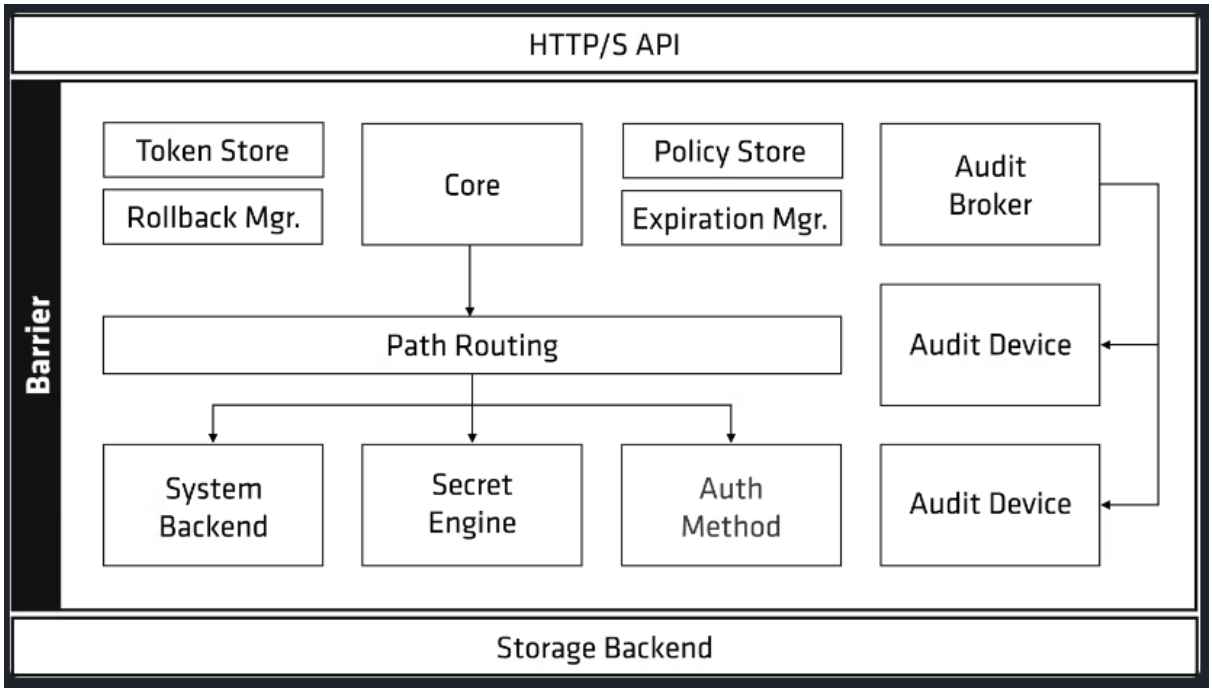
- Möhür miqrasiyası

Möhür miqrasiya prosesi fasiləsiz həyata keçirilə bilməz və möhür tətbiqlərinin texniki əsaslarına görə, proses sizdən bütün klasteri qısa müddətə aşağı salmağı tələb edir. Müəyyən fasilələrlə qarşılaşmaq qaçınılmaz ola bilsə də, biz hesab edirik ki, möhürlərin dəyişdirilməsi nadir hadisədir və dayanma müddətinin narahatlığı məqbul bir əvəzdir.

Arxitektura

Vault çoxsaylı fərqli komponentləri olan mürəkkəb sistemdir. Bu səhifə sistem arxitekturasını təfərrüatlandırır və Vault istifadəçilərinə və tərtibatçılarına əməliyyat nəzəriyyəsini başa düşərkən zehni model qurmağa kömək etməyə ümid edir.

Şəkil 2.4-də Vault-un arxitekturası göstərilmişdir.



Şəkil 2.4. Hashicorp Vault-un arxitekturası

Vault-un manea kimi adlandırılan şifrələmə qatı Vault məlumatlarının şifrələnməsi və deşifrə edilməsi üçün cavabdehdir. Vault serveri işə salındıqda, məlumatları saxlama arxa ucuna yazır. Yaddaşın arxa hissəsi maneədən kənarda yerləşdiyinə görə, o, etibarsız hesab olunur, ona görə də Vault onları yaddaş arxasına göndərməzdən əvvəl onları şifrələyəcək. Bu mexanizm təmin edir ki, zərərli təcavüzkar yaddaşın arxa hissəsinə giriş əldə etməyə cəhd edərsə, Vault məlumatların şifrəsini açana qədər şifrələnmiş qaldığı üçün məlumatların mühafizəsi mümkün olmayacaq. Saxlama arxa ucu verilənlərin qorunduğu və serverin yenidən işə salınması zamanı əlçatan olduğu davamlı məlumat qatını təmin edir.

Vault serveri işə salındıqda, o, möhürlənmiş vəziyyətdə başlayır. Vault üzərində hər hansı əməliyyat həyata keçirilməzdən əvvəl onun möhürü açılmalıdır. Bu, möhürü açma düymələrini təmin etməklə həyata keçirilir. Vault işə salınması zamanı o, bütün Vault məlumatlarını qorumaq üçün istifadə edilən şifrələmə açarı yaradır. Bu açar bütün digər Vault məlumatları ilə birlikdə saxlanılan, lakin başqa mexanizmlə şifrələnən kök açarı ilə qorunur: möhürü açma açarı.

Varsayılan olaraq, Vault möhürü açma düyməsini konfiqurasiya edilmiş sayda qırıqlara (açar paylaşımlar və ya möhür açma açarları) bölmək üçün Şamirin Gizli

Paylaşımından istifadə edir. Açar açarı yenidən qurmaq üçün dəqiq sayda qırıntılar tələb olunur, sonra isə Vault-un kök açarının şifrəsini açmaq üçün istifadə olunur.

Həm səhmlərin sayı, həm də tələb olunan minimum parça sayı müəyyən edilə bilər. Şamirin texnikası söndürülə bilər və kök açarı birbaşa möhürü açmaq üçün istifadə edilə bilər. Vault şifrələmə açarını əldə etdikdən sonra yaddaşın arxa hissəsindəki məlumatların şifrəsini açır və möhürlənməmiş vəziyyətə daxil olur. Möhürləndikdən sonra Vault konfiqurasiya edilmiş audit cihazlarını, auth üsullarını və gizli mühərrikləri yükləyir.

Audit cihazlarının konfiqurasiyası, auth metodları və sirr mühərrikləri təhlükəsizliyə həssasdır və Vault-da saxlanılır. İcazələri olan istifadəçilər onları dəyişdirə bilər və maneədən kənarda müəyyən edilə bilməz. Onları Vault-da saxlamaqla dəyişikliklər ACL sistemi tərəfindən qorunur və audit qeydləri ilə izlənilir.

Vault möhürü açıldıqdan sonra sorğular HTTP API-dən nüvəyə işlənə bilər. Əsas sistem vasitəsilə sorğu axınını idarə edir, ACL-ləri tətbiq edir və audit qeydinin aparılmasını təmin edir.

Müştəri Vault-a ilk dəfə qoşulduqda, müştəri autentifikasiya etməlidir. Vault konfiqurasiya edilə bilən autentifikasiya üsulları təqdim edir və istifadə edilən autentifikasiya mexanizmi daxilində çeviklik təklif edir. İstifadəçi adı/parol və ya GitHub kimi mexanizmlər operatorlar üçün istifadə oluna bilər, tətbiqlər isə autentifikasiya üçün ictimai/məxfi açarlardan və ya tokenlərdən istifadə edə bilər. Əsasdan və auth metoduna axan identifikasiya sorğusu sorğunun etibarlı olub olmadığını müəyyən edir və əlaqəli siyasətlərin siyahısını qaytarır.

Siyasətlər sadəcə adlandırılmış ACL qaydasıdır. Məsələn, "kök" siyasəti daxilidir və bütün resurslara girişə icazə verir. Siz yollar üzərində dəqiq nəzarətlə istənilən sayda adlanmış siyasət yarada bilərsiniz. Vault icazəli giriş rejimində işləyir, yəni siyasət vasitəsilə açıq şəkildə giriş icazəsi verilməyənə qədər fəaliyyətə icazə verilmir. İstifadəçinin birdən çox siyasəti əlaqəli ola biləcəyi üçün siyasət icazə verdikdə hərəkətlərə icazə verilir. Siyasətlər daxili siyasət mağazası tərəfindən saxlanılır və idarə olunur. Bu daxili mağaza həmişə "sys/" üzərində quraşdırılmış sistem arxa hissəsi vasitəsilə təsirlənir.

Autentifikasiya baş verdikdən və autentifikasiya metodu müvafiq siyasətlər toplusunu təmin etdikdən sonra yeni müştəri nişanı yaradılır və token mağazası tərəfindən idarə olunur. Bu müştəri nişanı gələcək sorğular etmək üçün istifadə olunur. Bu token metodu istifadəçi daxil olduqda vebsayt tərəfindən göndərilən kukiyə bənzəyir. Auth metodu konfiqurasiyasından asılı olaraq, müştəri tokeninin onunla əlaqəli icarəsi ola bilər və etibarsızlığın qarşısını almaq üçün vaxtaşırı yenilənməsi tələb oluna bilər.

Əsas audit brokerinə sorğuları və cavabları qeyd edir, sorğuları bütün konfiqurasiya edilmiş audit cihazlarına paylayır. Sorğu axınından kənarda, əsas xüsusi fon fəaliyyətləri həyata keçirir. İcarənin idarə edilməsi çox vacibdir, müddəti bitmiş müştəri nişanları və ya sirlərini avtomatik olaraq ləğv etməyə imkan verir. Əlavə olaraq, Vault geri qaytarma meneceri ilə qabaqcadan yazma qeydindən istifadə etməklə xüsusi qismən uğursuzluq hallarını idarə edir. Bu, əsas daxilində şəffaf şəkildə idarə olunur və istifadəçi tərəfindən görünmür.

Təşkilatlar həssas məlumatları qorumaq və sistemlərini qorumaq üçün səy göstərdikcə, möhkəm və hərtərəfli təhlükəsizlik həllərinə ehtiyac hər şeydən üstün olur. HashiCorp Vault sirləri idarə etmək, məlumatların şifrələnməsi və kritik infrastrukturun qorunması üçün təhlükəsiz və mərkəzləşdirilmiş yanaşma təmin edən aparıcı vasitə kimi ortaya çıxır.

Həssas məlumatların qorunmasında əsas problemlərdən biri şifrələmə açarlarının idarə olunmasıdır. Vault bu problemi Paylanmış Açar İdarəetmə vasitəsi ilə həll edir, bu da heç bir güzəşt nöqtəsinin şifrələnmiş məlumatlara icazəsiz giriş verə bilməməsini təmin edir.

Paylanmış açar idarəetmə modelində Vault əsas açarı əsas paylaşımlar adlanan bir neçə hissəyə bölmək üçün Şamirin Gizli Paylaşım alqoritmindən istifadə edir. Bu əsas paylaşımlar daha sonra etibarlı şəxslər və ya "açarları açmaq" kimi tanınan aparat cihazları klasterində paylanır. Heç bir fərdi şəxs və ya cihaz icazəsiz giriş riskini azaldan tam əsas açara malik deyil. Tipik olaraq, əsas açarı kollektiv şəkildə yenidən qurmaq və Vault-un kilidini açmaq üçün əvvəlcədən təyin edilmiş möhürləmə açarları tələb olunur.

Bu yanaşma yalnız nasazlığa dözümlülük təmin etmir, həm də bir uğursuzluq nöqtəsinin qarşısını alır. Bəzi möhürləmə açarları təhlükə altına düşsə və ya itirilsə belə, qalan möhürləmə açarları şifrələnmiş məlumatlara davamlı girişi təmin etməklə yenə də əsas açarı yenidən qura bilər.

Möhürləmə/açarları açmaq mexanizmi Vault-da şifrələnmiş məlumatların istirahətdə qorunmasına kömək edən mühüm təhlükəsizlik xüsusiyyətidir. Vault möhürləndikdə, sirrlərə və kriptoqrafik əməliyyatlara giriş qadağan edilir və məlumatı əlçatmaz edir. Təcavüzkar əsas yaddaşa fiziki giriş əldə etsə və ya host sistemini pozsa belə, möhürləmə icazəsiz girişə qarşı qoruyucu vasitə kimi çıxış edir.

Vault möhürünün açılması əvvəllər qeyd olunan paylanmış açar idarəetmə yanaşmasının istifadəsini tələb edir. Bu, kifayət qədər sayda açarların toplanması və əsas açarın yenidən qurulması üçün birləşdirilməsini əhatə edir. Möhür açıldıqdan sonra Vault fəaliyyətə başlayır və səlahiyyətli istifadəçilərə və tətbiqlərə sirrlərə daxil olmağa və kriptoqrafik əməliyyatlar həyata keçirməyə imkan verir.

Bu möhürləmə/açma mexanizmi müxtəlif ssenarilərdə Vault-un təhlükəsizliyinin təmin edilməsində mühüm rol oynayır. Məsələn, planlaşdırılan texniki xidmət və ya sistemin dayanması zamanı administratorlar məlumatların məxfiliyini və bütövlüyünü təmin etmək üçün Vault-u möhürləyə bilərlər. Əlavə olaraq, təhlükəsizlik pozuntusu və ya şübhəli bir kompromis halında, Vault-u möhürləmək imkanı sirrin məxfiliyini qoruyaraq, icazəsiz girişin qarşısını dərhal alır.

Düzgün autentifikasiyanın təmin edilməsi həssas məlumatların və resursların təhlükəsizliyinin əsas aspektidir. Güclü sirlərin idarə edilməsi vasitəsi olan HashiCorp Vault sirrlərə və kriptoqrafik əməliyyatlara girişi yoxlamaq və nəzarət etmək üçün möhkəm autentifikasiya mexanizmləri təqdim edir. Bu yazıda təşkilatlara güclü təhlükəsizlik təcrübələrini tətbiq etməyə və qiymətli aktivlərini qorumağa imkan verən Vault tərəfindən təklif olunan bəzi əsas autentifikasiya üsullarını araşdıracağıq.

Token əsaslı autentifikasiya: Token əsaslı autentifikasiya Vault-da istifadəçilərin və proqramların autentifikasiyası üçün istifadə edilən əsas üsuldur. İstifadəçi və ya proqram uğurla autentifikasiya edildikdə, Vault Vault API ilə sonrakı qarşılıqlı əlaqə üçün etimadnamə kimi xidmət edən nişan verir.

Tokenləri iki kateqoriyaya bölmək olar: kök tokenlər və kök olmayan tokenlər. Kök tokenlər Vault əməliyyatları üzərində tam nəzarəti təmin edən super istifadəçi imtiyazları verir, qeyri-root tokenləri isə təyin edilmiş siyasətlərə əsasən məhdud girişə malikdir. Varsayılan olaraq, kök tokenlər Vault-un işə salınması prosesi zamanı yaradılır və onların istifadəsini təhlükəsiz saxlamaq və məhdudlaşdırmaq vacibdir.

Kök olmayan tokenlər istifadəçilər və ya proqramlar üçün yaradılır və Vault daxilində sirrlərə və əməliyyatlara malik olduqları giriş səviyyəsini təyin edərək xüsusi siyasətlərlə konfiqurasiya edilə bilər. Tokenlər həmçinin məhdud müddət və ya yenilənmə intervalı ilə konfiqurasiya edilə bilər ki, bu da vaxtaşırı təkrar autentifikasiyanı təmin edir və uzunmüddətli giriş riskini azaldır.

LDAP və Active Directory (AD) autentifikasiyası: Vault LDAP və Active Directory kimi mövcud autentifikasiya sistemləri ilə problemsiz inteqrasiya edərək təşkilatlara öz mərkəzləşdirilmiş istifadəçi idarəetmə infrastrukturundan istifadə etməyə imkan verir. Vault-u LDAP və ya AD ilə inteqrasiya etməklə, vahid autentifikasiya təcrübəsini təmin etməklə, istifadəçi etimadnamələri mövcud kataloq xidmətlərinə qarşı yoxlana bilər.

LDAP və ya AD identifikasiyası ilə Vault istifadəçiləri LDAP və ya AD etimadnaməsinə əsasən autentifikasiya edə bilər. Müvəffəqiyyətli identifikasiyadan sonra Vault istifadəçiyə təyin edilmiş siyasətlər əsasında sirrlərə və əməliyyatlara giriş icazəsi verən nişan verir. Bu inteqrasiya istifadəçi idarəetməsini asanlaşdırır, giriş nəzarətini mərkəzləşdirir və mövcud autentifikasiya təcrübələrindən istifadə etməklə təhlükəsizliyi artırır.

Çox Faktorlu Autentifikasiya (MFA): İdentifikasiyanı daha da gücləndirmək üçün Vault Multi-factor Authentication (MFA) dəstəkləyir. MFA istifadəçilərdən giriş icazəsi verməzdən əvvəl çoxsaylı yoxlama formalarını təqdim etmələrini tələb etməklə əlavə təhlükəsizlik səviyyəsi əlavə edir.

Vault Okta, Duo və Google Authenticator kimi məşhur MFA provayderləri ilə inteqrasiya edir. İstifadəçilər öz əsas etimadnamələri (istifadəçi adı və parol) ilə autentifikasiya edirlər və sonra mobil proqram və ya aparat nişanı tərəfindən yaradılan birdəfəlik parol (OTP) kimi ikinci dərəcəli autentifikasiya faktoru təqdim edirlər. MFA

əlavə yoxlama addımı tələb etməklə, ilkin etimadnamələr pozulsa belə, icazəsiz giriş riskini əhəmiyyətli dərəcədə azaldır.

AppRole autentifikasiya: AppRole autentifikasiyası maşından maşına (M2M) rabitə üçün nəzərdə tutulmuşdur və tətbiqlərə insan iştirakı və ya interaktiv giriş axınına ehtiyac olmadan Vault ilə autentifikasiya etməyə imkan verir.

AppRole autentifikasiyası Vault daxilində əvvəlcədən təyin edilmiş rolun yaradılmasını və bu rola siyasətlərin təyin edilməsini əhatə edir. Müvafiq etimadnaməyə malik olan proqram autentifikasiya üçün öz rol ID və gizli ID-ni Vault-a təqdim edir. Vault etimadnamələri yoxlayır, autentifikasiya nişanı verir və həmin rol üçün təyin edilmiş siyasətlərə əsasən giriş verir. AppRole autentifikasiyası proqramların Vault ilə etibarlı autentifikasiyası və onların işləməsi üçün lazımi sirləri əldə etməsi prosesini asanlaşdırır.

Autentifikasiya həssas məlumatların və resursların qorunmasının mühüm aspektidir və HashiCorp Vault güclü təhlükəsizlik təcrübələrini tətbiq etmək üçün hərtərəfli autentifikasiya mexanizmləri dəsti təqdim edir. Token əsaslı autentifikasiya, LDAP/AD inteqrasiyası, MFA və AppRole autentifikasiyası ilə Vault təşkilatlara girişi idarə etmək, şəxsiyyətləri yoxlamaq və kritik aktivləri qorumaq imkanı verir.

Vault-un autentifikasiya mexanizmlərindən istifadə etməklə təşkilatlar yalnız səlahiyyətli istifadəçilərin və proqramların sirrlərə daxil ola bilməsini və kriptoqrafik əməliyyatları yerinə yetirməsini təmin edə bilər. Bu xüsusiyyətlər təşkilatlara incə giriş nəzarətini tətbiq etmək, tənzimləmə qaydalarına riayət etmək və öz dəyərli məlumatlarını mərkəzləşdirilmiş və təhlükəsiz şəkildə qorumaq imkanı verir.

Təşkilatlar getdikcə daha təkmilləşmiş kibertəhlükələrlə üzləşdikcə, həssas məlumatların və infrastrukturun təhlükəsizliyi hər şeydən üstün olur. HashiCorp Vault sirləri idarə etmək və kritik sistemləri qorumaq üçün hərtərəfli həll təklif edir. Paylanmış açarların idarə edilməsi və möhürləmə/açma mexanizmi vasitəsilə Vault etibarlı təhlükəsizliyi təmin edir, icazəsiz girişin qarşısını alır və istirahətdə şifrələnmiş məlumatları qoruyur.

Vault-un paylanmış açar idarəçiliyindən istifadə etməklə, təşkilatlar kompromis riskini azaldaraq, şifrələmə açarlarının idarə edilməsi məsuliyyətini bölüşdürə bilər.

Möhürləmə/açma mexanizmi inzibatçılara sirrlərə girişi idarə etməyə və icazəsiz girişdən qorunmağa imkan verən qoruyucu vasitə kimi çıxış edir.

Ümumi təhlükəsizlik strategiyasının tərkib hissəsi kimi, HashiCorp Vault təşkilatlara uyğunluq tələblərinə cavab vermək, təhlükəsizlik vəziyyətini gücləndirmək və kritik aktivləri potensial təhlükələrdən qorumaq səlahiyyətini verir.

Vault konfiqurasiya faylı

Konfiqurasiya faylı Vault arxitekturasının kritik komponentidir. O, yaddaşın arxa hissəsi, dinləyici ünvanı və digər mühüm parametrlər daxil olmaqla Vault konfiqurasiyasını və davranışını müəyyən edir.

Konfiqurasiya faylı Vault üçün müxtəlif parametrləri təyin etmək üçün istifadə edilən JSON formatlı fayldır. Bu parametrlərə yaddaş arxası, dinləyici ünvanı, jurnal səviyyəsi və digər mühüm konfiqurasiya seçimləri daxildir. Konfiqurasiya faylı adətən əməliyyat sistemindən asılı olaraq /etc/vault/config.hcl və ya /usr/local/etc/vault/config.hcl ünvanında yerləşir.

Konfiqurasiya faylında vacib parametrlərdən biri yaddaşın arxa hissəsidir. Bu parametr Vault məlumatlarını harada saxlamalı olduğunu müəyyən edir. Vault, Consul, Amazon S3 və fayl əsaslı yaddaş da daxil olmaqla bir neçə yaddaş arxa ucunu dəstəkləyir. Yaddaşın arxa hissəsi konfiqurasiya faylında "saxlama" blokundan istifadə etməklə müəyyən edilir. Məsələn, Konsulu saxlama arxası kimi istifadə etmək üçün konfiqurasiya faylına aşağıdakı konfiqurasiya bloku əlavə edilə bilər:

Bu konfiqurasiya bloku edir ki, Vault Consul serveri yerli hostda işləyir və 8500 portunu dinləyərək, Konsulun saxlanma ehtiyat hissəsi kimi istifadə istehlak. "Yol" seçimi Vault məlumatlarının saxlanacağı Consul yolunun idarəsidir.

Konfiqurasiya faylındakı digər vacib parametr dinləyicinin ünvanıdır. Bu parametr Vault-un daxil olan əlaqələri dinləməli olduğu şəbəkə ünvanını və portu müəyyən edir. Dinləyicinin ünvanı konfiqurasiya faylında "dinləyici" blokundan istifadə etməklə müəyyən edilir. Məsələn, 8200 portunda bütün şəbəkə interfeyslərini dinləmək üçün konfiqurasiya faylına aşağıdakı konfiqurasiya bloku əlavə edilə bilər.

Konfiqurasiya faylına həmçinin Vault audit jurnalını işə salmaq və konfiqurasiya etmək üçün parametrlər daxildir. Audit jurnalı sorğular, cavablar və səhvlər daxil olmaqla, Vault tərəfindən görülən bütün hərəkətləri qeyd etmək üçün istifadə olunur. Audit jurnalı konfiqurasiya faylındakı "audit" blokundan istifadə etməklə konfiqurasiya edilə bilər. Məsələn, auditi aktivləşdirmək və bütün audit hadisələrini fayla daxil etmək üçün konfiqurasiya faylına aşağıdakı konfiqurasiya bloku əlavə edilə bilər.

Yuxarıda qeyd olunan parametrlərə əlavə olaraq, konfiqurasiya faylına Vault jurnalının səviyyəsini konfiqurasiya etmək üçün parametrlər, performans tənzimləmə seçimləri və s. daxildir. Konfiqurasiya faylı Vault arxitekturasının kritik komponentidir və Vault-u yerləşdirərkən və idarə edərkən onun müxtəlif parametrlərini və seçimlərini başa düşmək vacibdir.

Konfiqurasiya faylı Vault arxitekturasının kritik komponentidir. O, yaddaşın arxa hissəsi, dinləyici ünvanı və digər mühüm parametrlər daxil olmaqla Vault konfiqurasiyasını və davranışını müəyyən edir. Konfiqurasiya faylı adətən əməliyyat sistemindən asılı olaraq /etc/vault/config.hcl və ya /usr/local/etc/vault/config.hcl ünvanında yerləşir. Vault-u yerləşdirərkən və idarə edərkən konfiqurasiya faylındakı müxtəlif parametrləri və seçimləri başa düşmək vacibdir.

2.4. Proqram təminatına lazım olan sirrlərin paylanması

Sirlərin paylanması ciddi problemdir. Əgər bizim sistemimiz paylanmışdırsa, bu daha da çətindir. Bu problemi aradan qaldırmaq üçün bir neçə həllər vardır. Onlardan biri də mərkəzi sirr idarəetmə sistemidir. Vault proqramı sirr idarəetmə sistemidir. Bir çox funksionallığa sahibdir. Yazdığımız proqramın sirrlərini biz rahatlıqla Vault-dan götürə bilirək. Gəlin, buna bir nümunədə baxaq. Sadə bir proqram təminatı ilə sirrlərin paylanmasının necə baş verdiyini aydınlaşdıraq:

Proqram təminatı Java proqramlaşdırma dilində “Spring framework”-dən istifadə edərək yazılmışdır (Şəkil 2.5). Gələn http sorğuları emal edərək geri cavab qaytarır.

```java
package az.edu.aztu.secretmanagement;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SecretManagementApplication {

    public static void main(String[] args) { SpringApplication.run(SecretManagementApplication.class, args); }

}
```

Şəkil 2.5. “Spring framework”-də yazılmış proqramın əsas sinifi

Proqram təminatı iki funksiya yerinə yetirir:

1. Məhsulları verilənlər bazasından gətirmək
2. Seçilən məhsullar üçün ödəniş etməyə imkan vermək

```java
package az.edu.aztu.secretmanagement.model;

import lombok.Data;

import java.math.BigDecimal;

@Data
public class Product {

    private int id;
    private String name;
    private BigDecimal price;

}
```

Şəkil 2.6. Məhsul sinifi

Şəkil 2.6-dəki Məhsul (Product) sinifinin 3 xüsusiyyəti var:

- Id - məhsulun nömrəsini saxlayır
- Name - məhsulun adını saxlayır
- Price - məhsulun qiymətini saxlayır

```java
package az.edu.aztu.secretmanagement.model;

import lombok.Data;

@Data
public class CustomerProductList {

    private int customerId;
    private int productId;
    private int productCount;

}
```

Şəkil 2.7. Müştərinin məhsullarının siyahısı sinifi

Şəkil 2.7-dəki Müştəri məhsullarının siyahısı sinifinin 3 xüsusiyyəti var:

- customerId - məhsul sahibi olan müştərinin nömrəsini saxlayır
- productId - məhsulun nömrəsi
- productCount - müştərinin sahib olduğu məhsul sayı

Şəkil 2.6 və şəkil 2.7-dəki siniflər verilənlər bazasındakı "product" və "customerProductList" cədvəllərinin kodda təsviridir.

Həmin cədvəllər şəkil 2.8-də göstərilmişdir.

```sql
create database vault;

use vault;

create table product(
    productId int,
    productName varchar(20),
    productPrice decimal(10,2)
);

create table customerProductList(
    productId int,
    customerId int,
    productCount int
);
```

Şəkil 2.8. Verilənlər bazasındakı cədvələrin yaradılması sorğuları

Şəkil 2.8-də ilk sətirdə biz verilənlər bazasının yaradıldığını görürük. Növbəti sətirlərdə isə "product" və "customerProductList" cədvəllərinin yaradılır.

“Product” cədvəli özündə məhsullar haqqında məlumatları saxlayır:

- productId - “int” tipində olan sütun özündə məhsulların nömrələrini saxlayır.
- productName - “varchar” tipində olan sütun özündə məhsulun adını saxlayır.
- productPrice - “decimal” tipində olan sütun özündə məhsulun bir ədədinə görə qiyməti özündə saxlayır.

“CustomerProductList” cədvəli özündə müştərinin sahib olduğunu məhsulları saxlayır:

- customerId - “int” data tipində olan sütun özündə müştərinin unikal nömrəsini saxlayır.
- productId - “int” data tipində olan sütun özündə müştərinin sahib olduğu məhsulun unikal nömrəsini saxlayır.
- productCount - “int” data tipində olan sütun özündə müştərinin sahib olduğu məhsulun sayını göstərir.

Bu cədvəllərlə və ya başqa sözlə ifadə etsək verilənlər bazasında olan məhsullarla bağlı məlumatlarla işləmək üçün mənbə kodumuzda “ProductManagementService” servisi vardır (Şəkil 2.9).

```java
package az.edu.aztu.secretmanagement.service;

import az.edu.aztu.secretmanagement.DatabaseConnection;
import az.edu.aztu.secretmanagement.model.Product;
import org.springframework.stereotype.Service;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

@Service
public class ProductManagementService {

    private DatabaseConnection databaseConnection;

    public ProductManagementService(DatabaseConnection databaseConnection) {
        this.databaseConnection = databaseConnection;
    }

    public List<Product> getProductList() {...}

    public Product getProduct(int productId) {...}

    public void saveProductToCustomer(int productId, int customerId, int count) {...}


    private Connection getConnection() {...}
}
```

Şəkil 2.9. "ProductManagementService" sinifi

"ProductManagementService" sinifinin 4 açıq, 1 gizli metodu və 1 ədəd də gizli xüsusiyyəti vardır. Methodlar bunlardır:

- "getProudctList()" açıq metodu verilənlər bazasından məhsulları götürmək üçün istifadə edilir. Metod çağrıldığı zaman "Product" cədvəlindəki məlumatlar metoddan geri qaytarılır. (Şəkil 2.10)

```java
public List<Product> getProductList() {
    List<Product> productList = new ArrayList<>();
    Connection connection = getConnection();
    try {
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery( sql: "select * from product");
        while (resultSet.next()) {
            Product product = new Product();
            product.setId(resultSet.getInt( columnLabel: "productId"));
            product.setName(resultSet.getString( columnLabel: "productName"));
            product.setPrice(resultSet.getBigDecimal( columnLabel: "productPrice"));
            productList.add(product);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return productList;
}
```

Şəkil 2.10. “getProductList” metodu

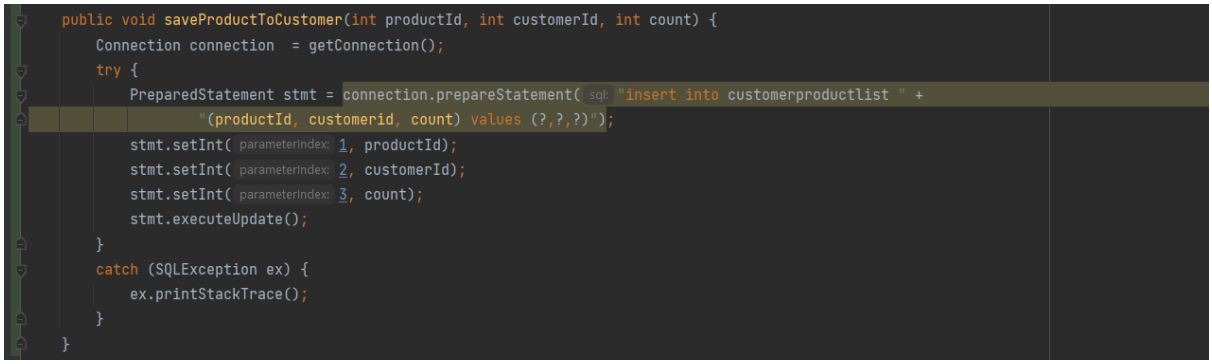
- “getProduct(int productId)” açıq metodu verilənlər bazasından metoda arqument kimi göndərilmiş nömrəli məhsulu geri qaytarır. (Şəkil 2.11)

```java
public Product getProduct(int productId) {
    Connection connection = getConnection();
    try {
        PreparedStatement stmt = connection.prepareStatement( sql: "select * from product where productId = ?");
        stmt.setInt( parameterIndex: 1, productId);
        ResultSet resultSet = stmt.executeQuery();

        Product product = null;
        if (resultSet.next()) {
            product = new Product();
            product.setId(resultSet.getInt( columnLabel: "productId"));
            product.setName(resultSet.getString( columnLabel: "productName"));
            product.setPrice(resultSet.getBigDecimal( columnLabel: "productPrice"));
        }
        return product;
    }
    catch (SQLException ex) {
        ex.printStackTrace();
    }
    return null;
}
```

Şəkil 2.11. “getProduct(int productId)” metodu

- “saveProductToCustomer(int productId, int customerId, int count)” metodu müştəri almaq istədiyi məhusulun ödənişini etdikdən sonra həmin məhsulu müştərinin hesabına əlavə edir. (Şəkil 2.12)



```
public void saveProductToCustomer(int productId, int customerId, int count) {
    Connection connection  = getConnection();
    try {
        PreparedStatement stmt = connection.prepareStatement( sql: "insert into customerproductlist " +
                "(productId, customerid, count) values (?,?,?)");
        stmt.setInt( parameterIndex: 1, productId);
        stmt.setInt( parameterIndex: 2, customerId);
        stmt.setInt( parameterIndex: 3, count);
        stmt.executeUpdate();
    }
    catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

Şəkil 2.12. "saveProductToCustomer(int productId, int customerId, int count)" metodu

- "getConnection()" gizli metodu verilənlər bazasına əlaqə qurmaq üçün lazım olan "Connection" obyektini əldə etmək üçün istifadə edilir. Hansı ki, sinifin digər açıq metodları da bu gizli metodu "connection" obyektini əldə etmək üçün istifadə edir. (Şəkil 2.13)

```
private Connection getConnection() {
    Connection connection = null;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection(
                databaseConnection.getUrl(),
                databaseConnection.getUsername(), databaseConnection.getPassword());
    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    }
    return connection;
}
}
```

Şəkil 2.13. "getConnection()" metodu

"getConnection()" metodu "MySql" verilənlər bazasına əlaqə qura bilmək üçün Java proqramlaşdırma dilinin "Mysql" drayverini işlədir. Bu metodda "ProductManagementService" sinifinin gizli xüsusiyyəti olan "databaseConnection" da istifadə edilmişdir. Həmin xüsusiyyətdən biz verilənlər bazası ilə bağlı məlumatları əldə edirik. (Şəkil 2.13)

```java
package az.edu.aztu.secretmanagement;

import org.springframework.context.annotation.Configuration;
import org.springframework.vault.core.VaultKeyValueOperationsSupport;
import org.springframework.vault.core.VaultTemplate;
import org.springframework.vault.support.VaultResponse;

@Configuration
public class DatabaseConnection {

    private String url;
    private String username;
    private String password;

    private VaultTemplate vaultTemplate;

    public DatabaseConnection (VaultTemplate vaultTemplate) {
        VaultResponse response = vaultTemplate
                .opsForKeyValue( path: "secret", VaultKeyValueOperationsSupport.KeyValueBackend.KV_2).get("database");

        this.url = (String) response.getData().get("url");
        this.username = (String) response.getData().get("username");
        this.password = (String) response.getData().get("password");
    }

    public String getUrl() { return url; }

    public String getUsername() { return username; }

    public String getPassword() { return password; }
}
```

Şəkil 2.14. “DatabaseConnection” sinifi

Şəkil 2.14-də gördüyümüz kimi sinif özündə verilənlər bazasına qoşulmaq üçün lazım olan məlumatları saxlayır:

- Url - verilənlər bazasına qoşulmaq üçün lazım olan mənbə məlumatlarını saxlayır
- Username - verilənlər bazasına qoşulmaq üçün istifadə olunan istifadəçi adı
- Password - verilənlər bazasına qoşulmaq üçün istifadə olunan istifadəçinin şifrəsi

“DatabaseConnection” sinfinin yuxarıda izah edilmiş xüsusiyyətlərindən əlavə “vaultTemplate” adlı başqa bir gizli xüsusiyyəti var. Əgər şəkil 2.14-də kodlara fikir versək konstruktorda “vaultTemplate” obyekti istifadə edilmişdir.

“VaultTemplate” haqqında daha ətraflı danışsaq, bu sinif verilənlər bazasına əlaqə qurmaq üçün lazım olan məlumatları “DatabaseConnection”-a təmin edir.

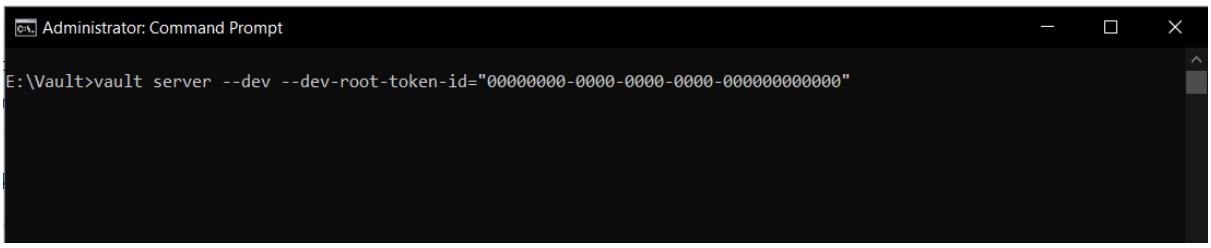
“VaultTemplate” məlumatları Hashicorp Vault proqramından əldə edir. Bunun üçün o mənbə kodunda konfiqurasiya edilmişdir. (Şəkil 2.15)

```
spring:
  cloud:
    vault:
      token: ${vault_token}
      scheme: http
```

Şəkil 2.15. “VaultTemplate” konfiqurasiyası

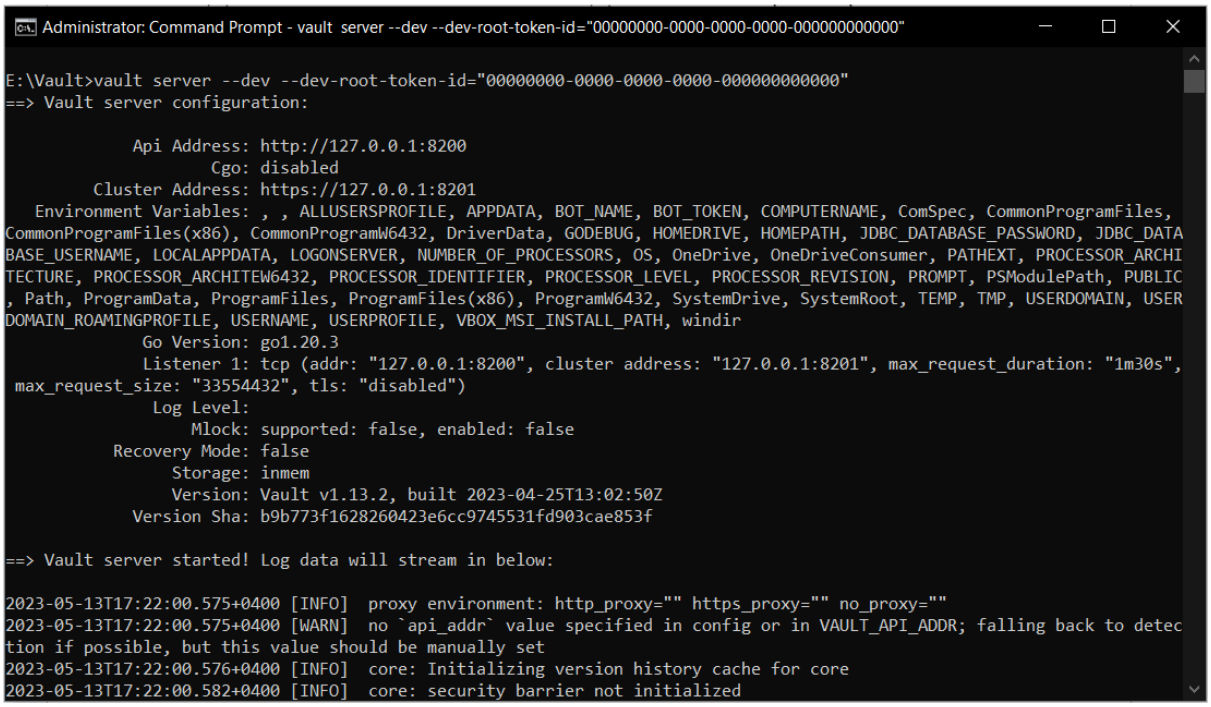
Şəkil 2.15-də konfiqurasiyaya fikir versək görərik ki, burada “${vault_token}” ifadəsini görə bilərik. Proqram təminatı işlədiyi mühitin dəyişənləri arasında “vault_token” dəyişənini axtaracaq və tapıb onu Vault tokeni kimi “VaultTemplate”-ə verəcək. Öz növbəsində də “VaultTemplate” bu tokendən istifadə edərək Vault proqramına qoşulacaq və oradakı icazə verilmiş sirləri oxuya biləcək.

İndi isə “Vault” proqramının işə salınmasına baxaq. Test məqsədli istifadə üçün “Vault” proqramı “dev” modunda işlədiləcək (Şəkil 2.16). İstehsal mühitidən istifadə üçün “dev” modu tövsiyə edilmir.



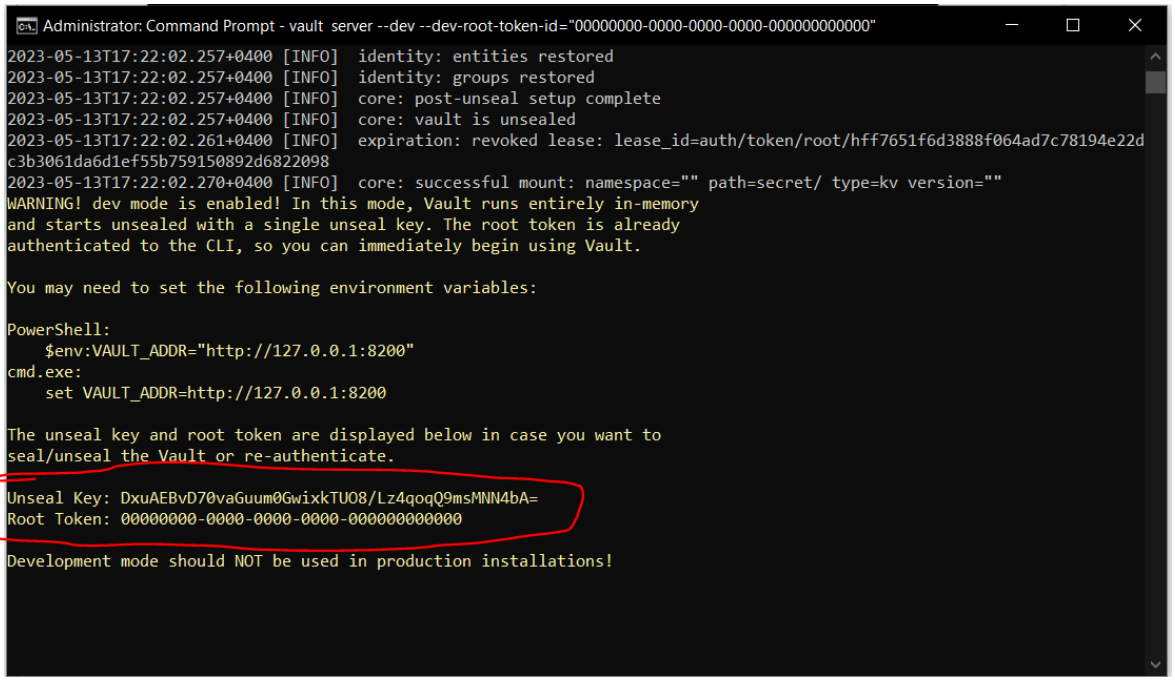
Şəkil 2.16. Vault serverin “dev” modunda işə salınması

Hashicorp Vault proqramından istifadə etmək üçün “Vault server” işə salınmalıdır. Şəkil 2.16-də “Vault server”-in işə salınması üçün lazım olan əmr göstərilmişdir. Əmrdə “Vault server”-in “dev” modda və “root-token-id”-nin dəyərinin “00000000-0000-0000-0000-000000000000” olması göstərilmişdir. Əmri icra etdikdə serverin işə düşdüyünü göstərən loqlar görünür. (Şəkil 2.17)



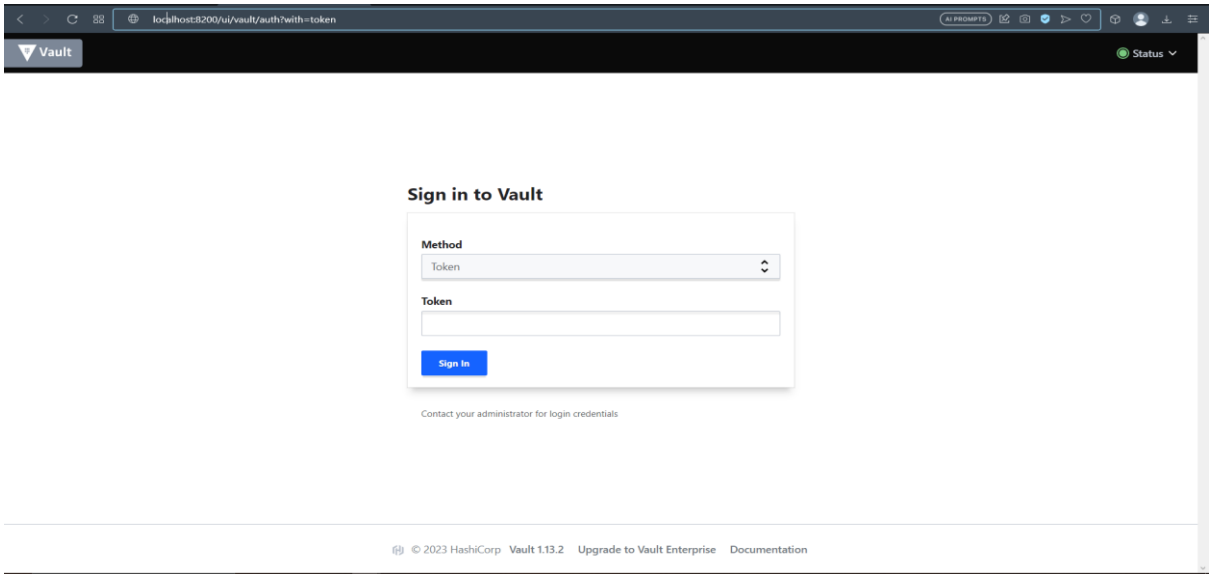
Şəkil 2.17. “Vault server” işə salındı

Qarşılaşdığımız loqlara diqqətlə baxsaq orada bizim əmrdə verdiyimiz token də vardır. (Şəkil 2.17)



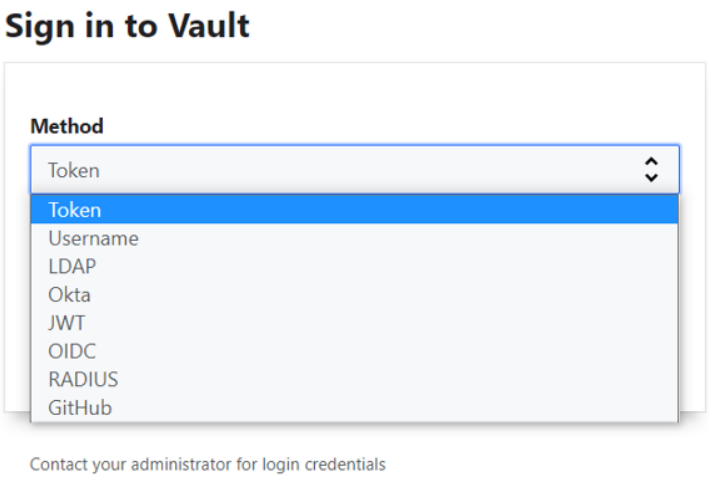
Şəkil 2.18. Loqda “Root Token” və “Unseal key” işarələnmişdir

“Root token” proqram təminatı tərəfindən vault proqramına qoşulmaq üçün istifadə olunur. Həmçinin eyni token Vault proqram təminatının istifadəçi interfeysinə daxil olmaq üçün də istifadə olunur. Vault proqramının istifadəçi interfeysinə daxil olmaq üçün “https://localhost:8200/ui” veb ünvanını brovserə daxil etmək lazımdır. Bu zaman Şəkil 2.19-dakı kimi bir səhifə ilə qarşılaşacağıq.



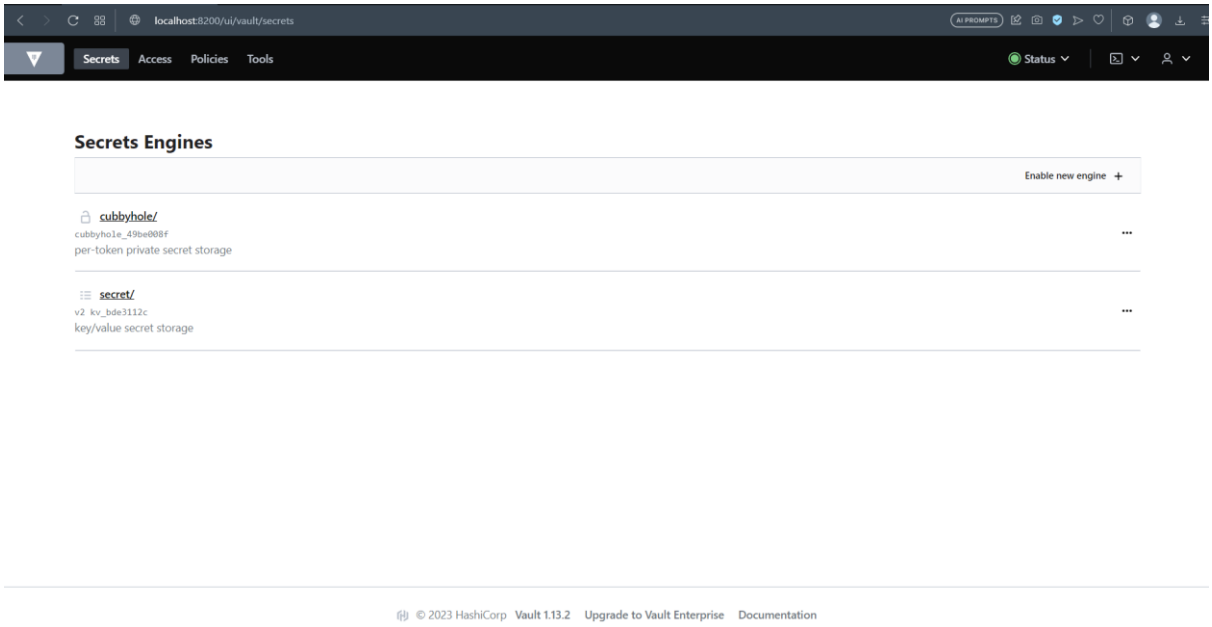
Şəkil 2.19. Vault proqramının istifadəçi interfeysinin giriş səhifəsi

Giriş səhifəsində biz sistemə daxil olmaq üçün təklifə olunan yollardan (Şəkil 2.20) birini istifadə edə bilərik.



Şəkil 2.20. Vault proqramına daxil olmaq üçün təklif olunan autentifikasiya metodları

Hazırda Vault-a daxil olmaq üçün biz “Token” autentifikasiya metodunu istifadə edəcəyik. Loqda olan “Root token”-i daxil edirik və sistemə daxil ola bilirik. İlk olaraq Vault istifadəçi interfeysinin əsas səhifəsi açılır. (Şəkil 2.21)



Şəkil 2.21. Vault istifadəçi interfeysinin əsas səhifəsi

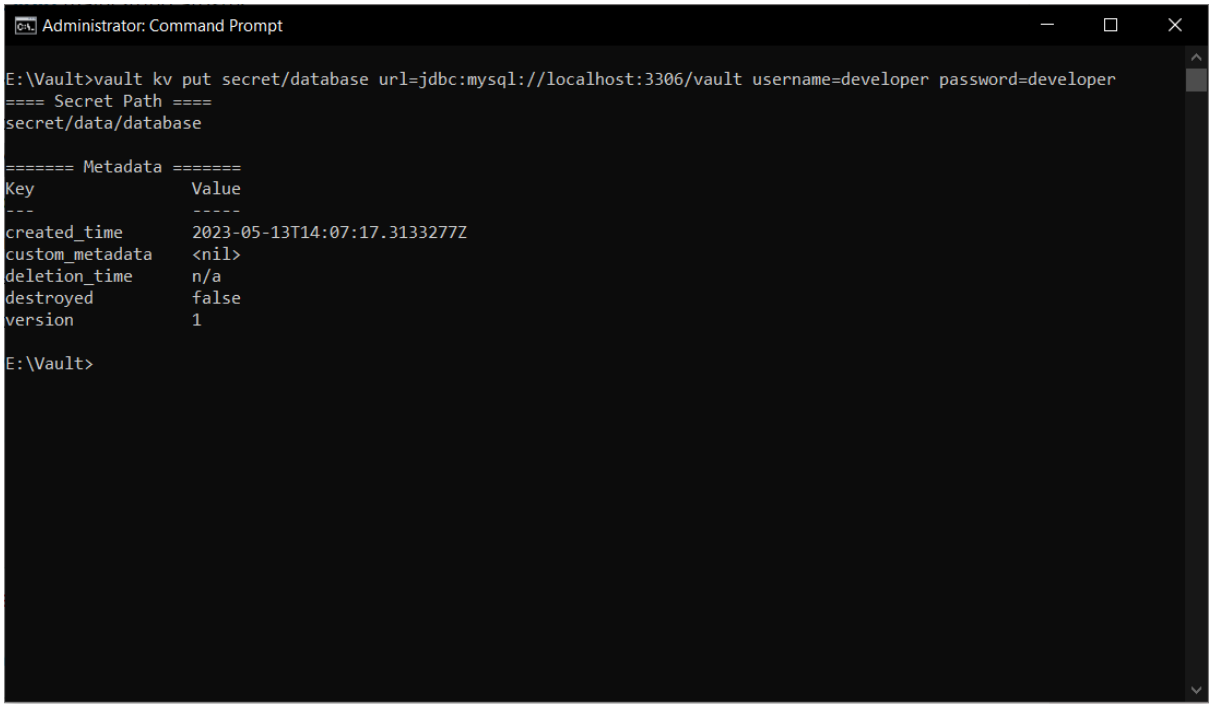
Verilənlər bazasının əlaqə məlumatlarını Vault-a əlavə edək. Bunun üçün biz əmr interfeysini istifadə edəcəyik. İlk olaraq bəzi dəyərləri muhitə əlavə edirik. (Şəkil 2.22)

```
E:\Vault>set VAULT_TOKEN=00000000-0000-0000-0000-000000000000

E:\Vault>set VAULT_ADDR=http://127.0.0.1:8200
```

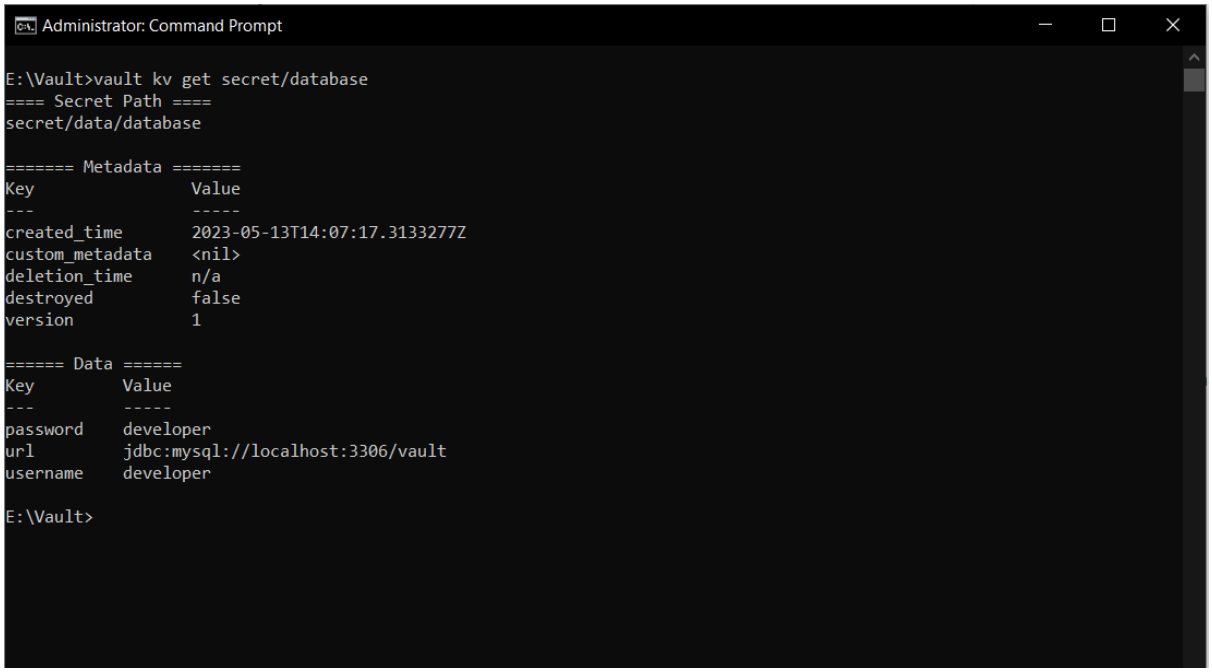
Şəkil 2.22. “VAULT_TOKEN” və “VAULT_ADDR” dəyişənlərinə dəyər verilir

Mühit dəyişənlərini əlavə etdikdən sonra verilənlər bazasına qoşulmaq üçün lazım olan sirləri Vault-a əlavə edə bilərik. (Şəkil 2.23)



Şəkil 2.23. Sirlər Vault-a əlavə edilir

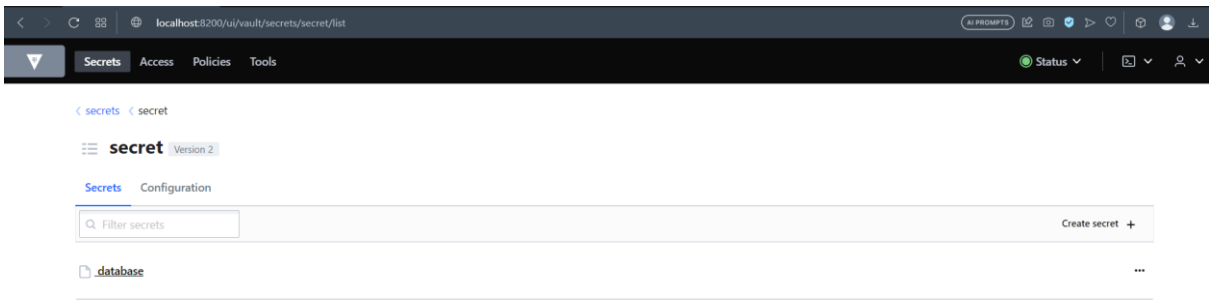
Şəkil 2.23-da göründüyü kimi biz sirləri Vault-a əlavə etmək üçün "vault kv put" əmrindən istifadə etdik və əmr nəticəsində sirlər "secret/database" yoluna yazıldı. Vault həmin bu sirləri şifrələyərək özündə saxlayır. Sirlərin Vault-a əlavə olunduğundan əmin olmaq üçün əmr interfeysində vault əmrlərindən istifadə edərək sirləri görə bilərik. (Şəkil 2.24)



Şəkil 2.24. “secret/database” yolundakı sirləri əldə etmək

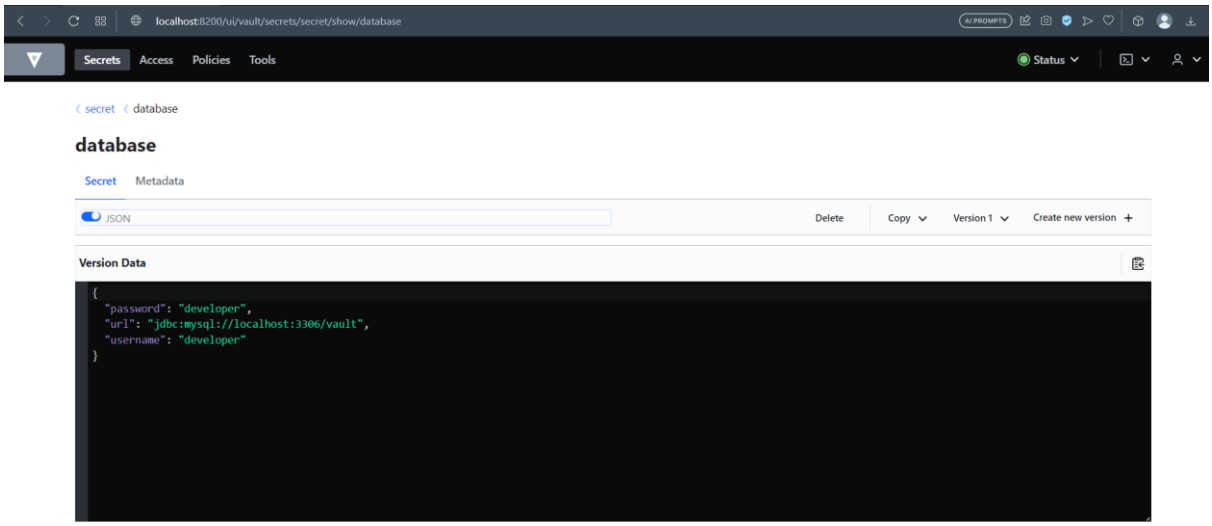
Şəkil 2.24-də istifadə edilmiş “vault kv get secret/database” əmri vasitəsi ilə biz şəkil 2.23-da əlavə etdiyimiz sirləri görə bilirik. Hər bir sirr “secret/database” yolunda açar-dəyər cütündə göstərilmişdir.

Əgər Vault istifadəçi interfeysinə nəzər yetirsək orada da “database” yolunun yarandığını görə bilərik. Bunun üçün Vault istifadəçi interfeysinin əsas səhifəsindəki “Secrets” bölməsinə daxil olmaq lazımdır. Açılan səhifədə “database” yolunu asanlıqla görürük. (Şəkil 2.25)



Şəkil 2.25. “Database” yolu Vault istifadəçi interfeysində görünür

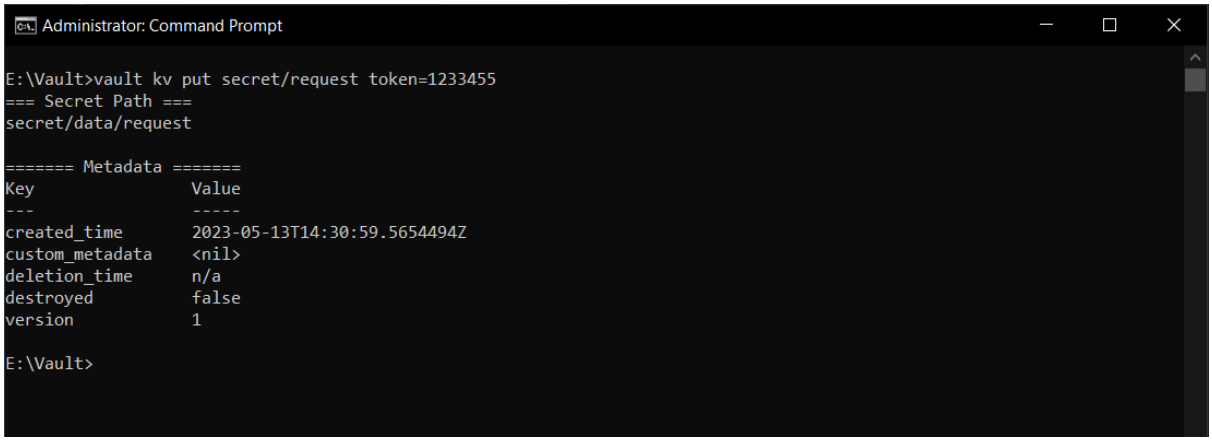
İstifadəçi interfeysində “database” yoluna daxil olsaq, burada əlavə etdiyimiz sirləri rahatlıqla görə bilərik. (Şəkil 2.26)



Şəkil 2.26. “Database” yoluna yazılmış sirlər

Əgər biz yenidən şəkil 20-ə nəzər salsaq, burada verilənlər bazası məlumatlarının “database” yolundan götürüldüyünü görə bilərik.

Verilənlər bazası məlumatlarını sirr kimi Vault-a əlavə etdiyimiz kimi, proqram təminatının ehtiyac duyduğu “request” tokenini də Vault-a əlavə etməliyik. (Şəkil 2.27)



Şəkil 2.27. “secret/request” yoluna “token” sirri əlavə edilir

Proqram təminatının Vault istifadə edə bilməsi üçün sirr mühərrikini işə salmaq lazımdır. (Şəkil 2.28)

Şəkil 2.28. Sirr mühərrikini işə salmaq

Proqram təminatı mənbə kodlarına geri qayıtsaq, müştərilərin məhsulları idarə etməsi üçün proqram təminatı onlara API təqdim edir. Bu API kodlarını proqram təminatının mənbə kodlarındakı “ProductManagementController” sinfində tapa bilərik. (Şəkil 2.29)

```java
package az.edu.aztu.secretmanagement.controller;

import ...

@RestController
public class ProductManagementController {

    private ProductManagementService productManagementService;
    private PaymentService paymentService;

    public ProductManagementController(ProductManagementService productManagementService,
                                       PaymentService paymentService) {
        this.productManagementService = productManagementService;
        this.paymentService = paymentService;
    }

    @GetMapping("/products")
    public ResponseEntity<List<Product>> getProductList() {...}

    @PostMapping("/pay")
    public ResponseEntity pay(@RequestBody PayRequest payRequest) {...}
}
```

Şəkil 2.29. "ProductManagementController" sinfi

Şəkil 2.29-da göstərilmiş mənbə koduna baxsaq burada 2 ədəd API olduğunu görə bilərik:

1. /products

   Http "Get" metodur. Sorğu gəldiyi zaman sorğunu göndərən müştəriyə bütün məhsulların siyahısını göndərir. (Şəkil 2.30)

```java
    @GetMapping("/products")
    public ResponseEntity<List<Product>> getProductList() {
        List<Product> productList = productManagementService.getProductList();
        return ResponseEntity.ok(productList);
    }
```

Şəkil 2.30. "/products" API

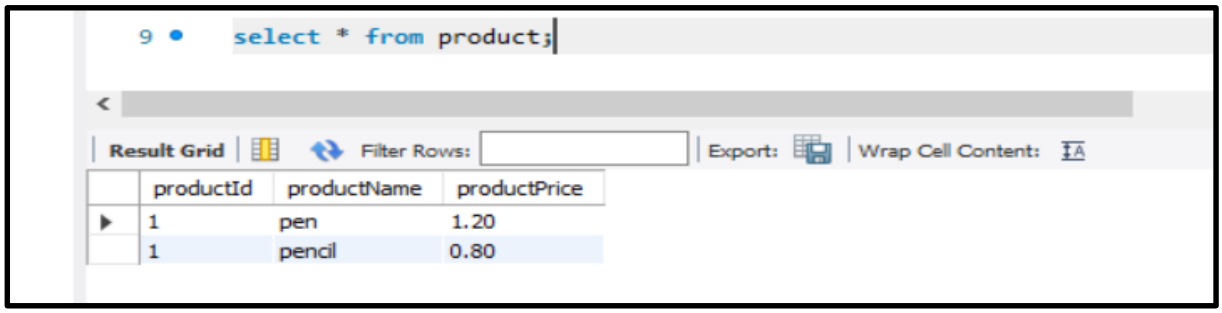
2. /pay

   Http "Post" metodur. Sorğu gəldiyi zaman göndərən sorğu göndərən müştərinin istədiyi məhsul üçün ödənişi keçirir, əgər ödəniş uğurludursa, məhsulu müştərinin hesabına əlavə edir. (Şəkil 2.31)

```java
@PostMapping("/pay")
public ResponseEntity pay(@RequestBody PayRequest payRequest) {
    Product product = productManagementService.getProduct(payRequest.getProductId());
    BigDecimal amount = product.getPrice().multiply(BigDecimal.valueOf(payRequest.getCount()));
    boolean result = paymentService.pay(amount);
    if (result) {
        productManagementService.saveProductToCustomer(payRequest.getProductId(), payRequest.getCustomerId(), payRequest.getCount());
        return ResponseEntity.ok(true);
    }
    else {
        return ResponseEntity.ok(false);
    }
}
```

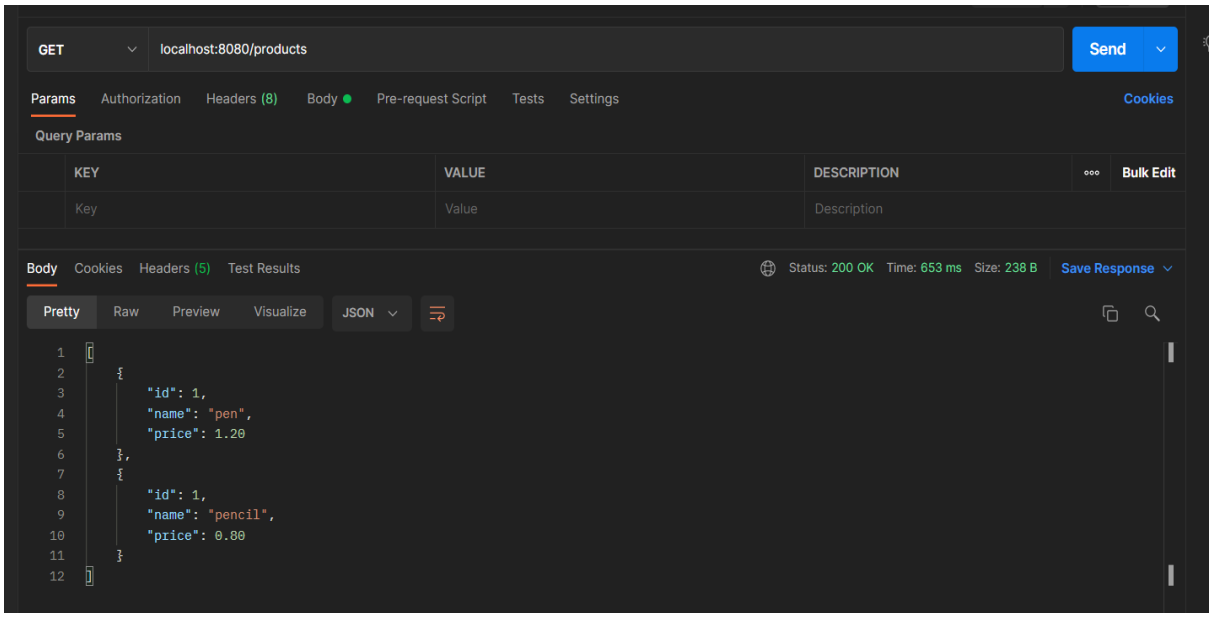
Şəkil 2.31. "/pay" API

Test məqsədi ilə verilənlər bazasına bir neçə məhsul əlavə edilmişdir. (Şəkil 2.32)



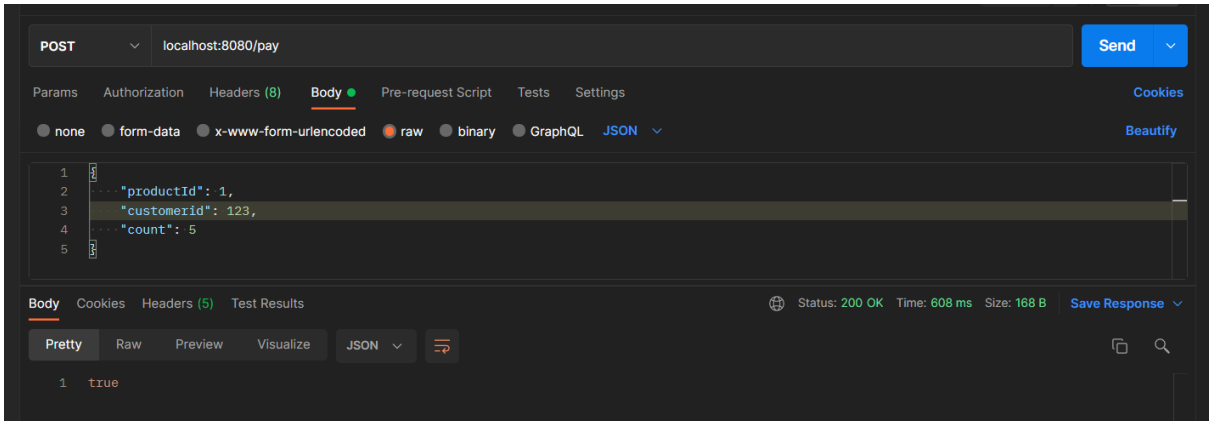
Şəkil 2.32. "Product" cədvəlinə əlavə edilmiş məlumatlar

"/products" API-nı istifadə edərək proqram təminatına sorğu göndərərsək, verilənlər bazasındakı məlumatların API-ın cavabında qayıtdığına şahid ola bilərik. (Şəkil 2.33)



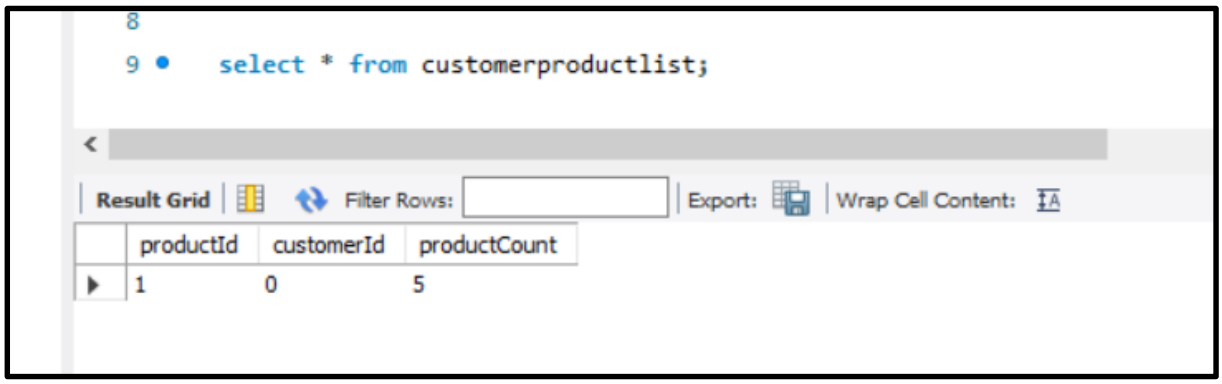
Şəkil 2.33. "/products" API-a sorğu atılaraq test edilməsi

Şəkil 2.33-da gördüyümüz kimi sorğunun nəticəsində geri qayıdan məhsullar verilənlər bazasında olan məhsullardır. Biz bu məhsullardan hər hansısa birini müştəriyə əlavə etmək üçün "/pay" API-dən istifadə etməliyik. Əgər "/pay" API-a atılan sorğu uğurlu olarsa, o zaman verilənlər bazasında "customerproductlist" cədvəlində bu məlumatları görə bilərik.



Şəkil 2.34. "/pay" API-nin test edilməsi

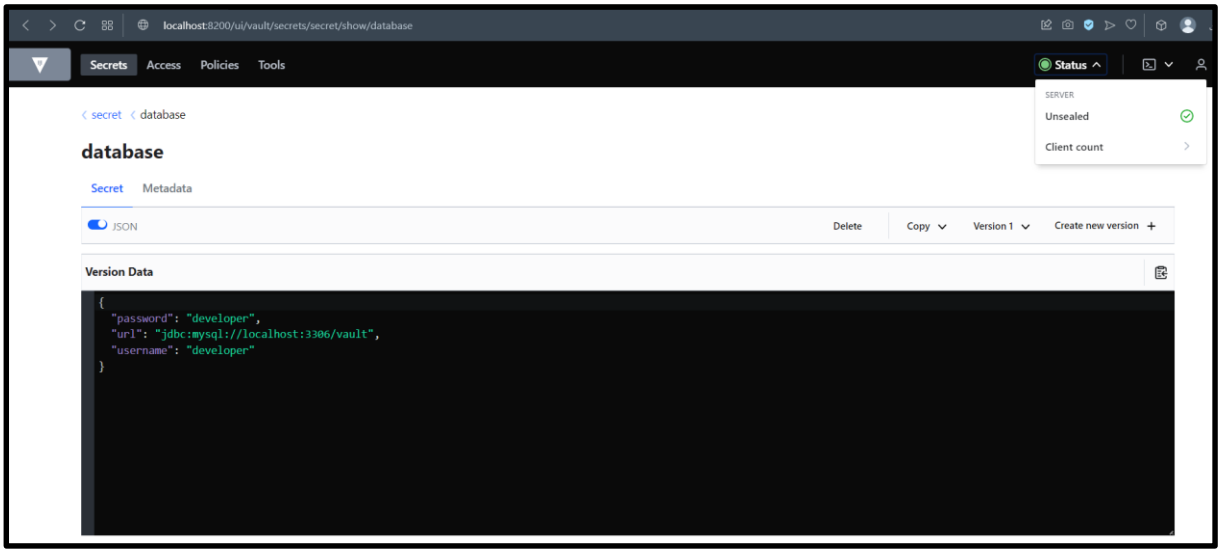
Şəkil 2.34-da göründüyü kimi "/pay" API-si test edilmişdir. Sorğu məzmunu olaraq 123 nömrəli müştərinin hesabına 1 nömrəli məhsuldan 5 ədəd əlavə olunması tələb edilir. Sorğu nəticəsi uğurludur. Nəticənin uğurlu olmasını verilənlər bazasından da yoxlaya bilərik. (Şəkil 2.35)



Şəkil 2.35. Müştərinin tələb etdiyi məhsul müştərinin hesabına əlavə edilmişdir.

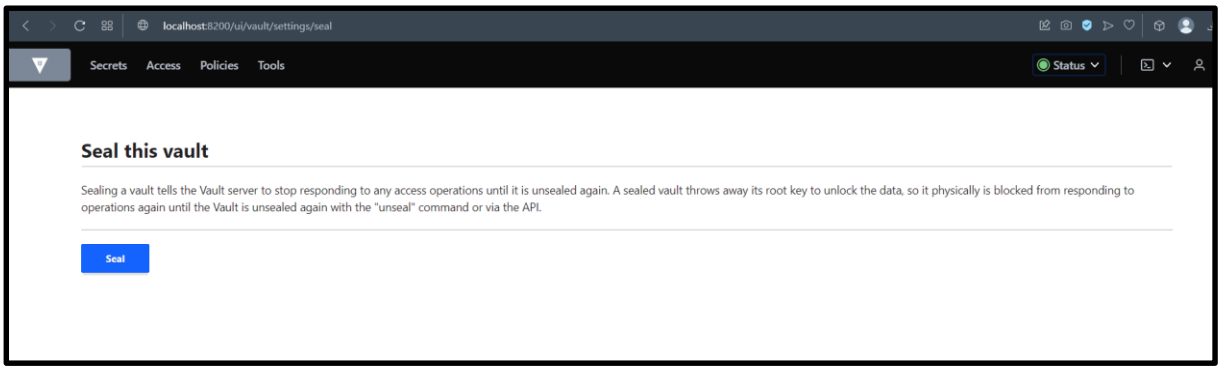
Vault proqramının funksionallıqlarından biri də saxlanılan sirlərin möhürlənməsidir (sealing). Möhürləndikdən sonra sirlərin digərləri tərəfindən oxunması qeyri-mümkün olur. Möhürlənmə bilərəkdən və ya yenidən başlama zamanı baş verir. Bu Vault-da sirlərin təhlükəsizliyini artırır.

Vaultda sirləri istifadəçi interfeysi və həmçinin əmr interfeysi ilə möhürləmək mümkündür.



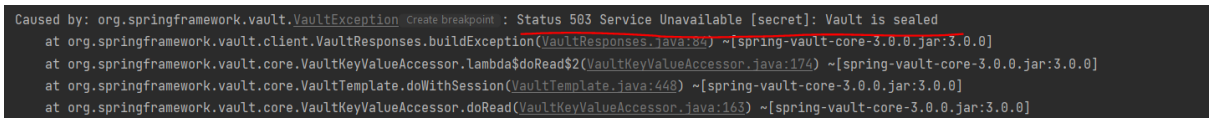
Şəkil 2.36. Vault istifadəçi interfeysində status menyusu

Şəkil 2.36-da Vault istifadəçi interfeysi göstərilmişdir. Səhifədə sağda yuxarıda status menyusu mövcuddur. Menyudakı “Unsealed” seçiminin üzərinə vurmaqla biz yeni səhifəyə keçid edirik. Bu səhifədə bizdən məlumatları möhürləməklə bağlı sual soruşur. (Şəkil 2.37)



Şəkil 2.37. Möhürləmə sorğusu olan səhifə

Şəkil 2.37-dəki “Seal” düyməsini basmaqla biz sirləri möhürləyirik. Möhürü sirlərin üzərindən götürmək üçün biz əvvəlcədən müəyyən edilmiş sayda “unseal” açarlarını daxil etməliyik.



Şəkil 2.38. Möhürlənmiş məlumatı oxumaq istədikdə proqramın atdığı xəta

Məlumatları möhürlədikdən sonra sirlər proqram təminatı tərəfindən oxunmaz hala düşdü. Proqram təminatı sirləri oxumaq üçün Vault proqramına müraciət etdikdə Şəkil 2.38-dəki xətaları atır. Bu xətanın səbəbi odur ki, proqram təminatı sirləri götürə bilmir və bu səbəbdən verilənlər bazası ilə əlaqə qura bilmir.

Nəticə

Bu dissertasiya işində mikroservis arxitekturası, mikroservis arxitekturasında nəzərə alınamalı olan təhlükəsizlik məsələləri və problemləri, həmin problemlərə müasir həllər araşdırıldı. Mikroservis arxitekturası təqdim etdiyi üstünlüklərə görə müasir proqram təminatı mühəndisliyində mühüm yer tutur. Sistemimizi mikroservis arxitekturasına nəzərən dizayn etdikdə, sistemimizin təhlükəsiz olması üçün müəyyən məsələləri nəzərə almalıyıq:

- Ən az imtiyaz
- Avtomatlaşdırma
- Versiyaların yenilənməsi
- Yedəkləmələr
- Etimadnamələr
- Tranzitdə data
- Durğun vəziyyətdə data
- Sirlərin saxlanılması

Dissertasiyanın məzmununda bu məsələlərin hər birinə geniş bir şəkildə toxunulub və bu məsələlərlə bağlı yaranacaq problemlər və onların həlləri ətraflı izah olunub. Sirlərin saxlanılması mövzusuna da ətraflı toxunulmuşdur. Mövcud sirlərin idarəedilməsi texnologiyaları araşdırılmışdır.

- Mərkəzləşdirilmiş sirlərin idarəedilməsi sistemləri
- Sirlər servis kimi
- Konteyner əsaslı sirlərin idarəedilməsi sistemi
- Sirr olmayan arxitektura

Bu texnologiyaların hər biri haqqında dissertasiya işində məlumat verilmiş və mərkəzləşdirilmiş sirlərin idarəedilməsi sisteminə nümunə kimi “Hashicorp Vault” proqramından istifadə edərək proqram təminatı yaradılmışdır. “Vault” proqram təminatını mərkəzləşdirilmiş sirlərin idarəedilməsi sistemidir. Dissertasiya işində Vault proqram təminatının verdiyi üstünlüklər həm nəzəri, həm də praktiki olaraq göstərilmişdir:

- Mərkəzləşdirilmiş sirlərin idarəedilməsi sistemi
- Təhlükəsiz saxlama və şifrələmə
- Dinamik sirlərin təminatı
- Ətraflı giriş nəzarəti
- Sirlərin yenilənməsi və versiyalanması
- Audit qabiliyyəti və uyğunluq
- Şəxsiyyət provayderləri ilə inteqrasiya
- Yüksək əlçatımlılıq və miqyaslanma qabiliyyəti

“Vault”-un yuxarıda göstərilmiş imkanları onun mikroservis arxitekturasında sirlərin idarəedilməsi üçün səmərəli və təhlükəsiz vasitə olduğunu göstərir.

İSTİFADƏ OLUNMUŞ ƏDƏBİYYATIN SİYAHISI

1. https://www.verizon.com/business/resources/reports/dbir/2020/introduction/
2. Hunt T., “Passwords Evolved: Authentication Guidance for the Modern Era”, 2017
3. https://blog.gitguardian.com/the-state-of-secrets-sprawl-2022
4. Meli M., McNiece M. R., and Reaves B., “How bad can it git? characterizing secret leakage in public github repositories.” in NDSS, 2019.
5. https://www.techtarget.com/searchsecurity/news/252500361/Popular-mobile-apps-leaking-AWS-keys-exposing-user-data
6. “Building Microservices: Designing Fine-Grained Systems”, Newman S., O'Reilly Media, Inc., 2015, 615 səhifə
7. "Microservice Security in Action", Siriwardena P. 2020, 616 səhifə
8. "Microservices for the Enterprise: Designing, Developing, and Deploying". Kasun İ. Siriwardena P., 2018, 441 səhifə
9. "API Security in Action", Neil M. 2020, 576 səhifə
10. https://developer.hashicorp.com/vault/docs, Vault dokumentasiya
11. “Kubernetes Secrets Management”, Alex S.B., Andrew B., 2023, 449 səhifə
12. “Introduction to Current Trade Secret Management Research”, Haakon T. L., 2016, SSRN Electronic Journal
13. “Secure Cloud Key Management based on Robust Secret Sharing”, Ahmed B., “10th International Conference on Cryptography and Information Security”, 2021
14. “12 Practices for Secret Management in Infrastructure as Code”, Akond R., Farhat L., Patrick M., 2021 IEEE Secure Development Conference (SecDev), 2021
15. “Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith”, Newman S. 2019, 255 səhifə
16. https://spring.io/guides/gs/accessing-vault/, “Spring framework” dokumentasiya

Xülasə

Bu tezis mikroservis arxitekturasında təhlükəsizliyin artırılması ilə bağlı tədqiqatı təqdim edir. Tədqiqat mikroservis arxitekturasında nəzərə alınmalı təhlükəsizlik məsələlərini, yarana biləcək problemləri və onları həll etmək üçün müasir həlləri tədqiq edir. “Hashicorp Vault” proqram təminatının üstünlükləri, Java proqramlaşdırma dilində yazılmış proqram təminatı ilə inteqrasiyası dissertasiya işində ətraflı göstərilmişdir. Dissertasiyanın 1-ci fəslində mikroservis arxitekturasının nə üçün bugünkü proqram təminatı mühəndisliyinin əhəmiyyətli olduğu araşdırılır. 2-ci fəsildə isə mikroservis arxitekturasındakı təhlükəsizlik məsələləri, əsasən də sirləri idarəedilməsi məsələləri araşdırılmış və mərkəzləşdirilmiş sirlərin idarəetmə sistemi olan “Hashicorp Vault” proqram təminatının üstünlükləri və inkişaf etdirilən proqram təminatlarına inteqrasiyası göstərilmişdir. Bu dissertasiyada təqdim olunan tapıntılar, araşdırmalar daha təhlükəsiz paylanmış sistemlərin qurulmasına kömək edir.

SUMMARY

This thesis presents research on security enhancement in microservices architecture. The study explores the security issues to be considered in microservices architecture, the problems that may arise, and modern solutions to solve them. Advantages of "Hashicorp Vault" software, integration with software written in Java programming language are detailed in the thesis. Chapter 1 of the thesis examines why microservices architecture is important to software engineering today. In the 2nd chapter, the security issues in the microservice architecture, mainly the issues of secret management, were investigated and the advantages of the centralized secret management system "Hashicorp Vault" software and its integration into the developed software were shown. The findings and research presented in this dissertation contribute to building more secure distributed systems.

РЕЗЮМЕ

В этой диссертации представлены исследования по повышению безопасности в архитектуре микросервисов. В исследовании рассматриваются вопросы безопасности, которые необходимо учитывать в архитектуре микросервисов, проблемы, которые могут возникнуть, и современные решения для их решения. В диссертации подробно изложены преимущества программного обеспечения "Hashicorp Vault", интеграция с программным обеспечением, написанным на языке программирования Java. В главе 1 диссертации рассматривается, почему архитектура микросервисов важна для разработки программного обеспечения сегодня. Во 2-й главе были исследованы вопросы безопасности в микросервисной архитектуре, в основном вопросы управления секретами и показаны преимущества программного обеспечения централизованной системы управления секретами «Hashicorp Vault» и его интеграции в разрабатываемое программное обеспечение. Выводы и исследования, представленные в этой диссертации, способствуют созданию более безопасных распределенных систем.