Mobile security

Threats

- There are three prime targets for attackers:
  - **Data**: smartphones are devices for data management, and may contain sensitive data like credit card numbers, authentication information, private information, activity logs (calendar, call logs);
  - **Identity**: smartphones are highly customizable, so the device or its contents can easily be associated with a specific person. For example, every mobile device can transmit information related to the owner of the mobile phone contract and an attacker may want to steal the identity of the owner of a smartphone to commit other offenses;
  - **Availability**: attacking a smartphone can limit access to it and deprive the owner of its use.

Consequences

- When a smartphone is infected by an attacker, the attacker can attempt several things:
  - The attacker can manipulate the smartphone as a zombie machine
  - The attacker can easily force the smartphone to make phone calls. (call paid services)
  - A compromised smartphone can record conversations between the user and others and send them to a third party. This can cause user privacy and industrial security problems;
  - An attacker can also steal a user's identity, usurp their identity (with a copy of the user's sim card or even the telephone itself) and thus impersonate the owner.
  - The attacker can reduce the utility of the smartphone, by discharging the battery (attack of "battery exhaustion" or "sleep deprivation torture").
  - The attacker can prevent the operation and/or be starting of the smartphone by making it unusable.
  - The attacker can remove the personal (photos, music, videos, etc.) or professional data (contacts, calendars, notes) of the user.

Attack based on SMS and MMS

- Some attacks derive from flaws in the management of SMS and MMS
- Some mobile phone models have problems in managing binary SMS messages. It is possible, by sending an ill-formed block, to cause the phone to restart, leading to the denial of service attacks. If a user with a Siemens S55 received a text message containing a Chinese character, it would lead to a denial of service.
- In another case, while the standard requires that the maximum size of a Nokia Mail address is 32 characters, some Nokia phones did not verify this standard, so if a user enters an email address over 32 characters, that leads to complete dysfunction of the e-mail handler and puts it out of commission.

Attacks based on applications vulnerabilities

- Web Browser
  - **Jailbreaking** the iPhone with firmware 1.1.1 was based entirely on vulnerabilities on the web browser. In this case, there was a vulnerability based on a stack-based buffer overflow in a library used by the web browser (Libtiff).

Countermeasures

- A central paradigm in mobile operating systems is the idea of a sandbox.
- Since smartphones can run many applications, they must have mechanisms to **ensure these applications are safe**:
  - for the phone itself,
  - for other applications and data on the system,
  - and for the user.
- If a malicious program reaches a mobile device, the vulnerable area presented by the system must be as small as possible.
- Sandboxing extends this idea to compartmentalize different processes, preventing them from interacting and damaging each other.

Jailbreak & rooting

- If a mobile device is jailbreaked/rooted it has acquired the root priviledge and can access to the whole file system and can control all processes
- in this mode applications not coming from the appstore can be installed, these applications can be trojan hiding a malicious part…
- applications on the appstore are checked and it is difficult that contains malware

App development guidelines

- To develop a secure app:
  - consider secuirity problems from the first prototype
  - don't assume safety of 3rd party dependencies
  - use https connection (TLS1.2) to connect with the server, use certificate pinning (do not trust CAs)
  - avoid to store personal information on the phone
  - use tokens to handle sessions
  - implement binary protection (code obfuscation)

Communication

- Encrypt all communication with the server
- TLS 1.2 should be used and everything weaker should be disabled.
- Certificates should be validated correctly and not weakened for testing purposes.
- Self signed certificates should not be accepted unless certificate pinning is in use.
- For particularly sensitive information, additional encryption should be applied on data before sending over TLS.
- Data sent over secure connections should not be logged.

Certificate pinning

- A certification authority may be compromised and generate a trusted certificate for a malicious server
- A certificate of a malicious CA may be stored with the root trusted certificates.
- With certificate pinning the application trust a specific known path (the public keys to be trusted are stored in the application)
- With certificate pinning even self-signed certificates are considered good

Data storage

- Do not save sensitive information (passwords, API keys, credit card numbers) on the local store, get it from the network or ask the user
- if it cannot be avoided encrypt it and use a key received from a server
- never trust data read from the local storage
- disable any caching
- avoid using iOS keychain, it can be read on a jailbroken device
- on iOS you can use the DataProtection APIs to encrypt files with key based on the user PIN

Binary protections

- A binary protection is a client side security control which will attempt to stop an attacker from reverse engineering or modifying the application.
- It cannot be done at 100%, any protection can be bypassed or removed if there is enough time

Obfuscation

- Binary obfuscation is a technique used to keep any client side implementation secret, useful for DRM or mobile payments.
- Binary obfuscation is the practice of transforming a binary into a version that has the same functionality as the original but with steps to make the binary more complex and therefore more difficult to reverse engineer.
- On some platforms (such as Android) when obfuscation is not applied, it is often possible to recover a highly accurate version of the source code.

Obfuscation techniques

- String Encryption
  - String constants are replaced with encrypted versions and run through a decryption function before use.
- Symbol Renaming
  - Symbols (such as method and variable names) are renamed to remove meaning and to make it more difficult to follow method calls.
- Code Flow Obfuscation
  - Code structures are removed, flattened or merged.
- Dummy Code Insertion
  - Code is inserted that has no effect on the program, but is executed.
- Instruction Substitution
  - Common instructions are substituted for instructions that have the same effect, but are less clear. Simple instructions can be expanded to many instructions.

root/jailbroke detection

- the application should detect if it is running on a jailbroken device and stop running
- many checks should be done as the attacker can modify the code to bypass the control
- the controls should be done inline and not using a function
- the root/jailbroke detection on andorid/iOS is not standard, can lead to false positives and then to frustrated users…

Debug protection

- An easy way for an attacker to understand and modify the application is by attaching a debugger.
- Android:
  - checking whether FLAG_DEBUGGABLE is set in the *ApplicationInfo* class and by checking *isDebuggerConnected* and *System.Diagnostics.Debugger.IsAttached*.

Hook detection

- functions can be replaced with a malicious version of the same funtion
- the malicious version may log the arguments to a file or to a server and call the original version
- the attacker can access to protected data
- not easy to detect

Runtime integrity checks

- protect users from trojaned or back doored versions of our applications
- protect the client side security controls we have implemented.
- techniques to be considered for integrity checks:
  - File Checksums - The application should compare the checksums of files (including libraries) used with values known by the application. If the checksums do not match, the application should close.
  - Code Checksums - The application should compare the checksum of functions loaded in memory. If these checksums do not match, the application should close.

Security grade

- Not all techniques need to be adopted for all kinds of applications
- Some applications need a higher security (e.g. bank applications)
- Other a medium security level
- some other a low security

Security grade

Security	Communication	Storage	Binary protection
High	Certificate Pinning is implemented and nothing traverses a cleartext channel	Nothing sensitive stored in the client	Custom protections is use along with those provided by off-the-shelf software
Medium	All traffic is sent over TLS	Data is encrypted with a PIN/passcode	Binary protections enabled with the use of off-the-shelf software
Low	Traffic is sent in cleartext or TLS/SSL has been weakened for development	Sensitive data is stored within the application sandbox	No binary protections present