Mobile applications security – Android OS (case study)

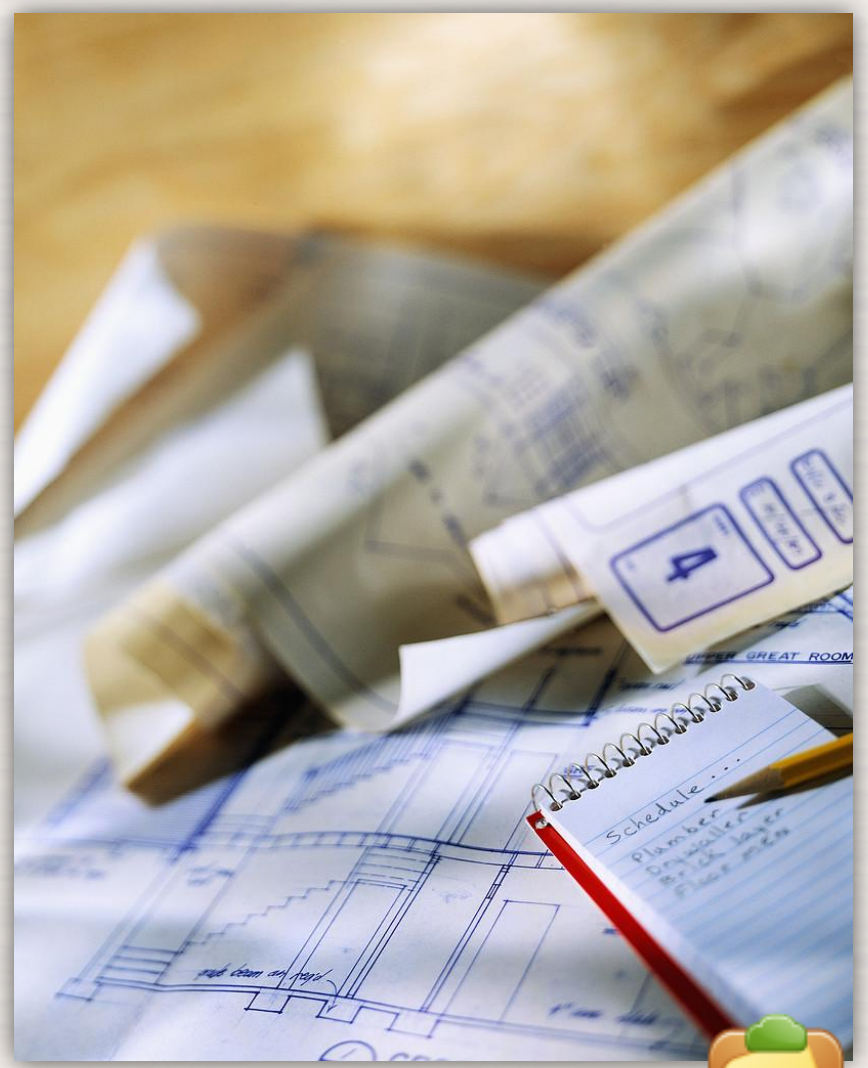
Maciej Olewiński

Cryptographic Seminar

16.05.2012r.

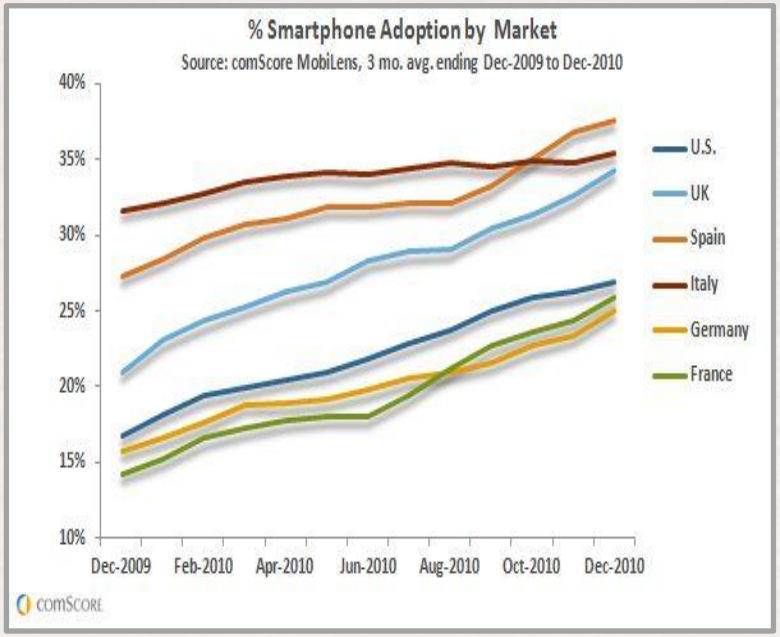
Presentation's schedule

- Mobile devices market
  - Smartphone's domination is coming
- Android basics
  - Main facts
  - Architecture model
- Applications security
  - On lower architectural layers
  - Essential principles of higher layers design
  - Android runtime as „local web services" environment
  - Explicit data sharing
  - Permissions mechanism
  - Known vulnerabilities
- Appendix
- Summary

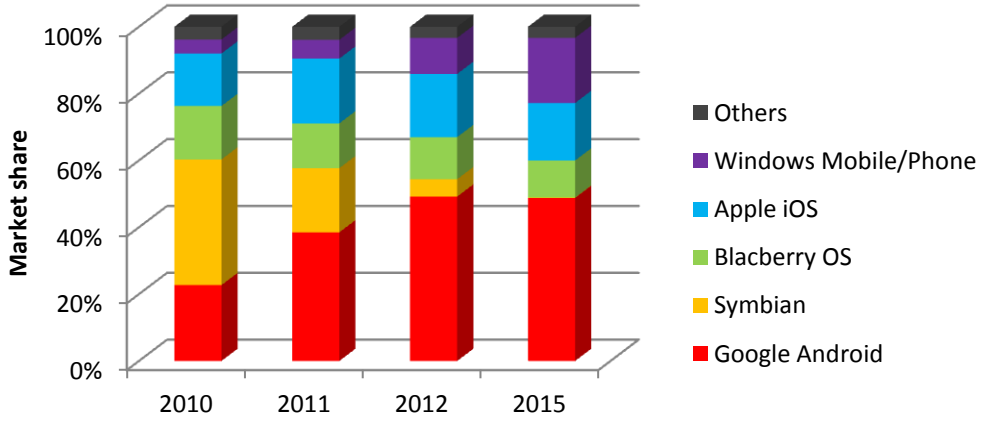


Smartphone's growth

- Smartphone's on their way for mobile world domination
  - Huge popularity grow
  - Popularization of mobile applications
  - Smartphones replaces more and more devices: mp3 players, simple cameras etc.
  - More and more sensitive data in small, portable devices

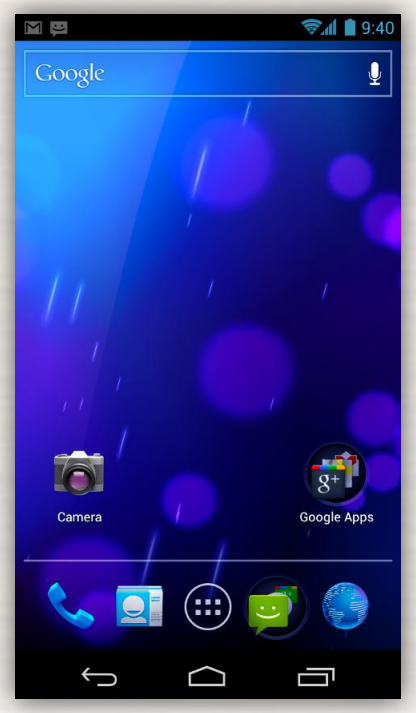


- Main smartphone's mobile platforms
  - Google Android
  - Symbian
  - iOS (Apple)
  - BlackberryOS
  - Windows Mobile/Windows Phone
  - Bada



Mobile platforms market share in years 2010-2015 (prediction) - Gartner

What is Android?

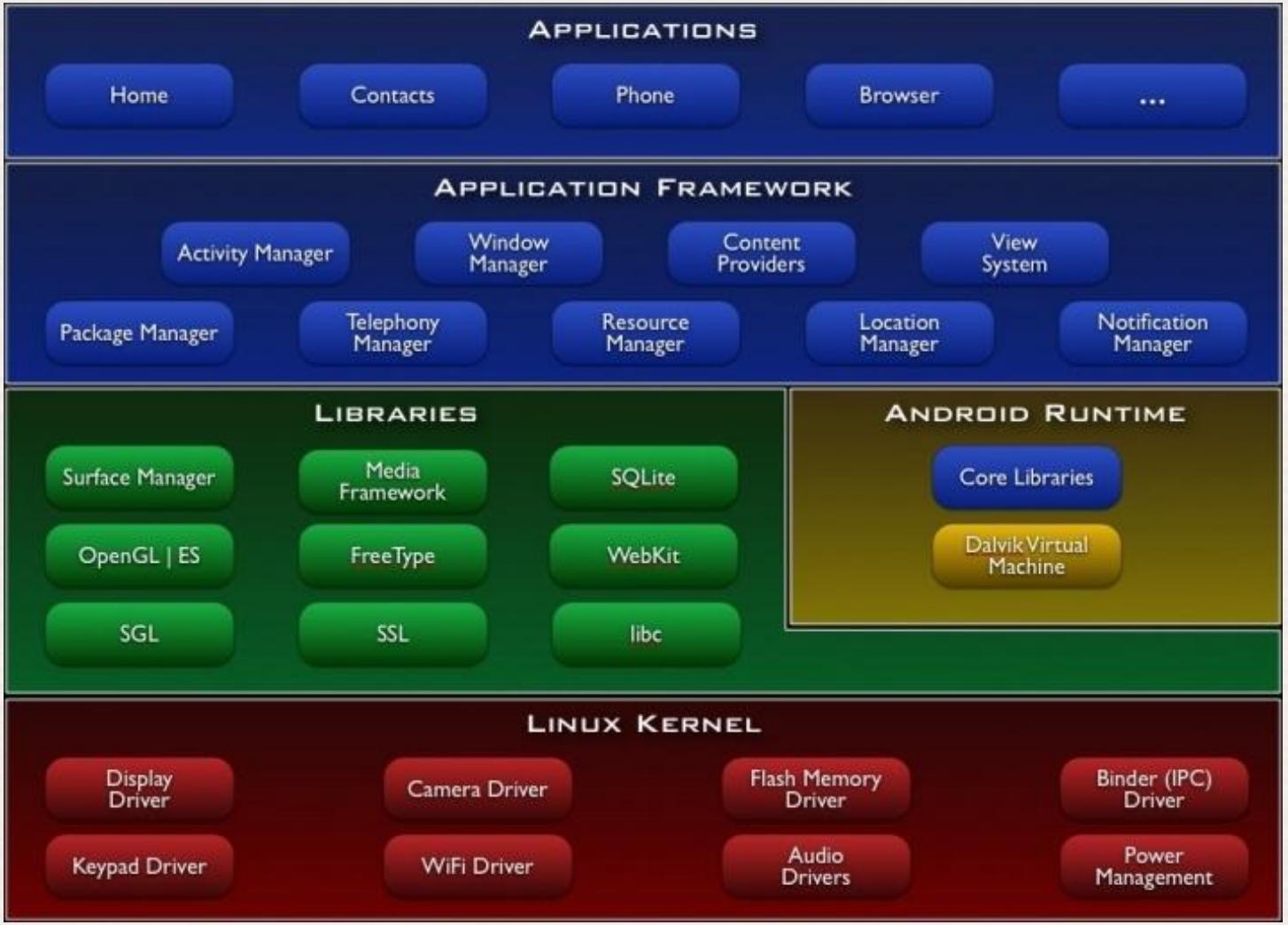


- *„Android is a software stack for mobile devices that includes an operating system, middleware and key applications”*
  - Created by Android Inc., a company acquired by Google in 2005
  - Developed under supervision of Open Handset Alliance (OHA)
  - Released and open-sourced in 2008

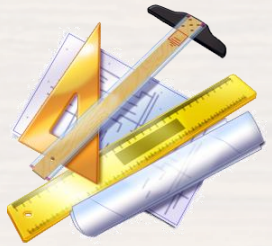
- Based on popular technologies
  - Linux Kernel
    - Not a typical Linux distribution
  - Java language
    - Not a typical Java platform

- The most popular smartphone’s OS in 2011
  - Also the fastest growing

Android architecture – the model



Android architecture – basic facts



- All applications are equal (with small exceptions for Phone application)
  - System applications have no precedence over 3rd party software (with small exception for safe-mode)
- All Android applications are written in Java programming language
  - Android SDK
- Android is not a typical Java platform; it only uses the language
  - Custom implementation of Virtual Machine – Dalvik VM (non-compliant with any Sun/Oracle solutions)
  - Totally redesigned application lifecycle mechanisms, user interface principles and more
  - Main packages set derived from Java SE platform (not from Java ME!)
    - Removed desktop-typical packages (e.g. Swing, AWT etc.)
    - Added mobile-typical packages (for telephony, location-based services, user interface and more)
- Possibility to use native code in Android applications
  - Compiled C/C++ libraries can be used by Android applications using NDK (JNI)
  - All Android security model principles are also applied to those libraries
- Custom security model
  - Defined across the all architectural layers

Applications security – lower layers

- *„A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user”* – Android Developers

- Android extensively use Linux-level security mechanisms
  - Even though each application package has its own instance of Dalvik Virtual Machine, it also gets its own Linux User ID (UID) and works in its own process
  - Any data used by application is assigned to package’s UID and is not accessible for other packages
  - For any data sharing higher-level (Android specific) mechanisms should be used (full protection on Linux level); exposing all application data for other applications must be done explicitly

- Applications signing for secure updates
  - Each application must be signed by developer’s certificate
  - Certificate can be self-signed as it is only used to distinguish if the newer version of application can work under the same UID

Applications – the essentials (1)

- All Android applications are build of 4 types of basic components
  - Activities
    - Main user interface containers - an Activity represents one single screen visible by user
  - Services
    - Used for performing long-running, background operations
  - Content Providers
    - Used for sharing data between application – explicit way for fully manageable exposing the data
  - Broadcast Receivers
    - Used for receiving broadcasts (something like Java listeners, but on higher level)

- Querying Content Provider uses URI's and MIME types for RESTful-like interface

- Interaction between Activities, Services and Broadcast Receivers is „driven by Intents"
  - Intents are kind of asynchronous messages used for components activation

Applications – the essentials (2)

- Intents – the heart of Android
  - Intents can call specific components explicitly by defining target component's class
  - The real power lies in the fact that they can also define only required action and (optionally) data description (with MIME and URI), calling the components implicitly
    - Runtime binding between components based on so called Intent Filters

- Android Applications are typically not „solid"
  - Extensive use of runtime binding between components instead of compilation time binding (with Intents)
  - Application are typically „a set of independent components", rather than homogenous entity
  - Anyway, it is possible to build „solid" application by using explicitly defined Intent targets

- Android OS – flexible, extensible environment of cooperating, distributed components
  - Typically, on almost all popular platforms one application can call another application
  - On Android, because of Intents, one application's **part** can call a **part** of another application
  - Applications don't have to be aware of each other; an OS resolves Intents dynamically
  - Many Intent Filters defined by system applications

Android runtime – „local web services"

- Loosely coupled components works together by specifying „contracts”
  - The same idea that is used in web services and SOA principle

- All components and their capabilities of specific applications are described in AndroidManifest.xml file (mandatory application descriptor for OS)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.android.notepad">
    <application android:icon="@drawable/app_notes" android:label="@string/app_name" >
        [...]
        <activity android:name="NotesList" android:label="@string/title_notes_list">
            [...]
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                [...]
                <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
            </intent-filter>
            [...]
        </activity>
            [...]
    </application>
</manifest>
```

- Where is the security?
  - Is the security of lower Android architecture’s layers spoiled by higher layers?

Android permissions

- Permissions as the security boundary between components
  - All basic applications components (Activities, Services, Content Providers and Broadcast Services) can define specific permission that another component must have granted to use them
  - Every application that uses components with defined permissions must have explicit admission granted by user during application's installation

- Components define permissions that they **require** in AndroidManifest.xml

```
<permission android:name="com.example.android.notepad.permission.READING_NOTES"
        android:label="Czytanie danych notatek"
        android:description="Umożliwia czytanie danych notatek. Złośliwe aplikacje mogą wysłać te dane
                             osobom trzecim"
        android:protectionLevel="dangerous" />
```

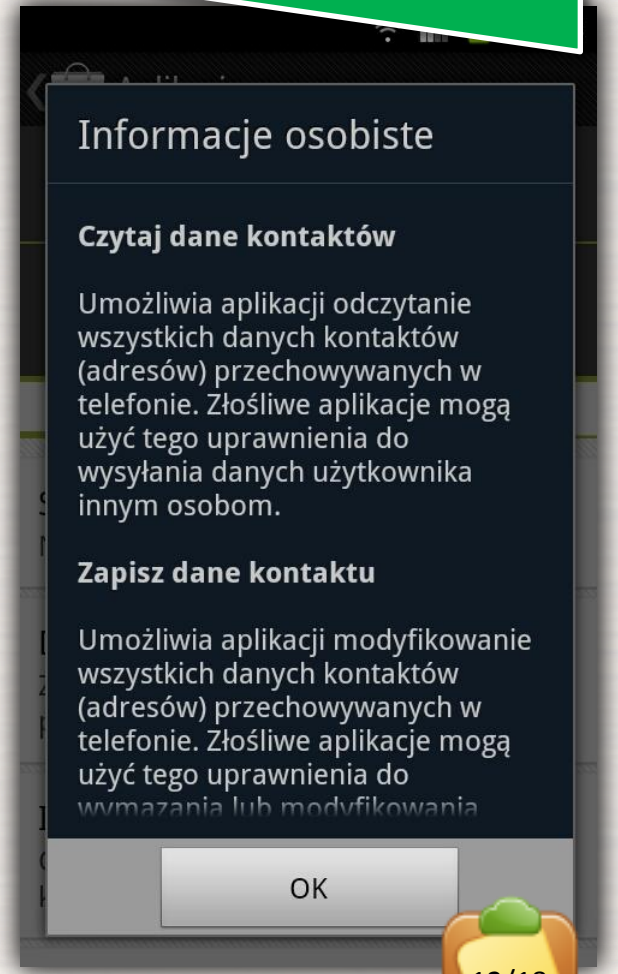
- Application that want to use components which require permission declares that it **uses** permission in its AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.android.app.otherApp" >
            [...]
            <uses-permission android:name="com.example.android.notepad.permission.READING_NOTES" />
            [...]
</manifest>
```

How do permissions work?

Application can not to define uses-permission data, but it will cause runtime security error while trying to access protected components

- Application's A component declares in its AndroidManifest.xml that it **requires** permission
- Application B which wants to use application's A component declare in its AndroidManifest.xml file that it **uses** permissions
- During installation of application B, OS reads its AndroidManifest.xml file looking for **uses**-permission declarations. If found, it asks user for granting permission
- OS shows permission description defined in application A
- In case of users refusal, OS stops an installation
- In case of user's acceptance, application is installed and can use capabilities that has just got granted
  - Problem – no control if application uses it in secure way (driving licencse and no road police?)
  - Result: with user's acceptance even the malicious software can be installed ☹
  - But it can also be easily uninstalled ☺



Permissions – summary

- Easy and extensible mechanism for creating security boundary in distributed environment of application's components
  - Android is very security-strict in its lower layers
  - Even though its higher layers define rich environment of independent, cooperating components
- Fully Optional for exposing data/functionality
- Strictly required for using permissions-protected data/functionality
- Very convenient to use
  - Applications developer don't have to create any additional code
  - Declaring permission name and description is sufficient
  - Android OS is responsible for forcing security contract keeping
- Perfect solution?

„Priviledge escalation” attack (1)

- Problem: Android OS does not provide transitive permissions checking

- Attack scenario (technical view)
  - Application A exposes data/functionality and requires permission P
  - Application B uses A’s data/functionality and has permission P granted (it declares its usage in AndroidManifest.xml and user has accepted it during B’s installation)
  - In the same time, application B exposes data/functionality that uses permissions-protected applications’ A data/functionality inside and does not require any permissions
  - Application C does not have permission P to use applications A data/functionality directly
  - Anyway, application C can use data/functionality exposed by application B because B does not require any permissions
  - Application C can use application’s A data functionality indirectly with no permissions granted!

- Attack scenario (real-life view)
  - An attacker offer useful application X that uses sensitive data/functionality
  - User grants all permissions to application X; he/she does not know that application X exposes the data/functionality further without requiring any permissions
  - An attacker encourage user to install another application Y that does not need any permissions to be granted
  - User accepts application Y; after installation it can stole/corrupt data

„Priviledge escalation” attack (2)

- Solution (official) – Android designers has known about this problem
  - Explicit, runtime checking permissions mechanism is also present in Android platform
  - Checking must be done by application that uses and exposes data/functionality in the same time (it means application B in naming convention used in this presentation)
  - Very, very bad solution – when we expose data/functionality in our application, we must trust the developers of external applications
    - They must not to expose our data/functionality further, or…
    - …they must check in runtime if other applications using their interfaces has the permission to our application granted
    - We don’t have any control of this!

The main cause of described problem lies outside of our application; though, we must rely on a good design of external applications which may be (intentionally or not) spoiled

- Solution (good practices)
  - The most obvious tip: we should not to expose any data/functionality if we really don’t have to
  - We can use `android:exported` property to all application components in our AndroidManifest.xml
    - If set to `false` for a specific component, it disables external access to this component
  - We can resign for dynamic Intents resolving, defining target components explicitly by class name

- Conclusion
  - Event though there are some traps, building secure Android applications is possible

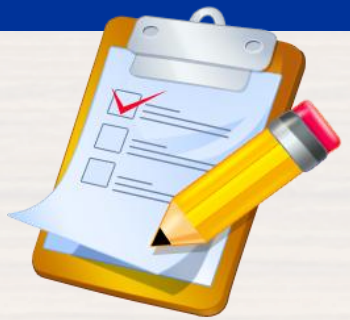
Security appendix

- Cryptography in Android
  - Full standard packages set from Java SE ready to use with Android SDK
    - Two independent sets of HTTP packages: native Java SE and taken from Apache Foundation
  - All good security coding practices from Java SE can be directly used in Android
    - Java SE cryptography code is fully portable (only performance aspects should be considered)
  - BouncyCastle for Java SE can also be used directly by importing it to project
    - No need for any changes in library – no need for separate Android version

- Android and malware
  - Most of the malware must be explicitly installed by user (psychology aspects important for attackers)
  - Only a few vulnerabilities caused by errors in platform itself
    - Unfortunately, updates are Achilles' heel of Android
  - There is always the possibility to simply uninstall the whole application (in worst case with safe mode)

- The biggest question – Google and privacy
  - Gmail, Calendar, Contacts synchronized with Google Account by default
  - Rich location-based services
  - Voice recognition mechanisms, Face Unlock etc.

Summary



- Mobile devices are becoming more and more popular
  - In the same time, they store more and more sensitive, personal data
  - They are very attractive as an attack target
  - Conflicting requirements: environment flexibility, convenient usage and strict security
- Android is designed to face those problems – and it does it
  - All applications are sandboxed
  - Very strict security solutions on lower architecture layers
  - Rich, flexible and distributed environment of higher levels
  - Security requirements are met because of two basic mechanisms
    - Explicit data sharing
    - Permissions
- Even though there are some known issues, building secure applications are possible and easy

Thank you for your attention

Questions, comments?