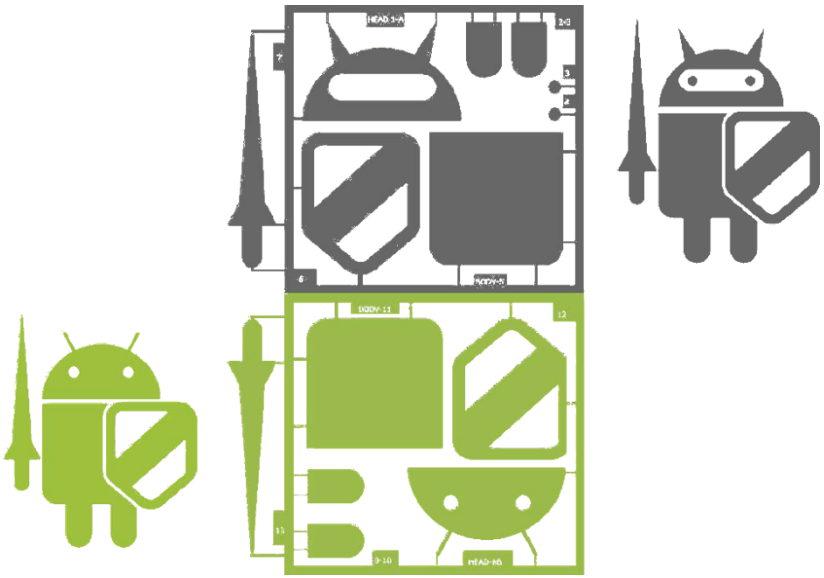
ANDROID

(UN)SECURITY GUIDELINES

Best practice e linee guida per uno sviluppo sicuro e responsabile



$whois

Paolo Stagno

Cyber Security Analist, iDialoghi

Luca Poletti

Crazy Mathematician & Developer

VoidSec.com

voidsec@voidsec.com

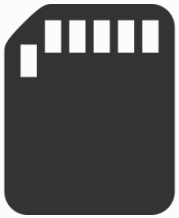
(Un)security

"**Digital trust** is an imperative, mobile app developers must take greater responsibility for ensuring that their applications follow the **secure programming practices** and **vulnerability responses** developed over the past decade, and by doing so provide the level of protection required for us to **trust our digital lives** with them."

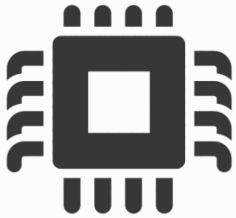
Vincent Weafer, senior vice president of **McAfee**

DATA STORAGE

External vs Internal Storage



- **Dati accessibili da chiunque**
  - `WRITE_EXTERNAL_STORAGE`
  - `READ_EXTERNAL_STORAGE`
- **Memoria accessibile fisicamente**



- **Dati protetti dal sistema Android**
- **Difficile accesso fisico alla memoria**
- **Accessibile con permessi di root**
- **Differenti modalità operative**
  - `MODE_PRIVATE`
  - `MODE_WORLD_READABLE`
  - `MODE_WORLD_WRITABLE`

SharedPreferences & SQLite

- **Salvati nell'Internal Storage**
- **Differenti modalità operative**
  - `MODE_PRIVATE`
  - `MODE_WORLD_READABLE`
  - `MODE_WORLD_WRITABLE`

- **Salvati nell'Internal Storage**
- **Non accessibile da altre applicazioni**
- **Accessibile tramite ADB > sqlite3 con i permessi di root o applicazioni debuggable**

Esempio: SQLite

YAHOO! MESSENGER

Su un dispositivo con root è possibile accedere ai database delle applicazioni.

`/data/data/com.yahoo.mobile.client.android.im/databases/messenger.db`

L'app di messaggistica e di posta salvavano in chiaro i dati degli utenti

- **Conversazioni**
- **Password**

profile_id	buddy_id	message	has
	699	Hey this is yahoo security testing activity	
	699	You are offline	

Log

- **Dati accessibili tramite logcat**
- **In Android < 4.0 qualsiasi app può leggere i log di qualsiasi altra app**
  - `android.permission.READ_LOGS`
- **Non usare i log per tener traccia di informazioni sensibili**

Usare la RAM

- **Accesso alla RAM consentito solo con permessi di root**
- **Liberare sempre la memoria dalle informazioni sensibili**
- **Non usare mai elementi immutabili per informazioni sensibili**
  - `char[]` **può essere sovrascitto dall'utente**
  - `String` **è immutabile, da evitarsi**

Riassunto

- Criptare **sempre i file sensibili**
- **Potenzialmente tutti i dati scritti possono essere letti, anche se eliminati**
- **Mai salvare su disco dati quali chiavi crittografiche o altri dati estremamente sensibili**
- **I dati sensibili in RAM non devono essere immutabili**
- **Mai rilasciare app con flag**
  `android:debuggable="true"`

ACCESSIBILITY

Content Provider

I ContentProvider permettono la condivisione di dati tra le applicazioni, bisogna essere cauti per evitare accessi non autorizzati ai dati

- **Limitare gli accessi tramite attributo**
  - `android:exported="true"`
  - `android:exported="false"`
- **Definire appropriate permission e protectionLevel**
  - `android:protectionLevel="signature"`

Intents

- **Mai usare Intent impliciti e broadcast pubblici per dati riservati o operazioni critiche**
  - **Intercettazioni e denial of service**
- **Usare Intent espliciti definendo il component name**
  - `setComponent(), setClass(), setClassName(), Intent([…], Class)`

Intents

- **Limitare il broadcast a singole applicazioni**
  - `Intent.setPackage(), LocalBroadcastManager`
- **Mai usare Intent impliciti con i `PendingIntent`**
  - **I `PendingIntent` hanno gli stessi privilegi della applicazione che li ha inviati**

Validation

I dati che vengono ricevuti da terze parti devono sempre essere controllati e resi in forma canonica.

- **URL ricevuti da content provider possono causare vulnerabilità di directory trasversal**
- **I dati presenti sulla SD possono essere alterati da chiunque**
- **I Broadcast Receiver devono verificare l'identità dell'autore dell'Intent**

CRITTOGRAFIA

Homemade Security

«Lo XOR non è un buon sistema di crittografia»

Non si ha prova della robustezza del proprio protocollo

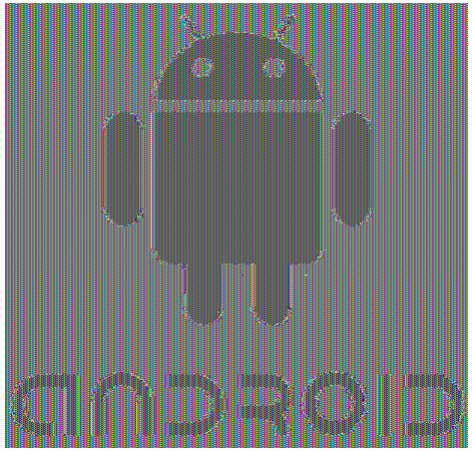
Ci si affida al concetto di *security through obscurity*

Il *reverse engineering* porta alla conoscenza della logica utilizzata

Si possono trovare vulnerabilità che invalidano completamente tutta la segretezza o presunta tale

Quali algoritmi?

AES-ECB →



- **Evitare AES-ECB**
- **IV generati casualmente**
- **Non usare chiavi crittografiche *hard-coded***
- **Nel `SecureRandom()` non usare seed *hard-coded***

Random

In ambienti legati al mondo della sicurezza, usare la classe SecureRandom

- **Evitare di usare il costruttore**
  `SecureRandom(byte[] seed)`
- **Evitare seed statici**
- **Porre particolare attenzione alle versioni Android obsolete (API < 18)**

Funzioni di derivazione della chiave (KDF)

Si può derivare una chiave crittografica a partire da un PIN o da una password

- **PKDF2, scrypt , bcrypt**
- **Usare almeno salt a 128 bit**
- **Usare più di 1000 iterazioni (16000 per scrypt)**
- **Usare delle policy per le password severe**

Certificate Pinning

Per *pinning* si intende il processo di associare ad un host il suo certificato o la sua chiave pubblica.

- **Controllare sempre l'identità dell'interlocutore tramite il certificato *hard-coded***
- **Le informazioni di interesse sono**
  - **Certificati**
  - **Chiavi pubbliche**
  - **Hash dei precedenti**

API

Backend

- **Elemento essenziale delle applicazioni Android è una logica lato server**
- **Mai fidarsi mai degli input dell'utente**
- **Vulnerabilità analoghe allo sviluppo WEB**

SQLi

SQL Injection inserimento di codice malevolo all'interno di una query

```
SELECT * FROM users WHERE user='.$_POST['user'].'
        AND pwd='.$_POST['pwd']'
```

```
SELECT * FROM users WHERE user='utente'
        AND pwd='or 1=1--
```

- **Exploiting: recupero db, tabelle, colonne, dati**
- **Prevenzione: usare le** `PreparedStatement`

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
SQLiteStatement stmt = db.compileStatement("SELECT *
FROM Country WHERE code = ?");
stmt.bindString(1, "US");
stmt.execute();
```

XSS

XSS esecuzione codice malevolo lato client

- raccolta e manipolazione di informazioni (cookie)
- visualizzazione e modifica di dati presenti sui server
- alterazione del comportamento dinamico delle pagine web

Due tipologie

- stored
- reflected effettuate dallo stesso client che subisce l'attacco

CSRF

Cross-site request forgery (CSRF) non viene verificata l'intenzionalità della richiesta

- **XSS sfruttano la fiducia di un utente in un sito**
- **CSRF sfruttano la fiducia di un sito nel browser di un utente**

CSRF - Esempio

L'utente è autenticato sul sito della sua banca.

```
www.example.com/pagamento?
    importo=XXXX&destinatario=YYYY
```

L'attaccante invia all'utente un tag img html

```
<IMG src="www.example.com/pagamento?
    importo=2000&destinatario=Attaccante">
```

Quando l'utente tenterà di accedere all'immagine, il browser invierà di fatto una richiesta HTTP alla pagina web della banca; Il sito rileverà, tramite il cookie, che la richiesta arriva effettivamente dall'utente e autorizzerà l'operazione.

User Enumeration

User Enumeration test con lo scopo di individuare un insieme di utenti di un determinato servizio

- **Compromissione di credenziali**
- **Recupero di informazioni non pubbliche**

Esempio:

- ***http://trade.aliexpress.com/mailingaddress/mailingAddress.htm?mailingAddressId=123456***
- **mailingAddressId è l'id dell'account utente**
- **Gli id sono stati assegnati in maniera incrementale e prevedibile**

OFFUSCAMENTO

Motivazioni

Serve a rendere meno comprensibile ad un lettore umano il codice sorgente

- **Proteggere la proprietà intellettuale dal riutilizzo non autorizzato del proprio codice**
- **Rendere difficile il reverse engineering**
- **Rendere più difficile la modifica malevola**
- **Evitare azioni di violazione della licenza d'uso, come la creazione di crack**

Offuscamento vs Performances

Offuscamento ≠ Minify

Minify

- **Riduce il validation e il class loading time**

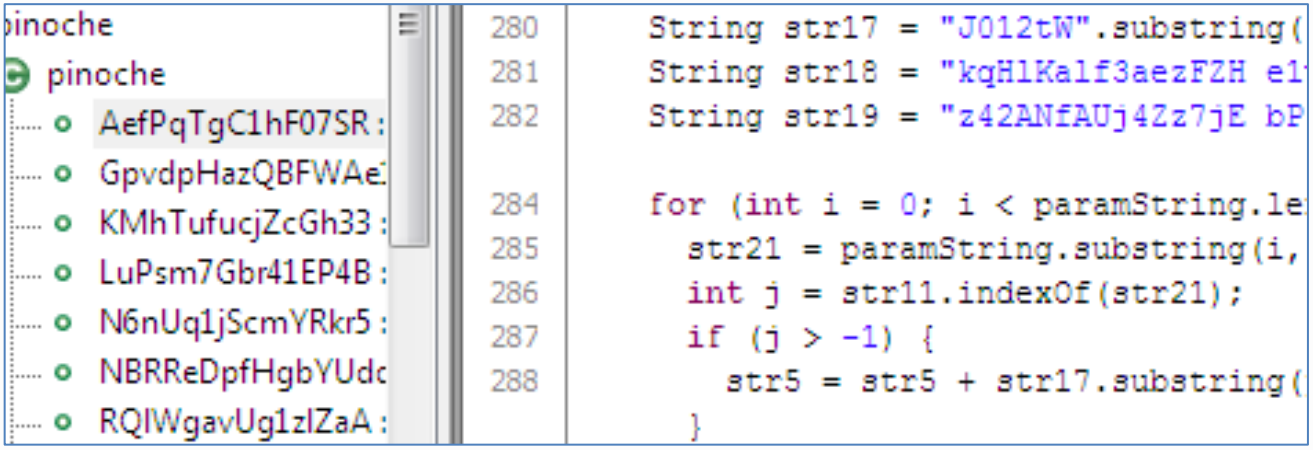
Offuscamento

- **Comunemente l'offuscamento porta ad un degrado delle performance**
- **Genericamente si ha una perdita trascurabile**
- **Opera a livello di bytecode**

```
acct.setBalance(newBalance)


if(x=0)a.s(b);
else{ //do something else }
```

Tipologie



Due tipi di offuscamento

- **Prima generazione: rinomina il package, le classi, I metodi e le variabili, elimina I commenti e le informazioni di debug.**
- **Seconda generazione: cambia il flusso di controllo, introduce cicli, condizioni e salti condizionati**

Conclusioni

In questo breve talk abbiamo intravisto alcune delle problematiche che possono verificarsi in ambito Android.

- **Crittografate sempre i dati sensibili**
- **Validate sempre gli input**
- **Non rilasciate app debuggabili**
- **Controllate i certificati**
- **Mettete in sicurezza il backend**
- **Offuscate**

Riferimenti

- http://developer.android.com/training/articles/security-tips.html
- https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=111509535
- https://www.nowsecure.com/resources/secure-mobile-development/
- http://www.jssec.org/dl/android_securecoding_en.pdf
- http://android-developers.blogspot.it/2013/08/some-securerandom-thoughts.html
- http://thehackernews.com/2014/12/aliexpress-website-vulnerability_7.html
- http://securityaffairs.co/wordpress/34084/malware/mcafee-labs-threat-report-201402.html
- http://voidsec.com/yahoo-messenger/

Domande?

Paolo Stagno

Cyber Security Analist, iDialoghi

Luca Poletti

Crazy Mathematician & Developer

"Some things in life are unpredictable,
your App doesn't have to be one of them"

VoidSec.com

voidsec@voidsec.com