Ders 7 - Input Validation Issues - Part 1

Dersin Hedefi

Diva uygulamasındaki “7. Input Validation Issues - Part 1” seçeneğine tıklayın. Bu seçenek ile açılan kullanıcı arama sayfasında bir açıklıktan faydalanarak kendi kullanıcı adınız dışındaki veritabanında yüklü diğer kullanıcıları tespit edin (Not: Diva uygulamasında kayıtlı hesabınızdaki kullanıcı adınızın “john” olduğunu varsayın).

Dersin Açıklaması

Girdi denetleme eksiklikleri ……………….

Gömülü sistemlerde çalışan uygulamalar tıpkı desktop sistemlerdeki veya sunucu sistemlerdeki uygulamaların ihtiyaç duydukları gibi bir veritabanı çözümüne ihtiyaç duyabilirler. Sektörde bilinen veritabanı sunucu çözümleri MySQL, Oracle, PostgreSQL, Microsoft SQL Server,... desktop ve sunucu sistemler için işlevsel konumdadırlar. Fakat gömülü cihazlar disk / bellek boyutu ve performans bakımından ufak limitlerde çalışabildiklerinden veritabanı sunucusu çözümleri aynı işlevselliği sunamayabilirler. Örneğin gömülü cihazlardaki ufak bellek kapasitesi dolayısıyla veritabanı sunucusu çözümü ağır / hantal gelebilir ve uygulama performansını oldukça aşağı çekebilir. Bu v.b. durumlar nedeniyle gömülü sistemler için hafifsıklet veritabanı çözümleri geliştirilmiştir. Bunlar arasında Firebird, SQL Server Compact Edition, SQLite,… gibi seçenekler mevcuttur.

Gömülü sistemler içerisinde mobil cihazlarda kullanımı sıklıkla tercih edilen SQLite bu yazının konusudur. SQLite hafifsıklet, sunucusuz (salt dosya halinde), kompakt, gömülü sistemlerde çalışabilen bir veritabanı çözümüdür. Bu veritabanı istemci/sunucu şeklinde çalışan veritabanları yerine yerelde veri depolama şeklinde çalışır. Yani SQLite network ile çalışmak yerine, yerelde sistemde depolu dosyadan sorgu ile cevap çıkarma, işlem yapma hizmeti sunar. C dilinde yazılmış sqlite yerelde bir çeşit detaylandırılmış fopen() işlevi görür. SQLite gömülü sistem veritabanı, veritabanı sunucusu çözümleri kadar kapsamlı bir hizmet sunmaz, fakat çoğu işlevlerini yerine getirebilir durumdadır. Android sanal sistemimizdeki diva uygulaması sqlite veritabanı kullanmaktadır.

Güvenlik açısından bakacak olursak normalde desktop veya web uygulamalarında merkezi bir veritabanı var olur ve saldırganlar veritabanı sunucusuna sızarak veri kaçırma gerçekleştirir. Fakat sqlite veritabanı kullanıldığı durumlarda uygulamayı kullanan her cihazın yerel dosya sisteminde ayrı ayrı veritabanı dosyası var olur ve saldırganlar istemci uçlara (cihazlara) sızarak veri kaçırma gerçekleştirir. Yani merkezi veritabanı sunucusunda uygulama savunmasızsa veritabanı risk altında olur. Gömülü sistem veritabanlarında istemci uçlar (cihazlar) veya istemci uçlardaki uygulamalar (cihazlardaki uygulamalar) savunmasızlarsa veritabanları risk altında olur.

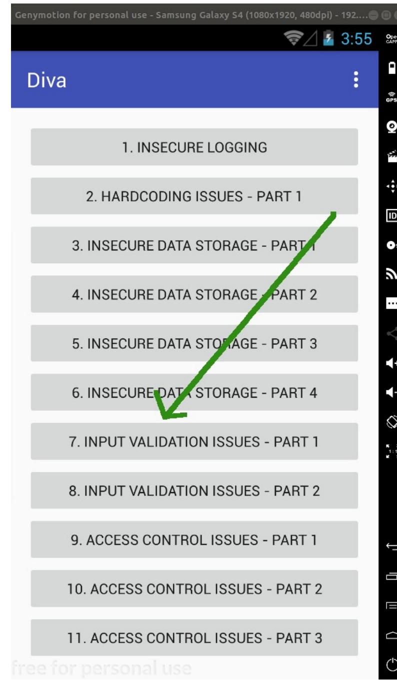
Gömülü sistemler içerisinde adlandırılan mobil cihazlar dünyasında saldırganlar hemen hemen bütün kritik verileri mobil cihazdan (istemci uçtan) elde ederler. Örneğin mobil cihaza (istemci uca) sızma ve mobil cihazın (istemci ucun) dosya sisteminde var olan bir veritabanı dosyasını elde etme gibi. Mobil cihazlarda (istemci uçlarda) kullanılan her bir uygulamanın kullanıcı verileri yerel sistemde ayrı ayrı bir arada depolu olduğundan mobil cihazların (istemci uçların) güvenliği önem arz eder. Çünkü bir mobil cihaza (istemci uca) başarılı sızma girişimi birden fazla uygulamaya ait kullanıcı kritik verilerinin elde edilebilmesine neden olabilir.

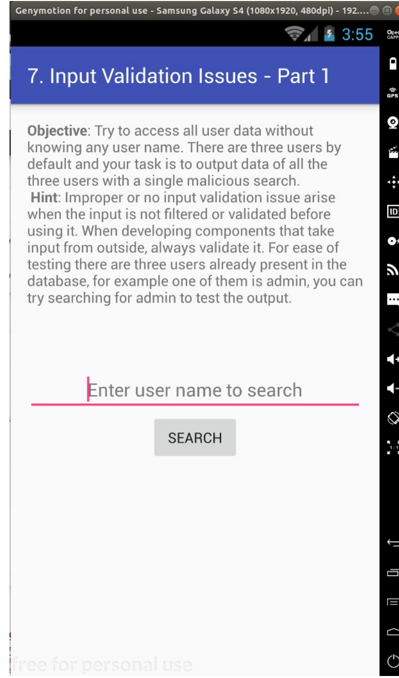
Gömülü sistem veritabanları ile veriler yerelde tutulduklarından kritik öneme sahip veriler yerelde şifreli halde tutulmalıdırlar. Bu sayede istemci uca başarılı bir sızma girişimi olursa saldırganın elde edebileceği veritabanı dosyası ona kritik önemdeki verileri açık metin yerine şifreli halde sunacaktır. Bu ise saldırgana kritik önemdeki verileri elde etme noktasında bir güçlük sunacağından bir güvenlik önlemi / katmanı olacaktır. Normalde saldırganlar ele geçirdiği veritabanı dosyalarındaki verilerle zarar odaklı faaliyetlerini sürdürebilirler / genişletebilirler. Fakat kritik önemdeki veriler şifreli halde olursa bu zarar odaklı faaliyetlerini sürdürmesi / genişletmesi önünde bir engel konulmuş olur. Bu nedenle gömülü sistem veritabanlarında kritik önemdeki veriler tıpkı veritabanı sunucusu çözümlerinde olduğu gibi şifreli halde tutulmalıdırlar.

Mobil cihazlardaki veritabanı çözümü kullanan uygulamalarda var olabilecek bir girdi açıklığı saldırganların uygulama üzerinden arkadaki veritabanına ulaşmalarına ve veri kaçırmalarına olanak verir. Bu saldırı türüne sql enjeksiyonu adı verilmektedir. Saldırganlara uygulamadaki girdi açıklığı yoluyla sızma ve veri kaçırma fırsatını vermemek için uygulamalarda girdi kontrolü / filtrelemesi / engellemesi yapılması gerekir.

Dersin Çözümü

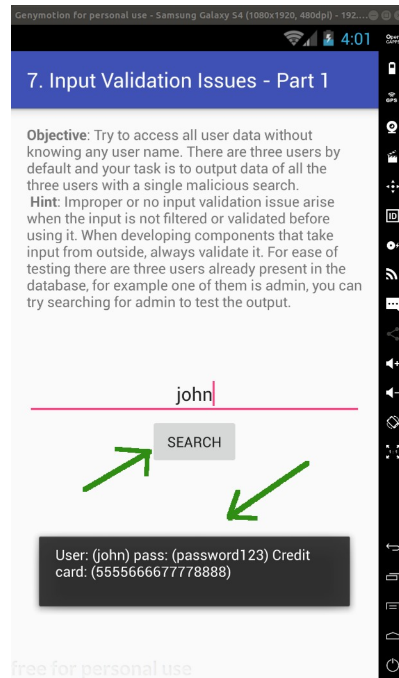
Ders ekranına bir göz atalım.





Uygulama sayfasına göz attığımızda bizden bir kullanıcı adı istemekte ve karşılığında bir arama sonucu döndüreceği gözükmektedir.

Örneğin Diva uygulamamızdaki hesap kullanıcı adımız “john” şeklindedir. Bu kullanıcı adını girdiğimizde sayfa bize veritabanındaki kaydını dönmektedir.



Bu ders sayfası basitleştirilmiş bir kullanıcıdan gelen girdinin veritabanında sorgulanması ve karşılığında bir kayıt varsa ekrana yansıtılması uygulamasının canlandırmasıdır. Bu basitleştirilmiş canlandırmaya göre girilen kullanıcı adı karşılığında veritabanındaki detay bilgiler ekrana basılmaktadır.

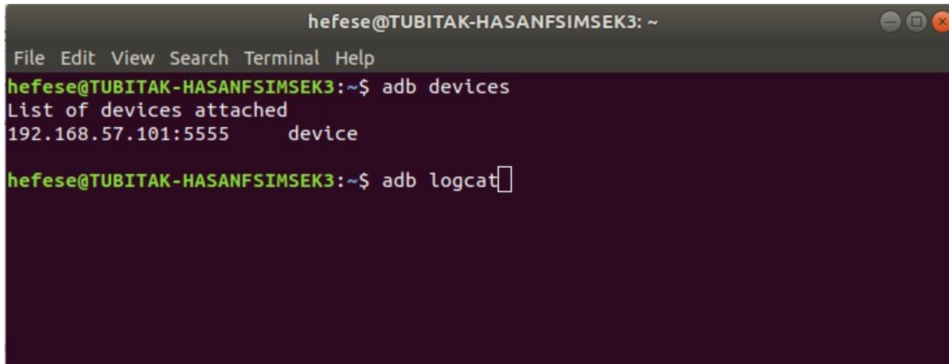
Normal şartlarda herhangi sıradan bir kullanıcı bu sayfada kendi bilgilerini girecektir ve karşılığında detay bilgilerini elde edecektir. Fakat kullanıcının kullanıyor olduğu mobil cihaz farklı ellere geçerse bu uygulama sayfasında bilinen bir girdi kontrolü eksikliği olduğu durumda saldırgan tarafından kullanıcı detay bilgileri kaçırılabilir.

Bu uygulama sayfasının davranışı gözlemlendiğinde girilen girdiye göre veritabanından bir kayıt döndüğü anlaşılmaktadır. O halde saldırganın bu mobil uygulamayı kendi cihazına indirdiğini ve bu sayfada girdi kontrolü yapılmakta mı testi yaptığını varsayalım. Bu test işlemi için saldırganın öncelikle bilgisayardan adb ile mobil cihazına bağlanması gerekir ve mobil cihazındaki ilgili uygulamadan dönecek hata bilgilerini gözlemlemek için adb logcat komutunu çalıştırması gerekir.

Ubuntu 18.04 LTS Linux Terminal:

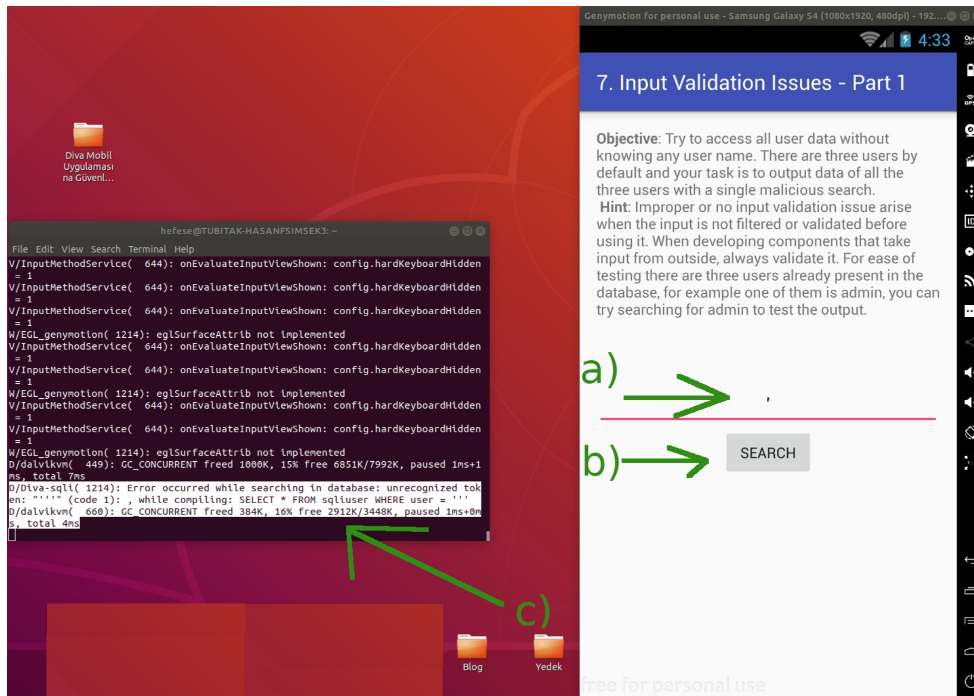
```
> adb devices
> adb logcat
```

Çıktı:



Saldırgan bu şekilde bir yandan mobil cihazındaki log kayıtlarını izlerken diğer yandan girdi kontrolü yapacağı uygulamayı başlatacaktır. Ardından uygulamanın ilgili sayfasında girdi kutucuğuna normal veri yerine veritabanı sorgulamalarında kullanılan özel bir karakteri (tırnak karakterini) girecektir. Bunun sonucunda sayfadaki arkada çalışan kodlar veritabanında bir kayıt bulamayacağı için herhangi bir yanıt dönüşü yapmayacaktır. Fakat girilen tırnak karakteri veritabanı sorgulama cümlesinde fazla bir özel karakter olacağından (yani kullanıcı girdisi / fazlalığı nedeniyle eşleşen bir başka kapatıcı tırnak karakteri olmayacağından) sistem log kayıtlarına bir veritabanı sorgulama cümlesi hata bildirimi düşecektir:

Çıktı:

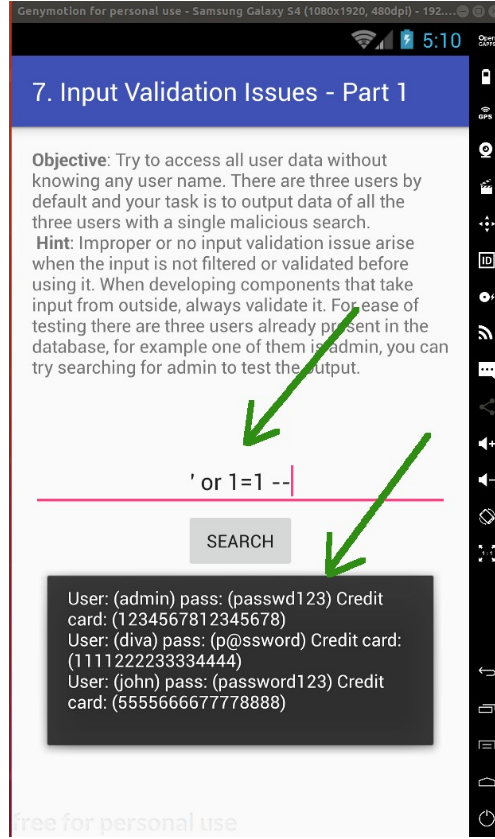


Sistem log'larındaki hata bilgisi şu şekildedir:

D/**Diva**-sqli(1214): Error occurred while searching in database: unrecognized token: "''''" (code 1): , while compiling: SELECT * FROM sqliuser WHERE user = **'''**
D/dalvikvm(660): GC_CONCURRENT freed 384K, 16% free 2912K/3448K, paused 1ms+0ms, total 4ms

Saldırgan tarafından girilen tırnak karakterinin hata üretmesi girdinin veritabanı sorgulama cümlesine filtrelenmeden / engellenmeden eklendiğini gösterir. Bu ise sorgu cümlesinin devamına sorgu cümlesi ilaveleri yapılabileceğini gösterir. Dolayısıyla saldırganlar bu uygulamanın yüklü olduğu mobil cihaz ellerine geçirdiklerinde girdi kutusuna ufak bir veritabanı sorgusu cümleciği yazarak kullanıcı hesap detaylarını elde edebilirler. Örneğin;

Saldırganın Tuşlayacağı Zararlı Girdi: ' or 1=1 --



Saldırganlar girdi kutusuna girecekleri

' or 1=1 --

cümleciği ile sayfada arkada çalışan veritabanı sorgusu cümlesinin devamını devralmaktadırlar ve sonra 1=1 ifadesini ekleyip -- ile cümleciği noktalamaktadırlar. Bu şekilde normalde sayfada çalışan veritabanı sorgulama cümlesinden bir adet kayıt dönecekken 1=1 cümleciği sürekli doğru eşitlik sonucu üreteceğinden mevcut veritabanı tablosundaki tüm kayıtların ekrana verilmesine sebep olacaktır.

Sonuç olarak saldırgan bir başkasının mobil cihazı eline geçtiğinde diva isimli uygulama üzerinde zararlı girdisini girerek cihazdaki veritabanından veri kaçırabilir (örn; kullanıcı hesap bilgilerini) ve kendi faydasına kullanabilir.

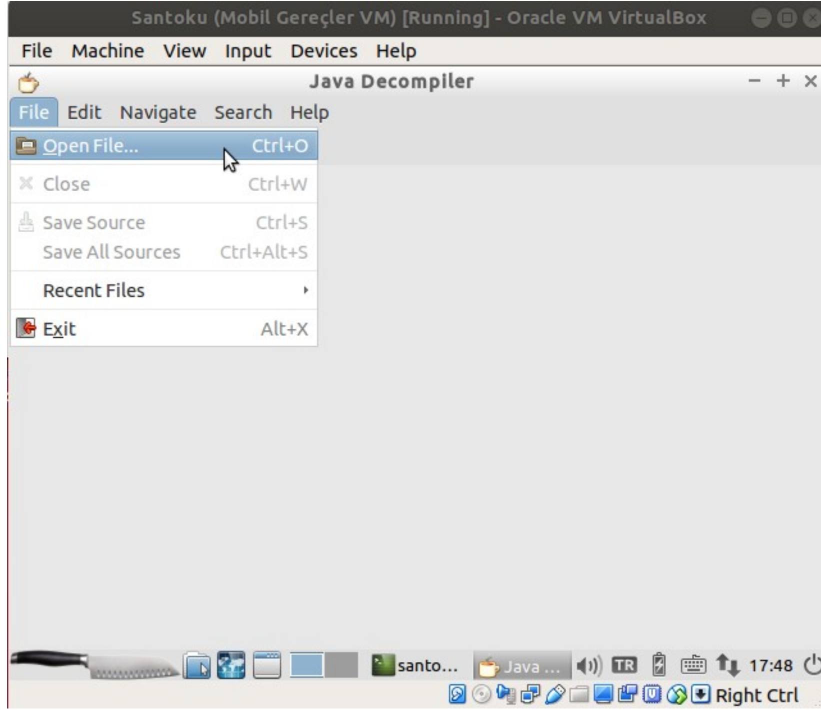
Şimdi bu uygulama sayfasında filtreleme / denetleme eksikliği hangi kod satırında mevcut tespit edelim. Bunun için mobil cihazdan çalışan uygulamanın kaynak kodlarını elde edelim.

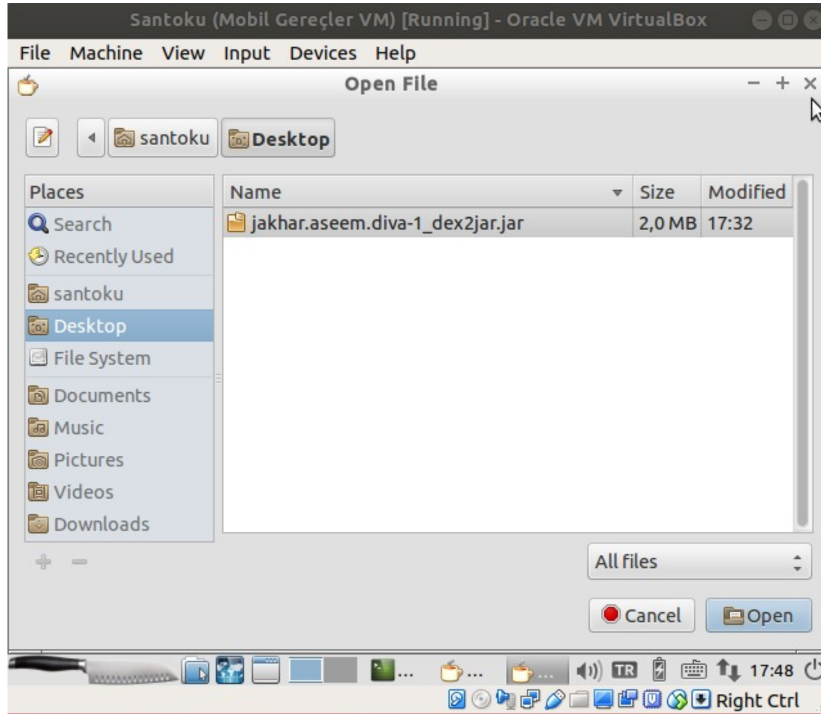
Bu makale dizisinin en başında yaptığımız uygulama binary dosyasını (.apk dosyasını) bilgisayara çekme, üzerinde tersine mühendislik yapma ve uygulamanın okunabilir java dosyalarını elde etme işini tekrarladığımızı varsayalım. Uygulamanın okunabilir java dosyalarını JD-GUI editörü ile inceleyerek bu uygulama sayfasını (“Input Validation Issues > Part 1” sayfasını) sunan / kontrol eden java dosyasını tespit edelim. Bahsedilen işlemlerin ayrıntısı için bkz. Başlangıç: Android Uygulamalarda Tersine Mühendislik ile Okunabilir Java Kaynak Kodları Elde Etme (Dex2Jar, JD-GUI, ApkTool)

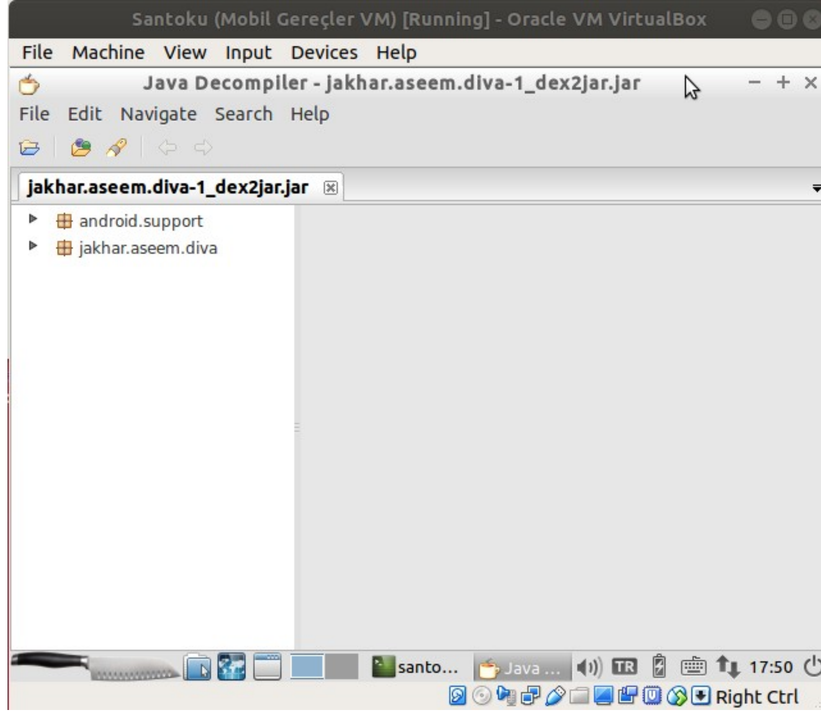
Santoku Linux Terminal:

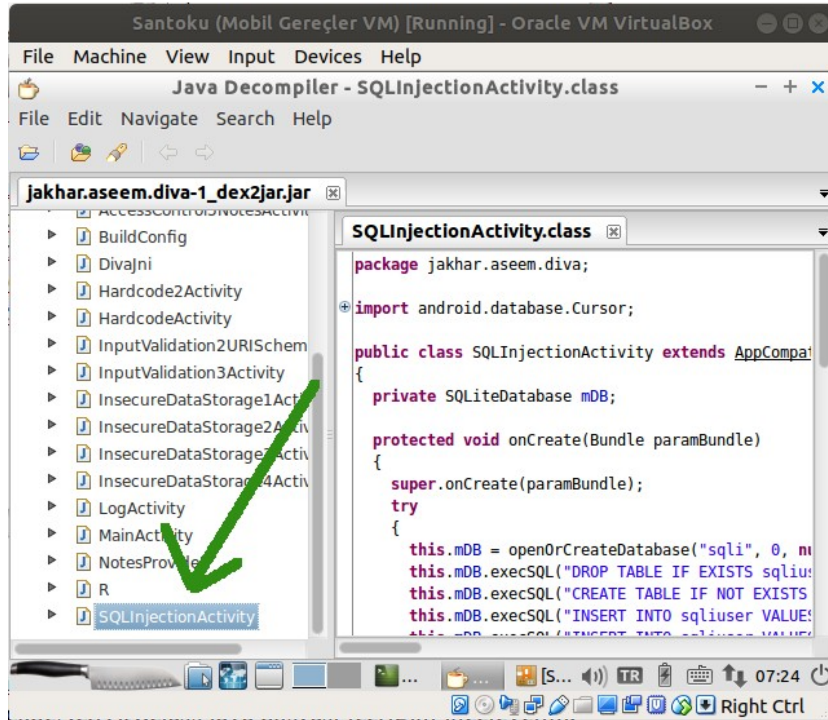
> jd-gui

Çıktı:









Java kaynak kod dosyaları incelendiğinde uygulamada görüntülüyor olduğumuz ekranın hangi java dosyası tarafından sunulduğu / kontrol edildiği yukarıdaki gibi görülebilir. Bu örnek için java dosyasını bulmak uygulama ekranındaki sayfa ismi ile benzerliği dolayısıyla kolay olacaktır ama gerçek bir senaryoda java dosyaları arasında inceleme yapmak ve doğru java dosyasını bulmak için okumalar yapmak gerekecektir.

Mobil cihaz ekranında görüntülemekte olduğumuz “Input Validation Issues > Part 1” sayfasını sunan / kontrol eden SQLInjectionActivity.class java dosyası içeriğini incelediğimizde search() isimli (yani Türkçe ifadeyle arama yap isimli) bir metot kullanımı görünecektir.

SQLInjectionActivity.class:

```
public void search(View paramView)
{
        EditText localEditText = (EditText)findViewById(2131493017);
        try
        {
                Cursor localCursor = this.mDB.rawQuery("SELECT * FROM sqliuser
        WHERE user = '" + localEditText.getText().toString() + "'", null);
                StringBuilder localStringBuilder = new StringBuilder("");
                if ((localCursor != null) && (localCursor.getCount() > 0))
                {
                        localCursor.moveToFirst();
                        do
                                localStringBuilder.append("User: (" + localCursor.getString(0)
+ ") pass: (" + localCursor.getString(1) + ") Credit Card: (" + localCursor.getString(2) + ")\n");
                        while (localCursor.moveToNext());
                }
                while(true)
                {
                        Toast.makeText(this, localStringBuilder.toString(), 0).show();
                        return;
                        localStringBuilder.append("User: (" +
localEditText.getText().toString() + ") not found");
                }
        }
        catch (Exception localException)
        {
                Log.d("Diva-sqli", "Error occurred while searching in database: " +
localException.getMessage());
        }
}
```

Metot içerisi incelendiğinde uygulama sayfasındaki metin kutusu nesne halinde, içerdiği verilerle beraber localEditText1'e atanmaktadır. Sonra try isimli bloğa girilmektedir. Bu blokta daha önceden tanımlanmış bir nesnenin (this.mDB'nin) rawQuery() isimli metodu çağırılmaktadır ve bir veritabanı sorgulama cümlesi kullanıcı girdisi ekli halde çalıştırılmaktadır. Bu sorguya karşılık veritabanından dönen kayıt bir nesneye atanmaktadır. Daha sonra nesnedeki kayıt verisi if bloğu içerisinde kullanıcı adı, parola, kredi kart no halinde bölümlendirilmektedir. Son olarak ise while döngüsünü içerisine girilip bölümlendirilmiş kayıt verisi Toast.makeText metoduyla ekrana basılmaktadır.

[\] Bir Ayrıntı:

Bu java dosyasının yukarılarında this.mDB nesnesi bir sqlite veritabanı nesnesi olarak tanımlanmıştır. Ardından onCreate() isimli metot içerisinde (bu metot android dünyasında sayfa çalışırken otomatik çağrılan / çalıştırılan metot anlamına gelir) this.mDB ile bir veritabanı klasörü (dizini) oluşturulmaktadır.

(Sqlite veritabanı nesnesi this.mDB tanımlanıyor)

(Sqlite nesnesi ile veritabanı klasörü (dizini) oluşturuluyor)

Yani bu bilgilerden hareketle search() metodu içerisinde this.mDB.rawQuery() ile çalışan veritabanı sorgulama cümlesinin sqlite veritabanına yapılmakta olduğunu söyleyebiliriz.

Bu uygulama sayfasını sunan / kontrol eden java kaynak kodundaki kullanıcı girdisini filtreleme / denetleme eksikliği barındıran satır veritabanı sorgulama cümlesinin bulunduğu satırdadır:

public void **search**(View paramView)
{

```
		EditText localEditText = (EditText)findViewById(2131493017);
		try
		{
			Cursor localCursor = this.mDB.rawQuery("SELECT * FROM sqliuser
		WHERE user = '" + localEditText.getText().toString() + "'", null);
			StringBuilder localStringBuilder = new StringBuilder("");
			if ((localCursor != null) && (localCursor.getCount() > 0))
			{
				localCursor.moveToFirst();
				do
					localStringBuilder.append("User: (" + localCursor.getString(0)
+ ") pass: (" + localCursor.getString(1) + ") Credit Card: (" + localCursor.getString(2) + ")\n");
				while (localCursor.moveToNext());
			}
			..
		..
	}
```

Uygulama sayfasındaki ekranda yer alan girdi metin kutusundan gönderilen veriler bu veritabanı sorgulama cümlesinin WHERE ifadesi sonrasında yer alan user kolonuna yerleştirilmektedir. Bu şekilde veritabanındaki ilgili tablodaki kayıtlardan user isimli kolon değerleri kullanıcının gönderdiği veri ile eşleşen kayıt(lar) cevap olarak döndürülmektedir. Kullanıcının girdi metin kutusu aracılığıyla gönderdiği veriler veritabanı sorgulama cümlesine burada olduğu gibi filtrelenmeden / bloklanmadan yerleştiğinden kötü niyetli kullanıcılara sql sorgusunu devam ettirme yolu açık bırakılmış olmaktadır. Çünkü bu derste yapıldığı gibi örneğin tırnak karakteri girerek sorgu devam ettirilebilir ve bir adet kayıt döndürmek üzere yapılandırılmış sorgu birden fazla kayıt döndürmek üzerine çalıştırılabilir: ' or 1=1 --

\>>> Girdi Denetleme / Kontrol Eksikliği

```
Cursor localCursor = this.mDB.rawQuery("SELECT * FROM sqliuser WHERE user = '" +
localEditText.getText().toString() + "'", null);
```

Bu nedenle kaynak koddaki veritabanı sorgulama cümlesinde kullanıcı girdisini tutan localEditText.getText().toString() ifadesi sakıncalı içeriğe karşı (örn; özel anlam ifade eden karakterlere karşı) filtrelenerek veya komple bloklama prosedürüne tabi tutularak sorgulama cümlesine yerleştirilmelidir.

Sonuç

Uygulamadaki var olan girdi kontrolü eksikliği yoluyla girdiğimiz zararlı kodlar sonucu hedef uygulamadan olması gerekenden daha fazla sayıda bilgi açığa çıkardık. Bu yaptığımız saldırıya sql enjeksiyonu adı verilmektedir. Bu uygulamanın açıklığını bilen bir saldırgan herhangi birinin mobil cihazını eline aldığında bu uygulamayı barındırıyorsa gireceği ufak bir payload (zararlı kod) ile elindeki mobil cihazda var olan sqlite veritabanındaki bilgileri elde edebilir durumdadır. Dolayısıyla bu senaryo saldırganın elinde kapalı kutu bir cihaz var olduğu durumda ve bu cihaza gireceği ufak payload'lar ile içerisinde yüklü veritabanındaki verileri elde etme olarak düşünülebilir.

Kaynaklar

https://blog.secureideas.com/2014/10/sqlite-good-bad-embedded-database.html
https://www.sqlite.org/whentouse.html
https://stackoverflow.com/questions/52201959/can-another-person-access-my-sqlite-data
https://en.wikipedia.org/wiki/Embedded_database
https://en.wikipedia.org/wiki/SQLite
https://dba.stackexchange.com/questions/21/is-it-possible-to-use-sqlite-as-a-client-server-database
https://stackoverflow.com/questions/31695766/no-credentials-needed-for-sqlite-database
https://dev.to/lefebvre/sqlite-is-not-a-server-56il