Checkmarx

ENDPOINT LABS

GÜVENLİ YAZILIM GELİŞTİRME

Zafiyetler, Riskler, Önlemler ve Kod Örnekleri

E.CÜNEYT KALPAKOĞLU

İÇİNDEKİLER

ÖNSÖZ ve TEŞEKKÜR

Yıllar önce 1976'larda IBM Hamburg'da ilk kez profesyonel olarak başlayan mesleki kariyerimde mensubu olmaktan onur duyduğum Bilgi Teknolojileri dünyasında ülkemizde pek çok ilklere imza atma gururunu yaşadım. Özellikle Siber Güvenlik ve Uygulama Güvenliği alanında Türkçe Literatür eksikliği beni bu dalda bir kitap hazırlama hazırlığına itti. Bu kitapçık, konusunda bir ilk olarak tamamı 220 sayfayı bulan ve sürekli güncellenen büyük bir araştırmanın kısaltılmışı olarak takdim edilmektedir.

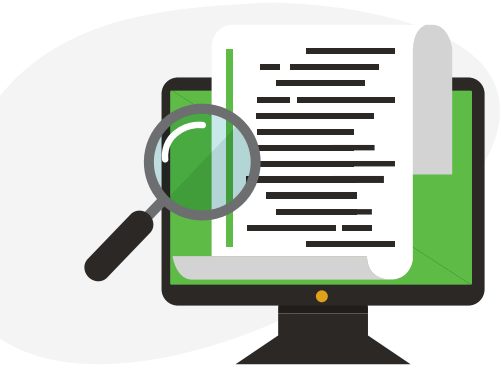
Kitapçığın hazırlanmasında yaptığı araştırmalarla çok değerli katkı sağlayan Merve KOÇ'a ve Endpoint Ekip arkadaşlarıma, bu araştırmanın kitaplaştırılmasını teşvik eden ve destekleyen Checkmarx yöneticileri Miri TAMİR ve Zack BENTOLILA'ya gönülden teşekkürlerimi sunarım.

Checkmarx

Software = **Security**

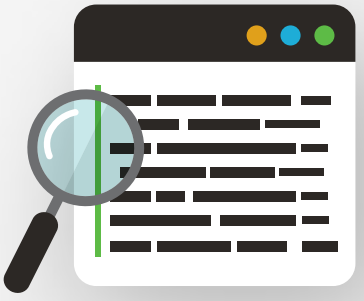
KOD ANALİZİ KAVRAMI

Zaman zaman “Statik kod analizi” kavramının kod kalitesini saptama olarak algılandığını ve kalite ölçmek için olan çözümlerle karıştırıldığını görüyoruz. Statik kod analizi, kodunuzun belli bir kaliteyi tutturması nı sağlamak için değil bu konu için geliştirilmiş özel yazılım ortamları tarafından taranarak Siber Güvenlik kriterlerine göre kritik zafiyetler içerip içermediğinin belirlenmesini amaçlar.



Kod incelenmesi (Code Review) ise programı geliştiren kişiden başka bir programcının kodu gözden geçirdiği manuel bir işlemdir. Kodlama kurallarından kaynaklanan küçük hataları veya sapmaları tespit etmenin ve göreceli olarak genel kod kalitesinin iyileştirmesini sağlamanın geleneksel yolu olmakla birlikte, kritik zafiyetlerin tespitini yapamaz ve başarısı kodu gözden geçiren kişinin dikkati ve birikimine bağlıdır.

Geliştirmekte olduğunuz Kodu test etmenin tüm yazılım geliştirme yaşam döngüsünün en önemli parçalarından biri olduğunu belirtelim. Bu açıdan, yazılımınızı geliştirirken kod güvenliğini sağlamak amacıyla kodunuzu test etmek için kullandığımız günümüzdeki en geçerli yöntem olan statik kod analizi (SAST Static Application Security Test) artık kritik önem arz ediyor. Yeni başlayan bir yazılımcıysanız veya bilginizi geliştirmeyi hedefliyorsanız, bu terimlere yabancı olabilirsiniz. Bu kitapçıkta statik analizin ana temellerine dikkat çekiyoruz. Bu yolla günümüzün modern kod inceleme araçlarının neden **“Siber Güvenliğin”** çok önemli ve kritik bir parçası olduğunu anlayacağız.



Statik ve Dinamik Kod Analizi nedir?

Kod incelemesini tartışırken, kodu incelemenin bilinen iki yolu arasında ayrım yapmak gerekiyor. Bu bağlamda hem statik hem de dinamik kod analizini anlamak zorunlu oluyor

Statik Analiz:

Yazılımcıların kodlarını derlenmemiş (uncompiled) ve yürütmeden (execute etmeden) öncesinde test etme yöntemine statik kod analizi diyoruz. Buna kodun **“çalışma öncesi ortamı”** diyebiliriz. Statik kod analiz araçları, programlama hatalarını ve zafiyetlerini göstermek ve bunların yazılımcılarca bulunması için olağanüstü verimli çözümler sunuyor. Bu şekilde, hatalar ve zafiyetler, kod derlendikten sonra bir sunucuda **“canlı”** olarak çalışmaya başlaması öncesinde hiçbir zarar vermeden önceden rahatlıkla tespit ediliyor.

Günümüzde Statik analiz genellikle kodu analiz etmenin çok kapsamlı ve etkin bir yolu olarak kabul edilir. Aynı zamanda daha ekonomik bir seçenektir. Kod hatalarını ve zafiyetleri erken aşamalarda belirlemek, uzun sürecek bir yazılım geliştirme sürecinde ciddi tasarruf sağlar. Kaynak kodu analizi tamamlandığında, yazılım kullanıcıya tüm güvenlik kusurlarını, kod ihlallerini ve diğer metrikleri detaylarıyla yakalanır ve kapsamlı geri bildirimler sağlar.

Dinamik Analiz:

Kodun geliştirilmesi tamamlandıktan sonra yapılan teste olan “Dinamik kod analizi” diyoruz. Kodun veri tabanları ile ilişkisi, sunucular ve hizmetlerle etkileşimini incelediğinden özellikle bu tip hatalar veya güvenlik açıklarını yakalamak için etkilidir. Ancak, dinamik analiz bazı önemli soru işaretlerini de barındırır. Örneğin, tüm kod tabanını incelemez, yalnızca “on an” yürütülmekte olan kodun çalışmasındaki hataları bulabilir.

Öte yandan, statik analiz ile bulunmayan bazı hatalar da dinamik bir testte, özellikle de kaynak kodun dışındaki servislere dayanan bölümlerle ilgili muhtemel oluşan hatalar yakalanabilecektir. Siber güvenlik açısından doğru olan, mümkün olan en yüksek düzeyde test kapsamı elde etmek için bu iki yöntemin de birlikte kullanılmasıdır. Kısaca gerçek anlamda Uygulama Güvenliği her iki test yöntemiyle birlikte sağlanacaktır.

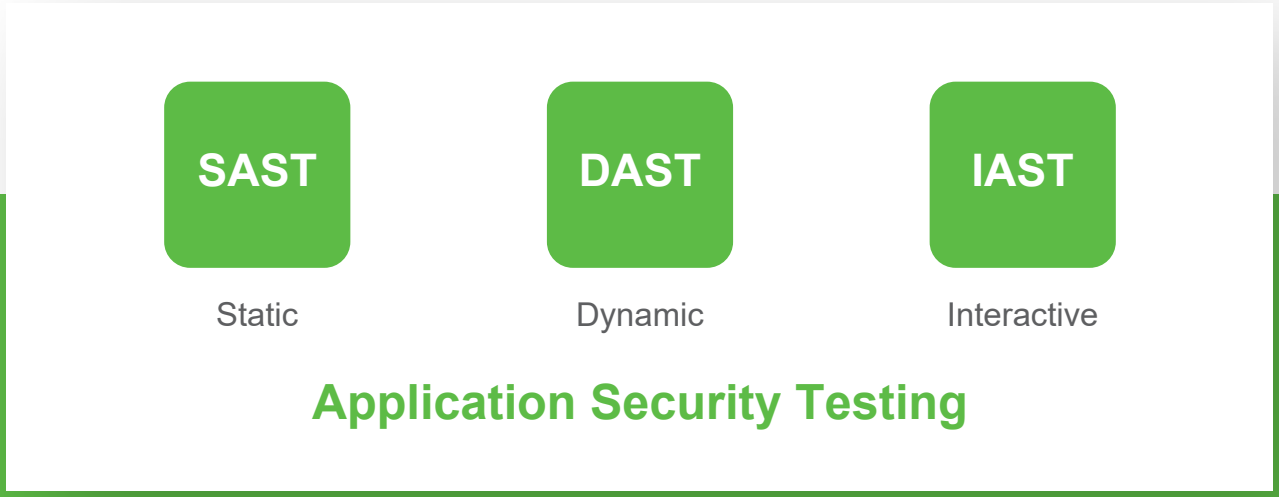
Statik Kod Analizi ile Kod Güvenliğini otomatikleştirme ve Orkestrasyon:

Uygulama açıkları nedeniyle meydana gelen sık ve büyük çaplı ihlaller göz önünde bulundurulduğunda, güvenlik testlerine yönelik geleneksel yaklaşımlar eksik kalıyor.

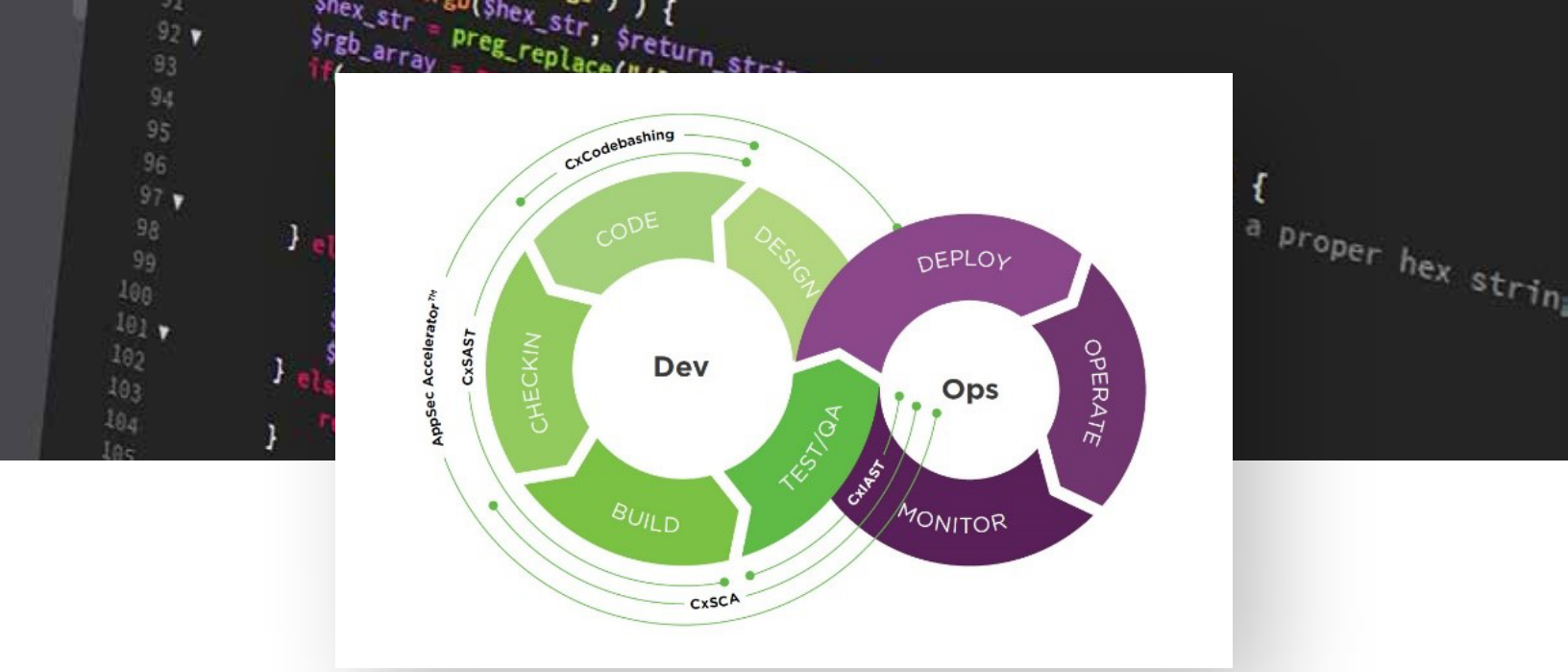
Kod ve uygulama güvenliği testlerinin yazılım geliştirme yaşam döngüsüne sorunsuz bir şekilde gömülmesi artık zorunlu hale geldi. DevSecOps'un arkasında, güvenlik görünürlüğünü ve güvenc-esini kod kontrolünden kod üretimine ve devam eden operasyonlara entegre eden ve otoma-tikleştiren **Zafiyet yönetimi ve uygulama yazılımı orkestrasyonu** *(Vulnerability Management & Apllica-tion Security Test Orchestration)* önemli bir kavram olarak ortaya çıktı.

Gartner 2019'lardan başlayan süreçte bu konuya dikkat çekerek "Başarılı DevSecOps için Gerekli olan 10 yaklaşım" raporunda "2019'dan itibaren kurumsal DevSecOps girişimlerinin %70'inden fazlasının otomatik güvenlik açığı ve yapılandırma taramasına ve bu çözümlere ihtiyaç duyacağını ve bu oranın sürekli arttığını dile getiriyor.

Artık güvenlik ve DevOps araçlarını yazılım geliştirme yaşam döngüsü boyunca entegre ederek tüm kod depoları, uygulama dağıtımları ve DAST, SAST ve IAST altyapıları arasında sürekli görünürlük ve izlemeyle kod ve uygulama güvenliği testlerini otomatikleştirip düzenlemeye olan gereksinim artıyor. Uygulama yazılımı güvenliğinde DevOps süreçlerinin üretkenliğini sağlayan SAST, DAST ve IAST ürünleriyle %100 entegre olarak çalışan ve süreci tamamlayıcı ortamlar **Yazılım Güvenliğinde** çığır açıyor. Genelde Statik kod analizi manuel olarak yapılabilse de, kod satır sayısı çok büyük projelerin kodlarıyla uğraşmak bazen çok uzun zaman alabilir. Bu süreci otomatikleştirmek verimliliği çok arttırır.



Otomatik analiz yöntemi kodunuzu sürekli olarak hatalara karşı kontrol etmenin çok daha hızlı ve kolay bir yoludur. Bu yöntemle güvenlik ve **DevOps** araçlarını yazılım geliştirme yaşam döngüsü boyunca entegre ederek tüm kod depoları, uygulama dağıtımları ve **DAST** ve **SAST** altyapıları arasında sürekli görünürlük ve izlemeyle kod ve uygulama güvenliği testlerini otomatikleştirip düzenlenerek, uygulama yazılımı güvenliğinde **DevOps** süreçlerinin üretkenliği sağlanır.



İlaveten **Orkestrasyon**, güvenlik açıklarından sorumlu kodu geliştirenin kim olduğunu saptamak, **SAST** araçları tarafından keşfedilen güvenlik açıklarından sorumlu olan kodu geliştireni otomatik olarak belirlemek, güvenlik açığını düzeltmek için doğru kişiyi bulmak için harcanan zamanı ortadan kaldırır.

Güvenlik açıklarını ilk etapta oluşturan geliştiricilere otomatik olarak sorunların atanması, düzeltme sürecini hızlandırır. Geliştirici düzeyindeki güvenlik açığı verileri, güvenli kodlamayı geliştiriciler için ölçülebilir bir **KPI** haline getirir ve güvenli kodlamayı öncelikler merdiveninden yukarı taşımaya yardımcı olur.

Yerel ortamlardaki güvenlik açıklarına yönelik görünürlük, geliştiriciler arasında gelişmiş güvenlik bilincine yol açar. Her geliştirici ve ekip tarafından oluşturulan güvenlik açıklarına göre özel olarak tasarlanmış eğitim programları, eğitimin yatırım getirisini artırır. İyileştirme veritabanı, geliştiricilerin şirkette biriken bilgi birikiminden yararlanmasına ve güvenlik açıklarını daha hızlı düzeltmesine olanak tanır.

Günümüz Trendleri:

Dünya çapında en rekabetçi ve en zorlu sektörler son yıllarda gittikçe artan talep ve nüfus artışları etkisiyle artık manuel süreçler, büyük hacimli verilerin yönetilmesinde yetersiz kalmaktadır. İş akışlarının kolaylaştırması ve mevcut sorunlarla başa çıkmak için benzersiz yenilikçi iş stratejileri geliştirilmesi zorunlu hale geliyor. Bu bağlamda siber güvenlik açısından yeni teknolojileri iyi anlamak ve bundan yararlanmak kritik önem taşıyor. İşletme sahipleri ve yöneticilerine bu konuda yardımcı olabilmeye yönelik olarak 2021 yılı için dikkate alınması gereken başlıca trendlere bakalım.

Günümüzde, yapay zekâ, artırılmış ve sanal gerçeklik, veri analitiği, bulut çözümleri vb. gibi teknolojiler hem girişimcilerin hem de işletmelerin kendi alanlarında yenilikçi dijital yaklaşımlar ile çözümler geliştirmelerine yardımcı olacak. Özellikle pandemi dönemindeki beklenti ve ihtiyaçlar, bu konunun önemini arttırdı. İşte 2021 yılını etkileyebilecekler:

Cybersecurity Ventures'ın 2017 raporunda, fidye yazılımı hasarlarının 2015'te 325 milyon dolar iken 2017'de dünyaya 5 milyar dolara mal olacağı öngörülmüştü; sadece iki yılda 15 kat artış oldu. 2018 için zararın 8 milyar dolar olduğu tahmin edildi ve 2019 için bu tutar 11,5 milyar dolara yükseldi. En son tahmin, küresel Siber Fidye Saldırısı hasar maliyetlerinin 2021 yılında 20 milyar dolara ulaşması yönündedir ki bu sayı 2015'e göre 57 kat daha fazladır. Dikkat çeken diğer konu 2016'da her 40 saniyede bir gerçekleşmekte olan siber fidyecilik saldırılarının, 2021 yılı için her 11 saniyede bir yapılacağının tahmin edildiğidir. Siber tehditlere Karşı savunma amaçlı kullanılan sayısız çözüme rağmen, siber fidye saldırıları hızla artıyor. Uzaktan çalışma patlamasının ardından, saldırganların seçtikleri taktik olarak fidye saldırılarının yükselişinin temelini oluşturan temel zayıflık, siber güvenliğin davranış temelli tehdit algılamaya aşırı güvenmesidir.

Uzaktan ve hibrit çalışma durumlarına sürekli bağlılıkla, "eski normal"in geri dönmesi artık olası değildir. Güvenli Kodlama, yazılım geliştiricilerin güvenlik açıklarına karşı koruma sağlamak ve önlem almak için kodlarına uyguladıkları standarda dayanan bir yönergedir.

Neden Güvenli Kodlama Uygulanmalıdır?

Veri ihlalleri, DOS saldırıları, Hizmet kaybı, gizli bilgilerin ele geçirilmesi ve çok daha fazlası gibi çeşitli güvenlik olayları hakkında sık sık duyumlarımız oluyor. Bunlar, güvenli kodlamanın önemini daha geniş bir şekilde vurgulayan risklerden birkaçıdır.

OWASP (Web Application Security Project) Nedir?

Open Web Application Security Project'in kısaltması olan OWASP web uygulamalarının güvenliği doğrultusunda araştırmalar yapan özgür bir topluluktur. Yapılan araştırma ve çalışmalar, herkese açıktır. OWASP vakfının hiçbir ticari kurum ve şirketle bağı yoktur ve tamamen topluluk amaçları doğrultusunda gönüllü çalışanlarla Ar-Ge çalışmalarını gerçekleştirmektedir.

OWASP (Top 10) Nedir?

OWASP İlk 10, sektör uzmanlarından oluşan bir grup tarafından derlenen, web uygulamaları için en tehlikeli on bilgi güvenliği riskinin sıralamasıdır. Derecelendirmenin her noktası için risk, uzmanlar tarafından **OWASP Risk Derecelendirme Metodolojisine göre** hesaplanır ve Zayıflık Yaygınlığı, Zayıflık Tespit Edilebilirliği ve İstismar edilebilme yanı sıra bunların operasyonlarının veya teknik etkilerinin sonuçlarının kritikliğinin bir değerlendirmesini içerir. OWASP İlk 10 resmi bir standart değildir, yalnızca zayıflıkların ve güvenlik ihlallerinin ciddiyetini ve önemini sınıflandırmak için, düzenlediği güvenlik açığı ödül programları ve bünyesindeki gönüllü siber güvenlik uzmanları tarafından yaygın olarak araştırılan ve sürekliliği olan ciddi bir teknik incelemedir. OWASP İlk 10 projesinin mevcut tüm riskleri kapsayamayacağını ancak derecelendirme yayınlandığı zaman diliminde yalnızca en alakalı ve en öne çıkanları temsil ettiği bilinmelidir.

1

SQL ENJEKSİYONU (SQL INJECTION)

Risk ne olabilir?

Bir saldırgan, sistemin tüm verilerine doğrudan erişebilir. Saldırgan, özel kullanıcı bilgileri, kredi kartı bilgileri, özel iş verileri ve diğer gizli veriler dahil olmak üzere sistem tarafından saklanan tüm hassas bilgileri çalabilir. Aynı şekilde, muhtemelen saldırgan mevcut verileri değiştirebilir, silebilir; hatta yeni sahte veriler ekleyebilir. Bazı durumlarda, veritabanı üzerinde kod çalıştırmak bile mümkün olabilir.

Nasıl olur?

Uygulama, işlenmek üzere veritabanına metinsel bir SQL sorgusu göndererek verileri bir veritabanında depolar ve yönetir. Uygulama sorguyu, güvenilir olmayan verileri gömerek basit string birleştirme yoluyla oluşturur. Bununla birlikte, veri ve kod arasında bir ayrım yoktur; ayrıca, gömülü veriler ne veri tipi geçerliliği açısından kontrol edilir ne de daha sonra temizlenir. Bu nedenle, güvenilir olmayan veriler SQL komutları içerebilir veya hedeflenen sorguyu değiştirebilir. Veritabanı, değiştirilen sorguyu ve komutları uygulamadan geliyormuş gibi yorumlar ve buna göre çalıştırır.

Nasıl Önlenir?

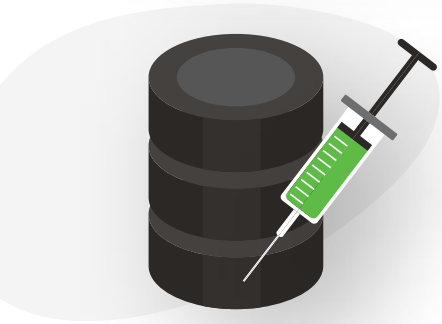
- Kaynaktan bağımsız olarak tüm güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir beyaz listeye (white list) dayanmalıdır: Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
    - Özellikle şunlar kontrol edilmelidir:
        - Veri tipi
        - Boyut
        - Range, sıra
            - Format, şekil
            - Beklenen değerler
- Veritabanı nesnelerine ve İşlevselliğe erişim En Az Ayrıcalık İlkesi'ne göre kısıtlanmalıdır.
- SQL sorguları oluşturmak için dinamik olarak birleştirilmiş string'ler kullanılmamalıdır.
- Tüm veri erişimleri için geçici olan dinamik sorgular yerine veritabanında depolanan prosedürlerin kullanımı tercih edilmelidir.
- Güvenilir olmayan string birleştirme işlemleri yerine parametreli sorgular ve komutlar gibi güvenli veritabanı bileşenleri kullanılmalıdır.
- Alternatif olarak, veritabanına doğrudan dinamik olarak erişmek yerine uygulama için etkinleştirilen izin verilen komutları önceden tanımlamak ve kapsüllemek için bir ORM kitaplığı kullanmak daha da iyi bir çözümdür. Bu şekilde kodlar ve veriler birbirinden izole edilmelidir.
- Veri doğrulama işlemi, OWASP Encoder veya ESAPI kütüphaneleri gibi güvenli kütüphaneler kullanılarak etkin bir şekilde gerçekleştirilebilir.
- Sorguları parametrelendirmek için PreparedStatement kullanmayı veya daha da iyisi CallableStatement kullanımı tercih edilmelidir. Ayrıca string birleştirmek yerine .set*() metotlarıyla dinamik veriler eklenebilir.
- Hibernate, myBatis veya diğerleri gibi bir ORM paketi kullanması önerilir.

Kod Örneği

Java String birleştirme kullanarak SQL sorgusu oluşturulması

```
public int getGuvensiz_KullaniciID(HttpServletRequest istek) throws ServletException,
IOException {
        int kullaniciID = 0;
        String kullaniciAdi = istek.getParameter("KullaniciAdi");
        String sql = "SELECT [KullaniciID] FROM [Kullanicilar] WHERE [KullaniciAdi] = '" +
kullaniciAdi + "' " ;
```

```
try {
            Connection conn = getConnection();
            Statement stmt = conn.createStatement();
            ResultSet data = stmt.executeQuery(sql);
            kullaniciID = data.getInt(1);
        } catch (SQLException ex) {
            handleExceptions(ex);    }
        finally {
            closeQuietly(data);
            closeQuietly(stmt);
            closeQuietly(conn);   }
        return kullaniciID;   }
```



2

KÖR SQL ENJEKSİYONU (BLIND SQL INJECTION)

Risk Ne olabilir?

Bir saldırgan, sistemin tüm verilerine doğrudan erişebilir. Saldırgan, özel kullanıcı bilgileri, kredi kartı bilgileri, özel iş verileri ve diğer gizli veriler dahil olmak üzere sistem tarafından saklanan tüm hassas bilgileri çalabilir. Aynı şekilde, muhtemelen saldırgan mevcut verileri değiştirebilir veya silebilir, hatta yeni sahte veriler ekleyebilir. Bazı durumlarda, veritabanı üzerinde kod çalıştırmak bile mümkün olabilir. Gizli bilgileri doğrudan ifşa etmeye veya değiştirmeye ek olarak, bu güvenlik açığı, kimlik doğrulamasını atlamak, güvenlik kontrollerini alt üst etmek veya bir veri izini taklit etmek için de kullanılabilir. Kötüye kullanım ihtimali arttıkça bu güvenlik açığının saldırganlar tarafından bulunması ve bundan yararlanması daha kolay olur. Bu durumda, gerçek istismar bir seferde tek bitlik bilgi ile sınırlandırılırken, bu işlem daha fazla zaman alıcı ve oldukça rahatsız edici olmasına rağmen, sistemden tüm verileri almak hala mümkündür. Veri değiştirme ve oluşturma, kod yürütme vb. gibi diğer sonuçların etkilenmediği ve yine de eşit derecede yararlanılabileceği unutulmamalıdır.

Nasıl olur?

Uygulama, işlenmek üzere veritabanına metinsel bir SQL sorgusu göndererek verileri bir veritabanında depolar ve yönetir. Uygulama sorguyu, güvenilir olmayan verileri gömerek basit string birleştirme yoluyla oluşturur. Bununla birlikte, veri ve kod arasında bir ayrım yoktur; ayrıca, gömülü veriler ne veri tipi geçerliliği açısından kontrol edilir ne de daha sonra temizlenir. Bu nedenle, güvenilir olmayan veriler SQL komutları içerebilir veya hedeflenen sorguyu değiştirebilir. Veritabanı, değiştirilen sorguyu ve komutları uygulamadan geliyormuş gibi yorumlar ve buna göre çalıştırır.

Bu durumda, saldırganın veritabanından veri döndüren uygulamaya güvenmesine gerek kalmaz. Bunun yerine, sunucu durumunu belirtmek için yalnızca uygulama hatalarının varlığına dayanarak, değişen kullanıcı girdisine dayalı olarak bir dizi boolean testi gerçekleştiren mevcut araçlardan yararlanmak mümkündür. Böylece, tam veritabanı içeriği aşamalı olarak, her seferinde bir bit olarak elde edilebilir

Nasıl önlenir?

Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:

Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.

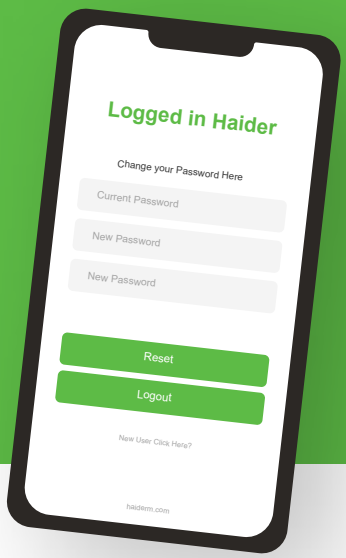
- Özellikle şunlar kontrol edilmelidir:
  - Veri tipi
  - Boyut
  - Range, sıra
  - Beklenen değerler
- Veritabanı nesnelerine ve işlevselliğe erişim En Az Ayrıcalık İlkesi'ne göre kısıtlanmalıdır.
- SQL sorguları oluşturmak için dinamik olarak birleştirilmiş string'ler kullanılmamalıdır.
- Tüm veri erişimleri için geçici olan dinamik sorgular yerine veritabanında depolanan prosedürlerin kullanımı tercih edilmelidir.
  - Güvenilir olmayan string birleştirme işlemleri yerine parametreli sorgular ve komutlar gibi güvenli veritabanı bileşenleri kullanılmalıdır.
  - Alternatif olarak, veritabanına doğrudan dinamik olarak erişmek yerine uygulama için etkinleştirilen izin verilen komutları önceden tanımlamak ve kapsüllemek için bir ORM kitaplığı kullanmak daha da iyi bir çözümdür. Bu şekilde kodlar ve veriler birbirinden izole edilmelidir.
  - Kullanıcının sorgulanan tablonu dinamik olarak isim vermesine izin verilmemelidir. Ayrıca mümkünse tablo isimlerini dinamik olarak belirtmekten tamamen kaçınılmalıdır.
  - Hatalar, sunucu durumu hakkında bilgi sızdırmadan veya herhangi bir hata oluştuğundan tüm istisnaların düzgün bir şekilde ele alındığından emin olunmalıdır.
  - Veri doğrulama işlemi, OWASP Encoder veya ESAPI kütüphaneleri gibi güvenli kütüphaneler kullanılarak etkin bir şekilde gerçekleştirilebilir.

- Sorguları parametrelendirmek için **PreparedStatement** kullanmayı veya daha da iyisi **CallableStatement** kullanımı tercih edilmelidir. Ayrıca string birleştirmek yerine **.set***() metotlarıyla dinamik veriler eklenebilir.
- Hibernate, myBatis veya diğerleri gibi bir ORM paketi kullanılması önerilir.

Kod Örneği

Java String birleştirme kullanılarak SQL INSERT sorgusu oluşturulması

```
public String güvensizKullanici_Kabulu(HttpServletRequest istek) throws ServletException, IOException {
        int satirSayisi = 0;
        String kullaniciAdi = istek.getParameter("KullaniciAdi");
        String sql = "INSERT INTO [Kullanici] ([KullaniciAdi], [KabulTarihi]) VALUES ('" + kullaniciAdi + "', CURRENT_TIMESTAMP) " ;
        try {
                Connection conn = getConnection();
                Statement stmt = conn.createStatement();
                satirSayisi = stmt.executeUpdate(sql);
        } catch (SQLException ex) {
                handleExceptions(ex);   }
finally {
                closeQuietly(stmt);
                closeQuietly(conn);   }
if (satirSayisi > 0)
                return "Başarılı";
else
                return "Başarısız!";   }
```



3

SEZGISEL SQL ENJEKSİYONU (HEURISTIC SQL INJECTION)

Risk Ne olabilir?

Bir saldırgan, sistemin tüm verilerine doğrudan erişebilir. Saldırgan, özel kullanıcı bilgileri, kredi kartı bilgileri, özel iş verileri ve diğer gizli veriler dahil olmak üzere sistem tarafından saklanan tüm hassas bilgileri çalabilir. Aynı şekilde, muhtemelen saldırgan mevcut verileri değiştirebilir, silebilir; hatta yeni sahte veriler ekleyebilir. Bazı durumlarda, veritabanı üzerinde kod çalıştırmak bile mümkün olabilir.Gizli bilgileri doğrudan ifşa etmeye veya değiştirmeye ek olarak, bu güvenlik açığı, kimlik doğrulamasını atlamak, güvenlik kontrollerini alt üst etmek veya bir veri izini taklit etmek için de kullanılabilir.

Kötüye kullanım ihtimali arttıkça bu güvenlik açığının saldırganlar tarafından bulunması ve bundan yararlanması daha kolay olur. Bu durumda, enjeksiyonun kendi dahili kontrollerini uygulayan harici bir bileşende görünüp görünmediğine dikkat edilmelidir.

Nasıl Olur?

Uygulama, işlenmek üzere veritabanına bir SQL sorgusu göndererek verileri bir veritabanında depolar ve yönetir. Uygulama sorguyu, güvenilir olmayan verileri gömerek basit string birleştirme yoluyla oluşturur. Bununla birlikte, veri ve kod arasında bir ayrım yoktur; ayrıca, gömülü veriler ne veri tipi geçerliliği açısından kontrol edilir ne de daha sonra temizlenir. Bu nedenle, güvenilir olmayan veriler SQL komutları içerebilir veya hedeflenen sorguyu değiştirebilir. Veritabanı, değiştirilen sorguyu ve komutları uygulamadan geliyormuş gibi yorumlar ve buna göre yürütür.

Görünür veritabanı erişiminin harici bir bileşen veya API içinde kapsüllendiği unutulmamalıdır. Böylelikle saldırgan, kullanıcı girdisini değiştirerek SQL sorgusuna rastgele veriler enjekte edebilir. Bu sorgu muhtemelen daha sonra, veritabanı sunucusuna gönderildiği API veya bileşene iletilir.

Nasıl Önlenir?

Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:

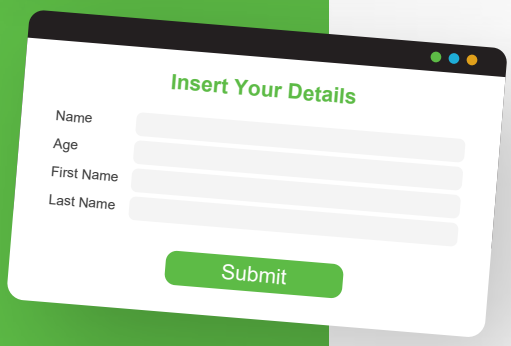
- Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:
  - Veri tipi
  - Boyut
  - Range, sıra
  - Format, şekil
  - Beklenen değerler
- Veritabanı nesnelerine ve işlevselliğe erişim En Az Ayrıcalık İlkesi'ne göre kısıtlanmalıdır.
- SQL sorguları oluşturmak için dinamik olarak birleştirilmiş string'ler kullanılmamalıdır.
- Tüm veri erişimleri için geçici olan dinamik sorgular yerine veritabanında depolanan prosedürlerin kullanımı tercih edilmelidir.
- Güvenilir olmayan string birleştirme işlemleri yerine parametreli sorgular ve komutlar gibi güvenli veritabanı bileşenleri kullanılmalıdır.
- Alternatif olarak, veritabanına doğrudan dinamik olarak erişmek yerine uygulama için etkinleştirilen izin verilen komutları önceden tanımlamak ve kapsüllemek için bir ORM kitaplığı kullanmak daha da iyi bir çözümdür. Bu şekilde kodlar ve veriler birbirinden izole edilmelidir.
- Opak 3. parti sürücüleri kullanmak yerine standart veri erişim kütüphaneleri ve platform API'leri tercih edilmelidir.
- Veri doğrulama işlemi, OWASP Encoder veya ESAPI kütüphaneleri gibi güvenli kütüphaneler kullanılarak etkin bir şekilde gerçekleştirilebilir.
- Sorguları parametrelendirmek için PreparedStatement kullanmayı veya daha da iyisi CallableStatement kullanımı tercih edilmelidir. Ayrıca string birleştirmek yerine .set*() metotlarıyla dinamik veriler eklenebilir.
- Özel bir veri sarıcı yerine Hibernate, myBatis veya diğerleri gibi bir ORM paketi kullanılması önerilir.

Kod Örneği

Java String Birleştirmeyi Kullanarak Pseudo (Sözde) Sorgu Oluşturma

```
public int getGuvensiz_KullaniciID(HttpServletRequest istek) throws ServletException,
IOException {
        int kullaniciID = 0;
        String kullaniciAdi = istek.getParameter("KullaniciAdi");
        String Veri = "SELECT [KullaniciID] FROM [Kullanicilar] WHERE [KullaniciAdi] = '"
+ kullaniciAdi + "' " ;
```

```
try {
                CustomDbDriver db = new CustomDbDriver();
                kullaniciID = db.runQuery(Veri);
        } catch (SQLException ex) {
              handleExceptions(ex);   }
        return kullaniciID;   }
```



4

İKİNCİ SEVİYE SEZGİSEL SQL ENJEKSİYONU (HEURISTIC SECOND ORDER SQL INJECTION)

Risk Ne olabilir?

Bir saldırgan, sistemin tüm verilerine doğrudan erişebilir. Saldırgan, özel kullanıcı bilgileri, kredi kartı bilgileri, özel iş verileri ve diğer gizli veriler dahil olmak üzere sistem tarafından saklanan tüm hassas bilgileri çalabilir. Aynı şekilde, muhtemelen saldırgan mevcut verileri değiştirebilir, silebilir; hatta yeni sahte veriler ekleyebilir. Bazı durumlarda, veritabanı üzerinde kod çalıştırmak bile mümkün olabilir.

Gizli bilgileri doğrudan ifşa etmeye veya değiştirmeye ek olarak, bu güvenlik açığı, kimlik doğrulamasını atlamak, güvenlik kontrollerini alt üst etmek veya bir veri izini taklit etmek için de kullanılabilir. Kötüye kullanım ihtimali arttıkça bu güvenlik açığının saldırganlar tarafından bulunması ve bundan yararlanması daha kolay olur. Bu durumda, enjeksiyonun kendi dahili kontrollerini uygulayan harici bir bileşende görünüp görünmediğine dikkat edilmelidir.

Nasıl Olur?

Uygulama, işlenmek üzere veritabanına bir SQL sorgusu göndererek verileri bir veritabanında depolar ve yönetir. Uygulama sorguyu, güvenilir olmayan verileri gömerek basit string birleştirme yoluyla oluşturur. Bununla birlikte, veri ve kod arasında bir ayrım yoktur; ayrıca, gömülü veriler ne veri tipi geçerliliği açısından kontrol edilir ne de daha sonra temizlenir. Bu nedenle, güvenilir olmayan veriler SQL komutları içerebilir veya hedeflenen sorguyu değiştirebilir. Veritabanı, değiştirilen sorguyu ve komutları uygulamadan geliyormuş gibi yorumlar ve buna göre yürütür.

Görünür veritabanı erişiminin harici bir bileşen veya API içinde kapsüllendiği unutulmamalıdır. Böylelikle saldırgan, kullanıcı girdisini değiştirerek SQL sorgusuna rastgele veriler enjekte edebilir. Bu sorgu muhtemelen daha sonra, veritabanı sunucusuna gönderildiği API veya bileşene iletilir

Nasıl Önlenir?

Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:

- Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:
  - Veri tipi
  - Boyut
  - Range, sıra
  - Format, şekil
  - Beklenen değerler
- Veritabanı nesnelerine ve işlevselliğe erişim En Az Ayrıcalık İlkesi'ne göre kısıtlanmalıdır.
- Veritabanı nesnelerine ve işlevselliğe erişim En Az Ayrıcalık İlkesi'ne göre kısıtlanmalıdır.
- SQL sorguları oluşturmak için dinamik olarak birleştirilmiş string'ler kullanılmamalıdır.
- Tüm veri erişimleri için geçici olan dinamik sorgular yerine veritabanında depolanan prosedürlerin kullanımı tercih edilmelidir.
- Güvenilir olmayan string birleştirme işlemleri yerine parametreli sorgular ve komutlar gibi güvenli veritabanı bileşenleri kullanılmalıdır.
- Alternatif olarak, veritabanına doğrudan dinamik olarak erişmek yerine uygulama için etkinleştirilen izin verilen komutları önceden tanımlamak ve kapsüllemek için bir ORM kitaplığı kullanmak daha da iyi bir çözümdür. Bu şekilde kodlar ve veriler birbirinden izole edilmelidir.
- Opak 3. parti sürücüleri kullanmak yerine standart veri erişim kütüphaneleri ve platform API'leri tercih edilmelidir.
- Veri doğrulama işlemi, OWASP Encoder veya ESAPI kütüphaneleri gibi güvenli kütüphaneler kullanılarak etkin bir şekilde gerçekleştirilebilir.
- Sorguları parametrelendirmek için PreparedStatement kullanmayı veya daha da iyisi CallableStatement kullanımı tercih edilmelidir. Ayrıca string birleştirmek yerine .set*() metotlarıyla dinamik veriler eklenebilir.
- Özel bir veri sarıcı yerine Hibernate, myBatis veya diğerleri gibi bir ORM paketi kullanılması önerilir.

Kod Örneği

Java Rastgele Verilere Dayalı String Birleştirme Kullanarak Pseudo (Sözde) Sorgu Oluşturma

```
public String getGuvensiz_AktifBaglanti(HttpServletRequest istek) throws ServletExcep-
tion, IOException {

        String DBdenKullaniciAdi;
        String baglanti;
        try {
                Connection conn = getConnection();

                CallableStatement readNameStmt =
conn.prepareCall("{call getActiveUser (?)}");

                readNameStmt.registerOutParameter(1,
java.sql.Types.VARCHAR);

                readNameStmt.execute();

                DBdenKullaniciAdi = readNameStmt.getString(1);
                String veri = "SELECT [Baglanti] FROM
[Kullanicilar] WHERE [KullaniciAdi] = '" + DBdenKullaniciAdi + " ' ";

                CustomDbDriver db = new CustomDbDriver();
                baglanti = db.runQuery(Veri);

        } catch (SQLException ex) {
                handleExceptions(ex);    }
        finally {

                closeQuietly(readNameStmt);
                closeQuietly(conn);   }

        return baglanti;   }
```

5

XPATH ENJEKSİYONU
(XPATH INJECTION)

Risk Ne olabilir?

XPath sorgusunu rastgele bir ifadeyle değiştirebilen bir saldırgan, XML belgesindeki hangi düğümlerin seçileceğini ve dolayısıyla uygulamanın hangi verileri işleyeceğini kontrol edebilecektir. Bunun, XML belgesinin türüne ve kullanımına bağlı olarak, gizli bilgilerin alınması, uygulama akışının kontrolü, hassas verilerin değiştirilmesi, dosyaların rastgele okunması ve hatta kimlik doğrulama atlama, kimlik çalıp o kimliğe bürünme ve ayrıcalık yükseltme gibi çeşitli etkileri olabilir.

Nasıl olur?

Uygulama, bir XPath sorgusu kullanarak bir XML belgesini sorgular. Uygulama, potansiyel olarak saldırganın denetimi altında olan güvenilir olmayan veriler dahil olmak üzere string'leri basitçe birleştirerek sorguyu oluşturur. Dış veriler ne veri türü geçerliliği için kontrol edilmediğinden ne de sonradan temizlendiğinden, veriler kötü niyetli olarak uygulamanın XML belgesinden yanlış bilgileri seçmesine neden olacak şekilde işlenebilir.

Nasıl Önlenir?

Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:

- Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:
  - Veri tipi
  - Boyut
  - Range, sıra
  - Format, şekil
  - Beklenen değerler
- XPath sorgusunu dış verilere bağımlı yapmaktan kaçınılmalıdır.
- Güvenilir olmayan verilerin sorguya dahil edilmesi kesinlikle gerekliyse, veriler en azından ilk olarak uygun şekilde doğrulanmalı veya temizlenmelidir.
- Mümkünse, XPath sorgularını harici parametrelerle eşlemek, veri ve kod arasındaki ayrımı korumak tercih edilir.
- Girdi temizleme gerektiğinde OWASP'ın ESAPI kütüphanesi ile Encoder.encodeForXPath() kullanılarak güvenli bir şekilde yapılabilir. Ek kütüphaneler de benzer fonksiyonları sağlayabilir.

Kod Örneği

Java Kullanıcı Girişli XPath İfadesi Kullanılarak XML Düğümü Seçimi

```
public String Oku_GuvensizKullaniciGrubu(HttpServletRequest istek) throws ServletException, IOException {

        String sonucGrup = " ' ";

        String kullaniciID = istek.getParameter("KullaniciID");

        String ifade ="//KULLANİCİLAR/KULLANİCİ[KullaniciID/text() =' " + kullaniciID + " ']/GRUP/text()";

        try {

                DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();

                DocumentBuilder docKullanicilar = builder.parse(new File(KULLANİCİLAR_XML_FILE));

                XPath navigator = XPathFactory.newInstance().newXPath();

                sonucGrup = navigator.evaluate(ifade, docKullanicilar);

        } catch (Exception ex) {

                handleExceptions(ex);    }

        return sonucGrup;   }
```

6 - BAĞLANTI STRING'I ENJEKSIYONU (CONNECTION STRING INJECTION)

Risk Ne olabilir?

Bir saldırgan, uygulamanın veritabanı sunucusuyla olan bağlantı string'ini değiştirebilirse, aşağıdakilerden herhangi birini yapabilir:

- Uygulama performansı hasarı (MIN POOL SIZE arttırarak)
- Ağ bağlantısıyla kurcalama (Örneğin güvenilir bağlantı yoluyla)
- Uygulamayı saldırganın sahte veritabanına yönlendirme
- Veritabanındaki system hesabının parolasının keşfedilmesi (Brute Force "Kaba Kuvvet" saldırısıyla)

Nasıl olur?

Veritabanıyla veya başka bir harici sunucuyla (örneğin, Active Directory) iletişim kurmak için, uygulama dinamik olarak bir bağlantı string'i oluşturur. Bu bağlantı string'i, kötü niyetli bir kullanıcı tarafından kontrol edilebilen, güvenilir olmayan verileri içerir. Sınırlandırılmadığından veya uygun şekilde sterilize edilmediğinden, güvenilir olmayan veriler, bağlantı string'ini kötü niyetle manipüle etmek için kullanılabilir.

Nasıl olur?

- Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:
  Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:

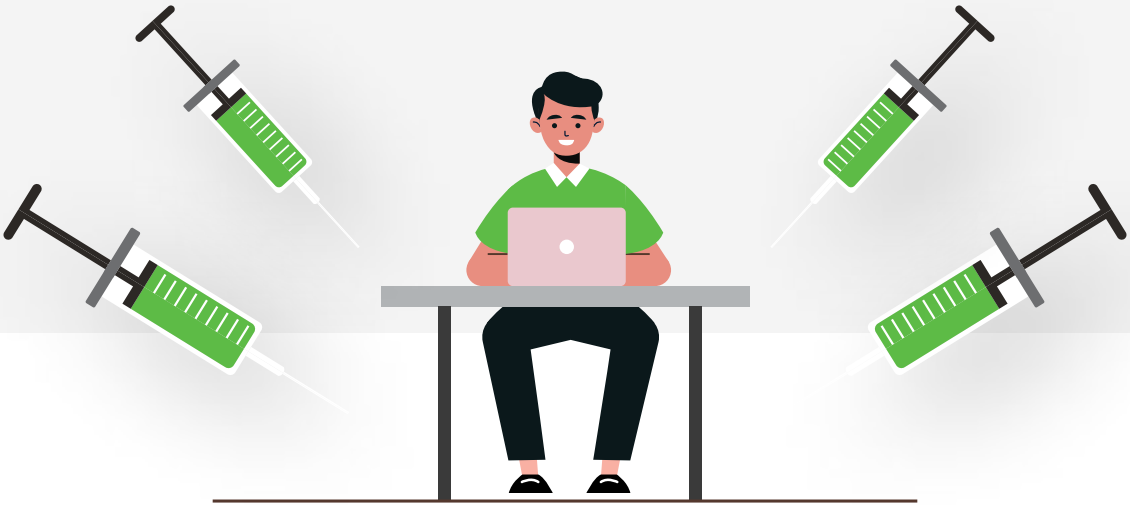
01	02	03	04	05
Veri tipi	Boyut	Range, sıra	Format, şekil	Beklenen değerler

- Kullanıcıların veritabanı bağlantı string'ini kontrol etmesine izin verilmemelidir. Güvenilir olmayan verilere, özellikle kullanıcı girdisine dayalı olup dinamik olarak bağlantı string'leri oluşturmaktan kaçınılmalıdır.
- Tüm bağlantı string'leri uygun yapılandırma mekanizmalarında saklanmalıdır. Runtime'da dinamik olarak bir bağlantı string'i oluşturmak gerekirse, güvenilir olmayan veriler doğrudan bağlantı string'ine dahil edilmemelidir; bunun yerine, kullanıcıların önceden tanımlanmış bağlantı string'leri arasından seçim yapmasına izin verilmelidir.

Kod Örneği

Java Kullanıcı Girdisi Kullanarak Veritabanına Bağlanma

```
private Connection Guvensiz_BaglantıAc(HttpServletRequest istek) throws ServletExcep-
tion, SQLException {
        Connection conn = null;
        String dbSunucu = istek.getParameter("Sunucu");
        String dbIsim = istek.getParameter("Veritabani");
        String kullaniciUrl = String.format(JDBC_URL_TEMPLATE, dbSunucu, dbIsim);
        try {
                conn = DriverManager.getConnection(kullaniciUrl, DB_KULLANICI, DB_SI-
FRE);
        } catch (Exception ex) {
                handleExceptions(ex);   }
        return conn;   }
```



7 - KOMUT ENJEKSİYONU (COMMMAND INJECTION)

Risk Ne olabilir?

Saldırgan, uygulama sunucusu ana bilgisayarında rastgele sistem düzeyinde işletim sistemi komutları çalıştırabilir. Uygulamanın işletim sistemi izinlerine bağlı olarak bunlar şunları içerebilir:

- Dosya eylemleri (okuma, yazma, oluşturma, değiştirme, silme)
- Saldırganın sunucusuna bir ağ bağlantısı açma
- Sistem servislerini başlatma veya durdurma
- Çalışan uygulamayı değiştirme
- Tam sunucu devralma

Nasıl olur?

Uygulama, görevini tamamlamak için uygulama kodu yerine işletim sistemi düzeyinde bir komut çalıştırır. Komut, bir saldırgan tarafından kontrol edilebilen güvenilir olmayan verileri içerir. Bu güvenilir olmayan string, bir saldırgan tarafından tasarlanan ve saldırgan doğrudan uygulama sunucusunda komutlar çalıştırıyormuş gibi çalıştırılabilen kötü amaçlı sistem düzeyinde komutlar içerebilir.
Bu durumda, uygulama kullanıcı girişinden veri alır ve bunu bir string olarak işletim sistemine iletir. Bu doğrulanmamış veriler daha sonra, uygulama ile aynı sistem ayrıcalıklarıyla çalışan bir sistem komutu olarak işletim sistemi tarafından yürütülür.

Nasıl önlenir?

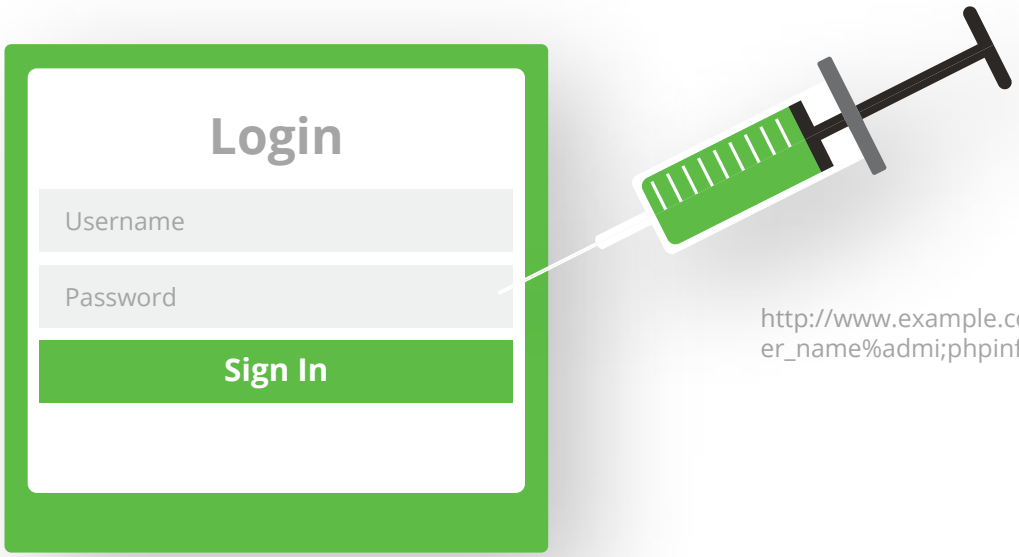
- Herhangi bir doğrudan Shell komutu yürütülmesini önlemek için kod yeniden düzenlenmelidir. Bunun yerine, platform tarafından sağlanan API'ler veya kütüphaneler kullanılır.
- Komut yürütmesini kaldırmak imkansızdır. Ancak dinamik ve kullanıcı kontrollü verileri içermeyen statik komutlar yürütülmelidir.
- Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:
  Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:

01	02	03	04	05
Veri tipi	Boyut	Range, sıra	Format, şekil	Beklenen değerler

- Derinlemesine bir savunma önlemi olarak hasarı en aza indirmek için, uygulama gereksiz işletim sistemi ayrıcalıklarına sahip olamayan, kısıtlı bir kullanıcı hesabı kullanarak çalışacak şekilde yapılandırılmalıdır.
- Mümkünse En Az Ayrıcalık İlkesi'ne göre, yalnızca uygulama tarafından kullanılan belirli komutlar ve dosyalar için minimum ayrıcalıklara sahip ayrı bir kullanıcı hesabı kullanmak için tüm işletim sistemi komutları izole edilmelidir.
- Bir sistem komutunu çağırmak veya kullanıcı girdisiyle harici bir programı yürütmek için kesinlikle gerekliyse, ilk parametre olarak sabit kodlanmış bir sistem komutu veya yürütülebilir dosya kullanarak ProcessBuilder sınıfının yalnızca güvenli biçimi kullanılmalıdır.
- Kullanıcı kontrollü girişle bash, cmd veya make gibi herhangi bir shell veya komut yorumlayıcısı doğrudan çalıştırılmamalıdır.

Java Kullanıcı Girdisi Kullanarak Veritabanına Bağlanma

```
private String Guvensiz_SistemKomutuCalistir(HttpServletRequest istek) throws Servlet-
Exception, IOException {
        String komutSonucu = "";
        String kullaniciKomutu = istek.getParameter("Komut");
        try {
                Runtime runtime = Runtime.getRuntime();
                Process altPros = runtime.exec("/bin/sh -c \"" + PROGRAM_ISIM + " " +
kullaniciKomutu+"\"");
                BufferedReader irProcOutput = new BufferedReader(new
        InputStreamReader(altPros.getInputStream()));
        String satir = null;
        while ((satir = irProcOutput.readLine()) != null)
                komutSonucu += satir;
        irProcOutput.close();
        } catch (Exception ex) {
                handleExceptions(ex);   }
        return komutSonucu;   }
```



8 - KOD ENJEKSIYONU (CODE INJECTION)

Risk Ne olabilir?

Saldırgan, uygulama sunucusu ana bilgisayarında rastgele sistem düzeyinde işletim sistemi komutları çalıştırabilir. Uygulamanın işletim sistemi izinlerine bağlı olarak bunlar şunları içerebilir:

- Hassas verileri okuma veya değiştirme gibi veritabanı erişimi
- Dosya eylemleri (okuma, yazma, oluşturma, değiştirme, silme)
- Dosya eylemleri (okuma, yazma, oluşturma, değiştirme, silme)
- Web sitesini değiştirme
- Uygulama şifreleme anahtarlarını kullanarak gizli verilerin şifre çözümü
- Sistem servislerini başlatma veya durdurma
- Tam sunucu devralma

Nasıl olur?

Uygulama, kötü niyetli bir kullanıcının kontrolü altında olan, güvenilir olmayan verileri içeren kod oluşturup çalıştırarak bazı eylemler gerçekleştirir. Veriler kötü amaçlı kod içeriyorsa, yürütülen kod, sanki saldırgan kodu doğrudan uygulama sunucusunda çalıştırıyormuş gibi, bir saldırgan tarafından tasarlanan sistem düzeyinde etkinlikleri içerebilir.

Nasıl önlenir?

Saldırgan, uygulama sunucusu ana bilgisayarında rastgele sistem düzeyinde işletim sistemi komutları çalıştırabilir. Uygulamanın işletim sistemi izinlerine bağlı olarak bunlar şunları içerebilir:

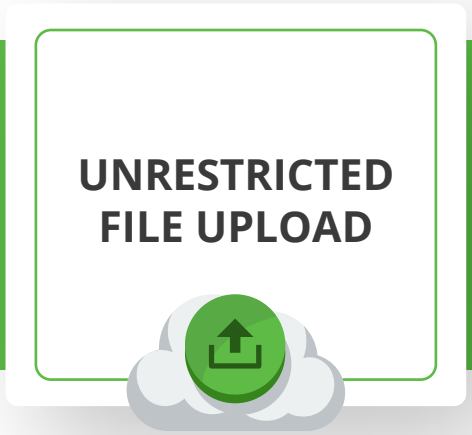
- Uygulama, kullanıcı girişi, yüklenen dosyalar veya bir veritabanı dahil olmak üzere herhangi bir harici kaynaktan güvenilir olmayan herhangi bir kodu derlememeli, yürütmemeli veya değer lendirmemelidir.
- Dinamik yürütmeye harici verilerin dahil edilmesi kesinlikle gerekliyse, verilerin koda parametre olarak aktarılmasına izin verilir, ancak kullanıcı verileri doğrudan yürütülmemelidir.
- Güvenilir olmayan verileri dinamik yürütmeye geçirmek gerekirse, çok katı veri doğrulaması zorunlu kılınmalıdır. Örneğin, yalnızca belirli değerler arasındaki tam sayılar kabul edilmelidir.
- Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır: Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:

- Mümkünse bir Black List ile karşılaştırmak yerine her zaman bilinen ve güvenilir girdileri White List'e almak tercih edilmelidir.
- Uygulama, gereksiz ayrıcalıkları olmayan kısıtlı bir kullanıcı hesabı kullanarak çalışacak şekilde yapılandırılmalıdır.
- Mümkünse En Az Ayrıcalık İlkesi'ne göre, yalnızca uygulama tarafından kullanılan belirli komutlar ve dosyalar için minimum ayrıcalıklara sahip ayrı bir kullanıcı hesabı kullanmak için tüm işletim sistemi komutları izole edilmelidir.
- Dinamik yürütme gerekliyse, tüm dinamik kod harici bir process'te çalıştırılmalı ve harici veriler ProcessBuilder kullanarak, process'e bir parametre olarak iletilmelidir.
- Alternatif olarak, kısıtlayıcı bir politika uygulamak için özel bir SecurityManager uygulayarak ve yalnızca önceden tanımlanmış korumalı alanlardan izin verilen sınıfların yüklenmesine izin vermek için özel bir ClassLoader uygulayarak ayrı bir iş parçacığı üzerinde çalışan yalıtılmış bir sanal alanda kodun dinamik olarak yürütülmesi mümkündür. (Bunun sanal alan istismarlarına karşı hiçbir zaman tam koruma sağlayamayacağı ve mümkünse tüm dinamik yürütmelerden kaçınılması gerektiği unutulmamalıdır.)

Kod Örneği

Java Kullanıcı Girdisi Kullanarak Veritabanına Bağlanma

```
private String Guvensiz_SistemKomutuCalistir(HttpServletRequest istek) throws Servlet-

Exception, IOException {

String komutSonucu = "";

String kullaniciKomutu = istek.getParameter("Komut");

try {

Runtime runtime = Runtime.getRuntime();

Process altPros = runtime.exec("/bin/sh -c \"" + PROGRAM_ISIM + " " + kullaniciKomu-

tu+"\"");

BufferedReader irProcOutput = new BufferedReader(new

InputStreamReader(altPros.getInputStream()));

String satir = null;

while ((satir = irProcOutput.readLine()) != null)

komutSonucu += satir;

irProcOutput.close();

} catch (Exception ex) {

handleExceptions(ex);   }

return komutSonucu;   }
```



9- KISITLANMAMIŞ DOSYA YÜKLEME (UNRESTRICTED FILE UPLOAD)

Risk Ne olabilir?

Kullanıcıların sınırsız boyutta dosyaları kaydetmesine izin vermek, saldırganların dosya depolamasını gereksiz olarak doldurmasına veya kaydetme işlemini yürüten sistemleri zorlayacak uzun yazma işlemleri gerçekleştirmesine olanak sağlayabilir. Bu depolama alanını tüketmek veya kullanılamayacağı noktaya sınırlamak, hizmet reddine (DoS – Denial of Service) neden olacaktır.

Nasıl olur?

Uygulama kodu, kullanıcılar tarafından yüklenen dosyaları depoya kaydetmeden önce dosya boyutunu doğrulamaz ve potansiyel olarak her boyuttaki dosyanın yüklenmesine izin verir.

Nasıl önlenir?

Saldırganların boyut kontrolleri gerçekleştirerek rastgele boyuttaki dosyaları yüklemesini önlemek için kodda amaçlanan dosya boyutu sınırlandırılmalıdır. İstemci tarafı boyut kontrollerine veya kullanıcılar tarafından sağlanan herhangi bir boyut parametresine güvenilmemelidir; bunun yerine sunucudaki dosyanın boyutu değerlendirilmelidir.

Kod Örneği

Java

```
public void saveMultipartFile(CommonsMultipartFile multipartFile, String path) throws
IOException {
        FileOutputStream fos = new FileOutputStream(path);
fos.write(multipartFile.getBytes());
fos.close();   }
```

10 - GÜNLÜK IŞLEME (LOG FORGING)

Risk Ne olabilir?

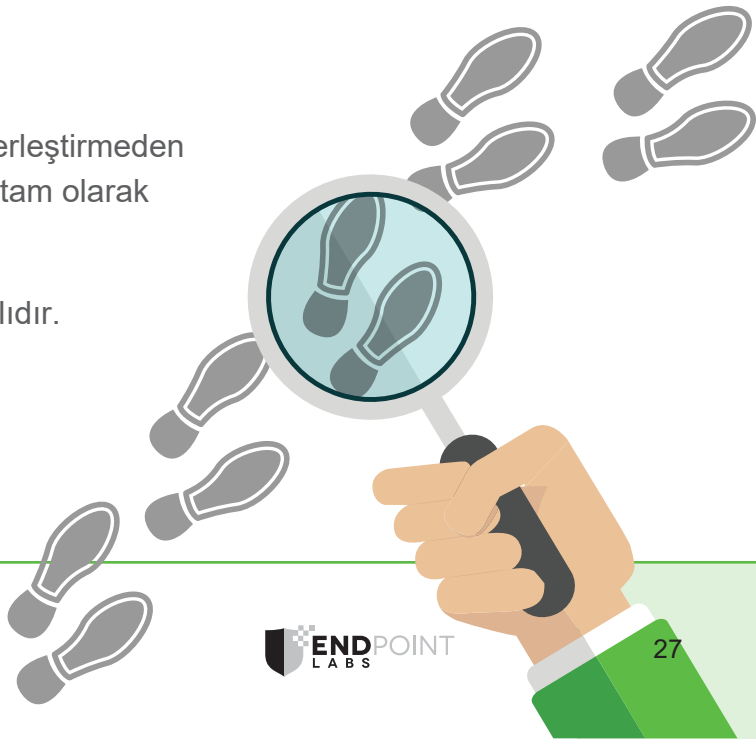
Bir saldırgan, güvenliğe duyarlı eylemlerin denetim günlüklerini tasarlayabilir ve yanlış bir denetim izi bırakarak potansiyel olarak masum bir kullanıcıyı manipüle edebilir veya bir olayı gizleyebilir.

Nasıl olur?

Uygulama, güvenlik açısından hassas eylemler üzerine denetim günlükleri yazar. Denetim günlüğü, veri türü geçerliliği için kontrol edilmeyen veya sonradan temizlenmeyen kullanıcı girdisi içerdiğinden dolayı, girdi, meşru denetim günlüğü verileri gibi görünmek için yapılmış yanlış bilgiler içerebilir.

Nasıl önlenir?

- Kaynaktan bağımsız olarak tam güvenilir olmayan veriler doğrulanmalıdır. Doğrulama bir White List'e dayanmalıdır:
  Kötü kalıpları reddetmek yerine yalnızca belirli bir yapıya uyan veriler kabul edilmelidir.
- Özellikle şunlar kontrol edilmelidir:
  - Veri tipi
  - Boyut
  - Range, sıra
  - Format, şekil
  - Beklenen değerler
- Doğrulama, kodlamanın yerini almaz. Günlüklere yerleştirmeden önce, kaynağına bakılmaksızın tüm dinamik veriler tam olarak kodlanmalıdır.
  - Güvenli bir günlük kaydı mekanizması kullanılmalıdır.



Kod Örneği

Java

```
protected void doPost(HttpServletRequest istek, HttpServletResponse yanit) throws

ServletException, IOException {

String renk = istek.getParameter("renk");

logger.info("{} seçildi.", renk);

if renkListesi.icerik(renk){

        // Yanıt İşleme

}else{

        // Yanıt İşleme
```

Yazar Hakkında

Emin Cüneyt KALPAKOĞLU

01 Endpoint Bilgi Teknolojileri Güvenliği AR-GE A.Ş. kurucusu ve Yönetim Kurulu Başkanı

02 Byte Ltd. Kurucusu ve Yönetim Kurulu Başkanı,

03 Türkiye Bilişim Vakfı Yönetim Kurulu Üyesi,

04 OWASP Üyesi,

SOFTWARE = SECURITY

AWARDS

Outstanding Performance Award of IBM Turkey , 1991 – 2001 (consecutively) - European Champion Award of IBM Europe EMEA ,1997 Budapest - Best Performance Award IBM Turkey IBM i-series Projects , 2004 - Best Performance Award IBM Turkey IBM i-series Sales, 2004 - IT Security & HSA Award of Vision Solutions - IBM EMEA ,2006 Malaga - TC Kadir Has Üniversitesi Turkey "IT Honor Award" 2006 İstanbul - Software Development Award of IBM Turkey ,2007 İstanbul - ITO (Istanbul Chamber of Commerce) Technology & Innovation Award 2012 (First Place in all Categories) - World's Most Dynamic Entrepreneur to watch 2020 of Turkey - 2020 Innovation Partner Award by CHECKMARX

TEŞEKKÜRLER

Checkmarx

ENDPOINT LABS

www.endpoint-labs.com