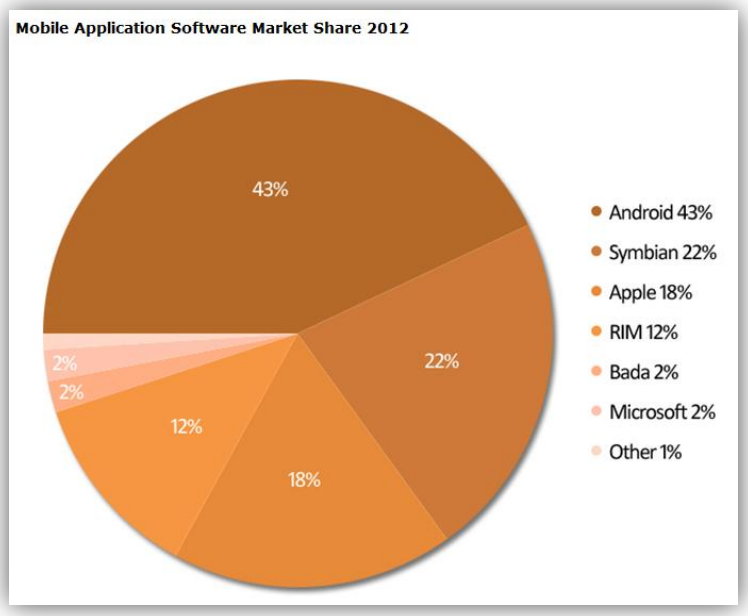
Güvenli Android Uygulama Geliştirme İpuçları

Lokasyon bazlı servisler, mobil sosyal ağlar, mobil bilgi arama, mobil ödeme (NFC), obje tanıma, mobil mesajlaşma ve e-posta, mobil video gibi trendler mobil uygulamaların artarak geliştirilmesini sağlayacaklardır.

Bu noktada, her geliştirme platformunda olması gerektiği gibi Android platformunda da güvenli uygulama geliştirme tekniklerine ihtiyaç vardır.



Şekil 1 Mobil uygulama pazar payı - 2012 , http://www.lexiconnect.co.uk/mobile-internet.html

Mobil uygulama pazarı değişime oldukça müsait gözükmekte olmasına rağmen üretilen uygulamaların o veya bu şekilde pazardaki akıllı cihazlarda uzun süre yaşayacağı unutulmamalıdır. Her durumda uygulamaların mobil uygulamalara has riskleri belirlenmeli ve güvenli geliştirilmeye çalışılmalıdır.

Bu doküman, Android uygulamalarında güvenli geliştirme yapabilmek için bazı ipuçları barındırır. Doküman içerisinde birçok yerde "Hassas" kategorisindeki verilerden bahsedilmektedir. Hassas veriler uygulamadan uygulamaya değişirken, aşağıdaki veriler her zaman hassas olarak değerlendirilebilir;

- TCKN, adres, isim, soyisim, telefon numarası gibi bilgi mahremiyeti kapsamına giren kişisel bilgiler
- Konuşma, SMS verileri,
- Ticari bilgiler,
- Şifre, kriptoğrafi anahtarlar,
- Sağlık bilgileri

1. Intent'ler hassas veri transferinde kullanılmamalıdır

Android platformunda Intent, gerçekleştirmek istenen operasyonların soyut bir tanımıdır. Intent'ler süreçler arasında veri trafiğini gerçekleştiren mekanizmalardır.

Android uygulamalara, anlık çalışan Task'leri listelemenin yanında yakın zamanda çalıştırılmış Task'leri listeleyebilme yeteneği sunmuştur. Bu şekilde, listelenen Task'leri tetikleyen Intent'lere ulaşılabilmektedir (*ActivityManager.getRecentTasks*).

```
ActivityManager m = (ActivityManager)this.getSystemService(ACTIVITY_SERVICE);

List<ActivityManager.RecentTaskInfo> recentTaskInfos = m.getRecentTasks(10,
                                                ActivityManager.RECENT_WITH_EXCLUDED);

if(recentTaskInfos != null){
        for(ActivityManager.RecentTaskInfo rti : recentTaskInfos){
                Log.d("GETTING RECENT TASKS", rti.toString());
                Log.d("GETTING RECENT TASKS", rti.baseIntent.toString());
                Intent i = rti.baseIntent;
                if(i != null && rti.baseIntent.getData() != null){
                        Log.d("GETTING RECENT TASKS", rti.baseIntent.getData().toString());
                }
        }
}
```

android.permission.GET_TASKS iznini almış bir uygulama, yukarıdaki kodu çalıştırabilir ve Intent verilerine erişebilir. Bu nedenle Intent'ler kullanılırken hassas verilerin açık şekilde kullanılmaması gerekmektedir.

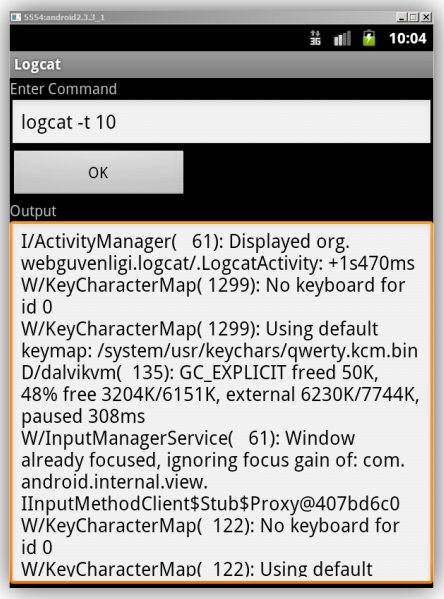
Ayrıca bir Activity'i başlatırken kullanılan Intent'ler, hedef Activity'lere ulaşmadan saldırgan bir uygulama tarafından doğru IntentFilter'ler ile araya girerek çalınabilirler.

Uygulamalar arası hassas veri gönderimi, veri depolama API'ları kullanılarak (Content Provider, SharedPreferences) yetkilendirme kontrolleri ile beraber gerçekleştirilmelidir.

2. Hassas veriler LogCat ile kayıt işlemlerinde kullanılmamalıdır

Debug çıktılarının toplanması ve gösterilmesi için Android, logcat isminde bir kayıt sistemi barındırmaktadır. Bazı sistem ve uygulama kayıtları bu sistemde toplanırlar ve logcat komutu ile bu kayıtlar gösterilebilir ve filtrelenebilir. Logcat, özellikle uygulama geliştiricileri için geliştirme ve problem çözme esnasında kullanılır.

Uygun hakka (*android.permission.READ_LOGS*) sahip bir uygulama Şekil 2'de gösterildiği gibi, LogCat yapısının içeriğini okuyabilmektedir.



Şekil 2 Java komut çalıştırma API Runtime.getRunTime kullanılarak kayıt dosyası içeriğine erişim

Uygulamalar hassas verileri kayıt dosyalarına yazmamalıdırlar. Bu durum özellikle gerçek ortamda çalışan uygulamalar için çok önemlidir. Kayıt dosyalarına hassas bilgiler hiç yazılmamalı veya yazılması zorunlu ise maskelenerek yazılmalıdır.

3. Girdi denetimi

Android uygulamaları bir çok noktadan veri alabilir. Girdi kaynağı, kullanıcılar olabildiği gibi başka uygulamalar, SMS kanalları ve Internet kaynakları da olabilir.

Özellikle, Intent'ler yolu ile tetiklenen bileşenler veriler üzerinde mutlaka beyaz liste kontrolü gerçekleştirmelidir. Aşağıdaki kod parçası bir uygulamadan diğer bir uygulamadaki Activity'nin tetiklenmesinin ve veri gönderilmesinin bir yolunu göstermektedir.

```
Intent i = new Intent(Intent.ACTION_MAIN);
PackageManager manager = getPackageManager();
i = manager.getLaunchIntentForPackage("org.webguvenligi.activity2");
i.putExtra("url", "http://www.webguvenligi.org/");
i.addCategory(Intent.CATEGORY_LAUNCHER);
startActivity(i);
```

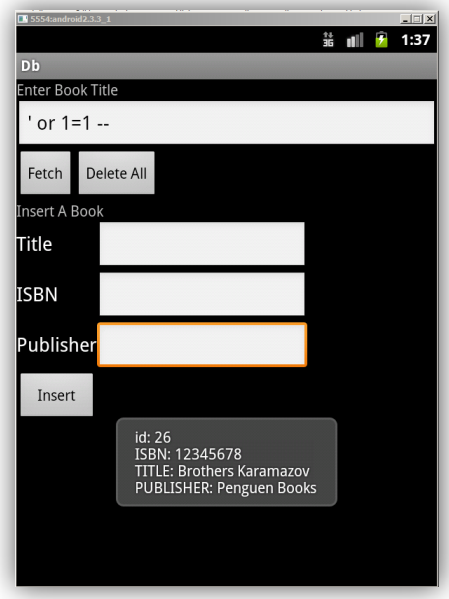
Bu durumda tetiklenen uygulamanın Intent aracılığı ile aldığı veriler, kullanılmadan önce (sorgu parametresi v.b.) mutlaka yetkilendirme kontrollerinden ve girdi denetiminden geçirilmelidir.

```
Intent i = getIntent();
String url = i.getStringExtra("url");
// url whitelist kontrolünden geçirilmelidir.
```

Beyaz liste girdi kontrolü, denetimi yapılan girdinin her zaman güvenli olduğu anlamına gelmez. Bu nedenle injection saldırılarını önlemek için mutlaka API'lar güvenli olarak kullanılmalıdır. Bu duruma en güzel örnek SQL Injection saldırısıdır. Uygulamalar güvenilmeyen kaynaklardan aldıkları girdileri SQL sorgularını oluşturmak için dinamik olarak kullandıklarında SQL Injection zafiyeti oluşur. Çok bilinen bu saldırıda, saldırgan girdiği SQL parçaları ile veritabanında yetkisiz işlemler koşturabilir.

```
public Cursor getTitle(String title) throws SQLException{
        Cursor mCursor =
        db.query(true, DATABASE_TABLE, new String[] {
                                KEY_ROWID,
                                KEY_ISBN,
                                KEY_TITLE,
                                KEY_PUBLISHER
                                },
                                KEY_TITLE + "='" + title +"'",
                                null,
                                null,
                                null,
                                null,
                                null);
        if (mCursor != null) {
                mCursor.moveToFirst();
        }
```

Yukarıdaki kod parçası ile beraber Şekil 3'de kullanıcıdan alınan girdi ile SQL Injection yapılabilen örnek bir uygulama göstermektedir.



Şekil 3 Kitap adı olarak girilen SQL sorgu parçası, veritabanında bulunan ilk kaydı göstermektedir

Uygulamalar güvenilmeyen kaynaklardan aldıkları bütün veriler üzerinde beyaz liste girdi kontrolü gerçekleştirmelidirler. SQL Injection özelinde ise SQL işlemlerinde API'ların sağladığı parametrelenmiş sorgular kullanılmalıdır.

Örnek güvenli bir kod parçası aşağıda gösterilmektedir.

```
public Cursor getTitleSecure(String title) throws SQLException{
        Cursor mCursor =
        db.query(true, DATABASE_TABLE, new String[] {
                                KEY_ROWID,
                                KEY_ISBN,
                                KEY_TITLE,
                                KEY_PUBLISHER
                                },
                                KEY_TITLE + "=?",
                                new String[] {title},
                                null,
                                null,
                                null,
```

```
                                null);
        if (mCursor != null) {
                mCursor.moveToFirst();
        }
```

4. Bileşenler arası kimlik doğrulama ve yetkilendirme kontrolleri uygulanmalıdır

Android uygulamaları bir veya daha fazla bileşenlerden oluşur. Farklı Android bileşenleri aşağıdaki gibi sıralanabilir;

- Activity
- Service
- Content Provider
- Broadcast Receiver

Activity, kullanıcıların etkileşimde oldukları arayüzleri barındırır. Örneğin kullanıcının okuması için akıllı cihaz ekranında listelenen en yakın eczane noktaları, bir Activity ürünüdür.

Service, arkaplanda çalışarak hizmet veren süreçlerdir. Arkaplanda sürekli çalışarak, kur durumlarını Internet üzerinden kontrol eden bir bileşen Service uygulamasına örnek olabilir.

Content Provider, uygulamalar arası verilerin paylaşılması için kullanılan, veritabanı işlemlerinin yapıldığı nokta olarak da düşünülebilen bileşendir.

Broadcast Receiver, sistem veya sistemde bulunan diğer uygulamaların bilgilendirme verilerini dinleyen ve bu veriler geldiğinde çalışması tetiklenen bileşenlerdir. Kur durumundaki bir değişikliği algılayarak kullanıcıya ayrıca bir bilgilendirme mesajı gönderen bir uygulama bu bileşene örnek olarak gösterilebilir.

Bileşenler arası iletişim için kullanılan ana unsur Intent'lerdir. Mesaj göndermek isteyen uygulama bir Intent nesnesi oluşturur ve çalıştırılmak istenen bileşenin ya tam ismi verilir ya da bu nesneye kategoriler eklenerek sistemin hedef alınan bileşeni bulması ve çalıştırılması beklenir. Intent'ler aynı zamanda veri de taşıyabilirler.

Bileşenler farklı uygulama içinde yer alabilirler. Bu şekilde birbirleri ile iletişim kuran bileşenler, kimlik doğrulama / yetkilendirme kontrolleri gerçekleştirmek durumunda kalabilirler. Örneğin Activity B'nin herhangi bir bileşen tarafından başlatılması istenmeyen bir durum olabilir. Bu gibi bir durumda, Activity B'yi içeren uygulama AndroidManifest dosyasında kendini çağırabilen uygulamaların belirli bir izne sahip olup olmadıklarını gösteren bir hak içerebilir.

```
<activity
    android:name=".ActivityB"
    android:label="@string/app_name"
    android:permission="org.webguvenligi.android.permissions.activity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Bu şekilde Activity A'nın, Activity B'yi başarılı bir şekilde çağırabilmesi için

"*org.webguvenligi.android.permissions.activity*"

iznine sahip olması gerekecektir.

Dikkat: Activity B bu şekilde, gerekli izinlere sahip bir uygulamanın kendisini çağırabilmesini sağlamaktadır. Bu da yetkilendirme işlemidir. Ancak Activity B kendisini sadece ve sadece belirli uygulamaların çağırabilmesini isterse, bu durumda şart koşulan hakkın tanımında "signature" protectionLevel'ı olmalıdır. Bu şekilde Activity A, Activity B'yi ancak ve ancak iki uygulama aynı sertifika ile imzalanmış ise çağırabilir.

```
<permission android:name="org.webguvenligi.android.permissions.activity"
    android:label="@string/mylabel"
    android:description="@string/mydescription"
    android:permissionGroup="android.permission-group.COST_MONEY"
    android:protectionLevel="signature" />
```

Duruma tersten bakılırsa, Activity A'nın başlatmak istediği Activity B'yi tanıması nasıl sağlanabilir? Intent'ler kullanılarak başlatılan Activity'ler, sınıf ismi içerebilirler.

```
import org.webguvenligi.activity2.*;

…

Intent explicitIntent = new Intent(Activity1.this, Activity2.class);
startActivity(explicitIntent);
```

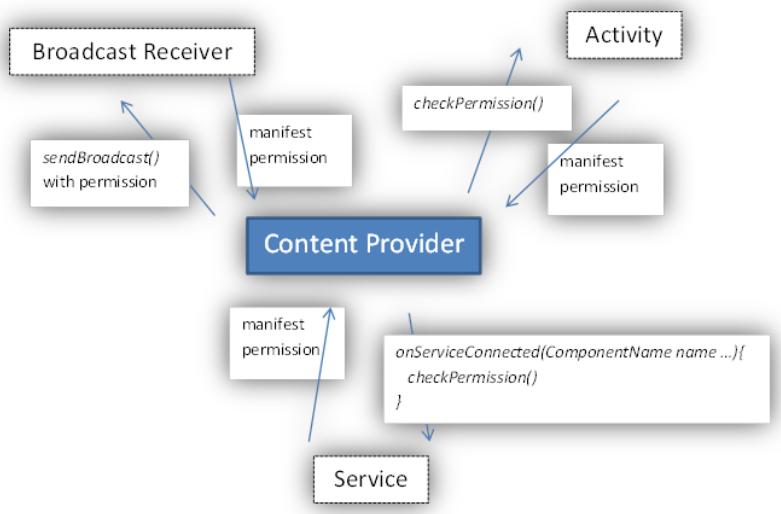
Ayrıca çağırılan uygulamanın package ismi kullanılarak, belirli bir izne sahip olup olmadığı da kontrol edilebilir;

```
PackageManager pm = this.getPackageManager();
int canContinue = pm.checkPermission("org.webguvenligi.android.permissions.activity",
                                     "org.webguvenligi.activity2");
if(canContinue == pm.PERMISSION_GRANTED){
    Intent explicitIntent = new Intent(Activity1.this, Activity2.class);
    startActivity(explicitIntent);
}
```

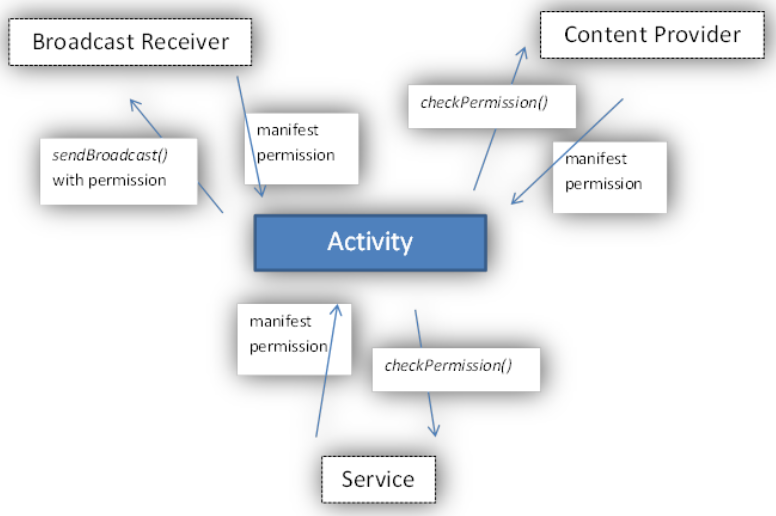
Dikkat: *checkCallingOrSelfPermission* yetki kontrolü içinde bulunduğu metodu çağıran uygulamanın (ki bu uygulamayı metodun barındıran uygulamanın dışında bir uygulama olarak düşünelim) belirli bir izne sahip olup olmadığını kontrol eder. *checkCallingOrSelfPermission* bu kontrol ile kalmaz, içinde bulunduğu metodu barındıran uygulamanın da belirli bir izne sahip olup olmadığını kontrol eder. İki kontrolden biri başarılı olursa, metot PERMISSION_GRANTED değerini döner. Yani izni verir. Bu nedenle *checkCallingOrSelfPermission* metodu mümkün olduğunca kullanılmaması gereken bir metottur.

Android bileşen tiplerine göre alınabilecek yetki kontrolleri aşağıda listelenmeye çalışılmıştır. Bazı durumlarda AndroidManifest ile gerçekleştirilebilecek izin kontrolleri, diğer durumlarda kod parçaları ile gerçekleştirilebilir. Sıkı bir denetim için ise, aynı imzalara sahip uygulamalar kullanılmalıdır.

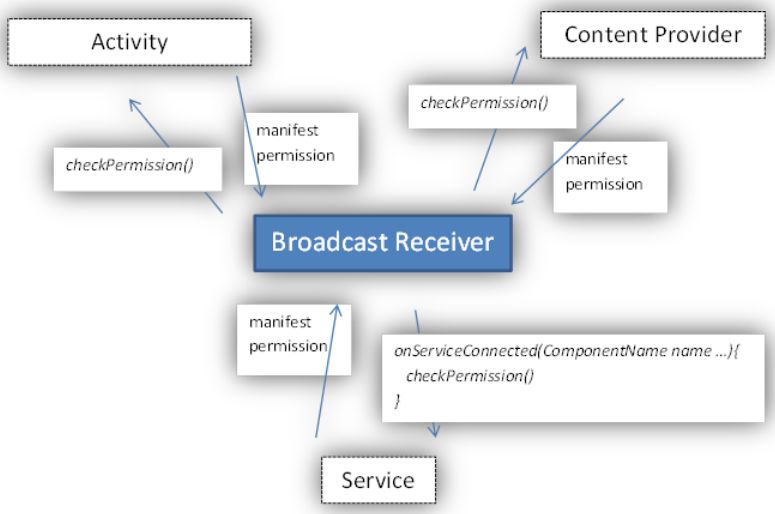
```
if(pm.checkSignatures("org.webguvenligi.activity2", "org.webguvenligi.activity1") ==
                                                                    pm.SIGNATURE_MATCH){
        …
}
```



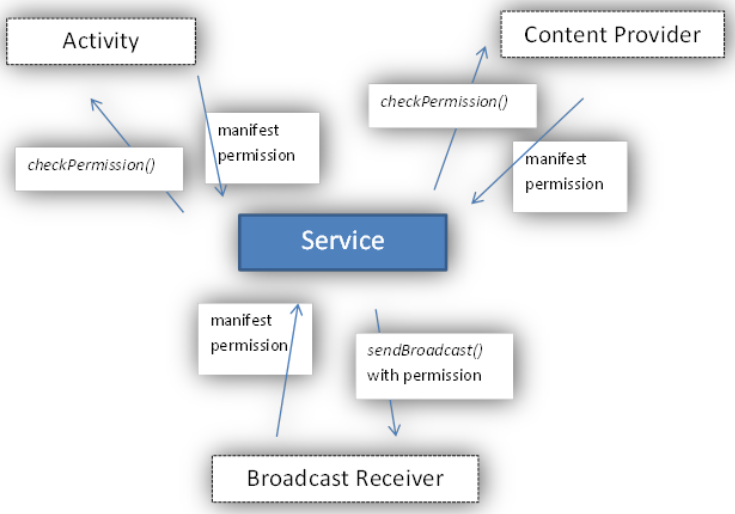
Şekil 4 Content Provider merkezli yetki kontrolleri



Şekil 5 Activity merkezli yetki kontrolleri



Şekil 6 Broadcast Receiver merkezli yetki kontrolleri



Şekil 7 Service merkezli yetki kontrolleri

5. Hassas veriler güvenli bir şekilde depolanmalıdırlar

Açık bir şekilde depolanan her hassas bilgi bir tehdit oluşturmaktadır. Android işletim sistemi sandbox mantığında çalıştığından cihaza kurulu bir uygulamanın hakları sınırlıdır. Bu haklar kullanıcının vereceği izinler ile genişletilebilir. Ancak bu izinler genişletildiğinde bile bir uygulama, diğer bir uygulama izin vermediği müddetçe dosyalarına yetkisiz erişim gerçekleştiremez. Bu durum, Android sisteminde köklü bir güvenlik problemi çıkmadıkça doğrudur.

Uygulamalar verilerini güvenli saklamak durumundadırlar ve bunun için doğru API'leri doğru şekilde kullanmalıdırlar.

Dosya Sistemi İzolasyonu

Varsayılı olarak (by default) uygulamalar tarafından oluşturulan dosyalar sadece uygulamalar tarafından erişilebilmektedirler. Bu sadece root olarak cihaza erişebilen kullanıcılar için geçerli değildir.

Dikkat: Harici bir veri depolama alanında (SD Card) verilerin saklanması durumunda yukarıda bahsedilen izolasyon **sağlanmamaktadır**. Bu nedenle harici alanlarda saklanması gereken verilerin şifrelenmesi gerekmektedir.

Dosya işlemlerinde kullanılan mode'lardan MODE_PRIVATE varsayılı değerdir ve maksimum korumayı sağlar. Diğer değerler MODE_WORLD_READABLE ve MODE_WORLD_WRITABLE, isimlerinden de anlaşılacağı üzere diğer uygulamalara da dosyayı okuma ve dosyaya yazma olanaklarını tanır.

Aşağıdaki örnek kod parçası ile oluşturulan *puanlar* dosyası Şekil 8'de de görülebileceği gibi, diğer uygulamalar tarafından okunamazlar.

```
try{
        FileOutputStream fos = openFileOutput("puanlar", MODE_PRIVATE);
        OutputStreamWriter out = new OutputStreamWriter(fos);
        out.append("oyuncu1#80\n");
        out.append("oyuncu2#34\n");
}
catch(FileNotFoundException fnfe){
        Log.e("FILESYSTEM", fnfe.getMessage());
}
catch(IOException ioe){
        Log.e("FILESYSTEM", ioe.getMessage());
}
```

```
# pwd
pwd
/data/data/org.webguvenligi.filesystem/files
# ls -l
ls -l
-rw-rw-rw- app_40   app_40          0 2012-03-21 15:13 seviye
-rw-rw---- app_40   app_40          0 2012-03-21 15:13 puanlar
#
```

Şekil 8 MODE_PRIVATE ve MODE_WORLD* mode'unda oluşturulan dosyaların sistem hakları

Dikkat: Eğer bir uygulamanın ürettiği veriler başka bir uygulama ile paylaşılmak istenirse ve eğer uygulamalar aynı geliştirici takımı tarafından geliştiriliyorsa uygulamalar için aynı UID'nin kullanılabilir.

Preferences ve Veritabanı İzolasyonu

Verilerin saklanması için Android, dosya operasyonları dışında preferences ve veritabanı metotlarını da sunmaktadır.

SharedPreferences için de MODE_PRIVATE mod'u Android'in sağladığı izolasyonu en güvenli bilgi saklama mode'udur.

Örnek bir kod parçası aşağıda gösterilmiştir. Şekil 9'de oluşturulan puanlar.xml dosyasının hakları gösterilmektedir.

```
SharedPreferences mSharedPrefs = getSharedPreferences("puanlar", MODE_PRIVATE);
SharedPreferences.Editor mPrefsEditor = mSharedPrefs.edit();
mPrefsEditor.putInt("oyuncu1", 84);
mPrefsEditor.putInt("oyuncu2", 24);
mPrefsEditor.commit();
```

```
# pwd
pwd
/data/data/org.webguvenligi.preferences/shared_prefs
# ls -l
ls -l
-rw-rw-rw- app_41    app_41         166 2012-03-21 15:25 seviyeler.xml
-rw-rw---- app_41    app_41         138 2012-03-21 15:25 puanlar.xml
#
```

Şekil 9 MODE_PRIVATE ve MODE_WORLD* mode'unda oluşturulan Preferences dosyaların sistem hakları

MODE_PRIVATE çoğu API için varsayılı değerdir. Örneğin, veritabanı işlemlerinde kullanılabilen aşağıdaki kod parçası

```
private static class DatabaseHelper extends SQLiteOpenHelper{
        public DatabaseHelper(Context context){
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
```

Şekil 10'deki haklar ile veritabanı dosyasını oluşturur. Bu şekilde diğer uygulamalar *books* veritabanına erişemezler.

Dikkat: Bu erişim konusunda Content Provider'ların yetkilendirme kontrolleri de etkilidir. Dosyaya ulaşamayan bir saldırgan, yetkisizi bir Content Provider ile sorgular gerçekleştirerek verilere erişim sağlayabilir.

```
# pwd
pwd
/data/data/org.webguvenligi.db/databases
# ls -l
ls -l
-rw-rw---- app_39    app_39        5120 2012-03-21 13:59 books
#
```

Şekil 10 Varsayılı olarak oluşturulan veritabanı sistem hakları

Veritabanı oluşturmak için openOrCreateDatabase metodu kullanılırsa, MODE_PRIVATE ve diğer mode'lar kullanılabilmektedir. Her zaman ki gibi, MODE_PRIVATE mod'u izolasyonu en yüksek kaynaklar oluşturur.

Bütünlük ve Şifreleme

Android sisteminin sağladığı güvenli veritabanı veya dosya oluşturma, bunları açma API'ları saklanacak verilerin güvenliğini sağlamaktadır. Ama cihazın çalınması gibi durumlarda bu önlemler yeterli olmayacaktır. Bu nedenle kaybolması durumunda riski yüksek veriler dosya sisteminde saklanırken ayrıca şifrelenmelidirler.

Güvenliğin iki önemli maddesi gizlilik (encryption) ve bütünlük (integrity) konusunda Android gerekli API'ları da sağlamaktadır.

Bütünlük konusu herhangi bir dış parametreye gerek duyulmaksızın gerçekleştirilebilirken, gizlilik ancak bir anahtar (key) ile sağlanabilir. Ancak bu durumda da şifreleme işleminde kullanılan anahtarın güvenliği söz konusudur.

Yani, kullanıcılar uygulamayı kullanarak bilgilerini girdiğinde veya uzaktan çektiğinde bu bilgiler güvenli olarak şifrelendikten sonra sistemde saklanmalıdır. Bu şifreleme işlemi için gereken anahtar;

- güvenli olarak nasıl elde edilmelidir?
- güvenli olarak nasıl saklanmalıdır?

Hardcoded olarak uygulamalar içerisine gömülen anahtar, apk dosyalarının incelenmesi ile kolaylıkla bulunabilir. Bu nedenle şifreleme algoritmalarında kullanılacak anahtarlar, kullanıcıdan uygulama her açıldığında alınan kullanıcı adı/şifre ikilisinden üretilebilir, PBKDF (Password Based Key Derivation Function). Bu şekilde üretilen anahtar şifreleme veya şifre çözme işlemlerinde kullanıldıktan sonra saklanmadan atılabilir.

Eğer uygulama, kullanıcının tekrar tekrar kullanıcı adı/şifre girmesini engellemek isterse, ilk kullanıcı adı/şifre doğrulandıktan sonra üretilebilecek bir anahtar doğru API'lar kullanılarak dosya sisteminde saklanabilir. Android bu durum için henüz daha güvenli bir yol sunmamaktadır. Bu metoda uygulamanın risk hesaplanması yapıldıktan sonra onay verilmesi gerekliliği unutulmamalıdır.

6. HTTP iletişim güvenliği HTTPS ile sağlanmalıdır

Bir çok Android uygulaması Internet'e bağlanarak arka uç web uygulamaları noktaları ile veri alış erişinde bulunurlar. Örneğin bir sözlük uygulaması aranan terimlerin anlamlarını bir web uygulaması ile iletişime geçerek cevaplandırır.

En temel bir web bağlantısı aşağıda gösterilmiştir.

```
URL url = new URL("http://www.google.com/");
HttpURLConnection con = (HttpURLConnection) url.openConnection();
readStream(con.getInputStream());
```

Taşınan verilerin hassasiyetine göre (kullanıcı adı, şifre, lokasyon bilgisi, kart numaraları, kişisel bilgiler) HTTPS bağlantı standardı olarak seçilmelidir. Bu işlem için yapılacak değişiklikler basittir.

```
URL url = new URL("https://www.google.com/");
HttpsURLConnection con = (HttpsURLConnection) url.openConnection();
readStream(con.getInputStream());
```

Güvenli bağlantı için iki konu önemlidir;

1. Bağlanılan sunucunun veya istemcinin kimliğinin doğrulanması
2. Bağlantının gizliliğinin sağlanması

İlk madde için genellikle istemcinin kimliği kullanıcı adı ve şifre ile doğrulanırken, sunucunun kimliği ise SSL sertifikalarının kontrolleri ile sağlanır. Sertifikanın güvenilen bir kök sertifika makamı ile imzalanmış olması ve bağlanmak istenen sunucu ile sertifika üzerinde yazan sunucu isminin eşleşmesi sunucunun kimliğinin doğrulanması işleminde çok önemli noktalardır.

Bağlantının gizliliği ise kimlikler doğrulandıktan sonra desteklenen şifreleme algoritmaları ile gerçekleştirilir. Her ne kadar risk tarayıcılardaki kadar yüksek olmasa da (phishing tehlikesi, bağlantı bilgileri hard-coded olacağından düşüktür), araya girme saldırıları nedeniyle (dns/arp poisoning) bu tehlike göz ardı edilmemelidir. Dolayısıyla özel HostnameVerifier sınıfları kullanılmamalıdır.

7. Broadcast verilerinin güvenilirliği sağlanmalıdır

Intent'lerin broadcast edilmesi uygulamalar arası iletişimin bir yöntemidir. Bu şekilde sisteme gönderilen Intent'ler, doğru Intent Filter'lar yardımı ile Broadcast Receiver'ler tarafından yakalanırlar ve işlenirler.

Bu şekilde veri ve bilgilendirme iletişiminde dikkatli olunması gereklidir, çünkü, gönderilen veriler veya bilgilendirmeler herhangi bir uygulama tarafından kayıt edilmiş Broadcast Receiver tarafından da yakalanabilirler. Uygulamalar bu durum söz konusu olduğunda, broadcast edilen Intent'ler ile hassas veriler göndermemelidirler.

Örneğin, SMS ile hassas bir veri bekleyen bir Broadcast Receiver uygulaması olsun. Bu uygulama, aşağıdaki hakka sahip olma ön koşulu ile

```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

aşağıda verilen örnek bir IntentFilter ile SMS geldiğinde sistem tarafından gönderilen broadcast bilgisini yakalayacak ve işleyebilecektir.

```
<intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
</intent-filter>
```

Bu durumda aynı hakka sahip saldırgan bir uygulama,

```
<intent-filter android:priority="999">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
</intent-filter>
```

gibi bir IntentFilter yardımı ve kod onReceive kod parçası ile

```
public void onReceive(Context context, Intent intent) {
    Bundle bundle = intent.getExtras();
    SmsMessage[] msgs = null;
    String str = "";
    if (bundle != null)
    {
        Object[] pdus = (Object[]) bundle.get("pdus");
        msgs = new SmsMessage[pdus.length];
        for (int i=0; i<msgs.length; i++){
            msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
            str += "SMS from " + msgs[i].getOriginatingAddress();
            str += " :";
            str += msgs[i].getMessageBody().toString();
            …
```

SMS mesajına ve içeriğine ulaşabilir.

Hassas veriler SMS ile açık bir şekilde gönderilmemelidir. SMS içeriğinin şifrelemesi bir çözüm olarak düşünülebilir. Dikkat: Aynı durum SMS farklı portlar ile gönderildiğinde de geçerlidir.

Referanslar

https://www.owasp.org/index.php/Data_Validation

http://developer.android.com/reference/android/R.attr.html#sharedUserId

http://code.google.com/p/android-apktool/

http://stackoverflow.com/questions/8091519/pbkdf2-function-in-android

http://labs.mwrinfosecurity.com/tools/2012/03/16/mercury/documentation/quick-start/