ATTACK SURFACE ANALYSIS OF THE TIZEN OS

Irfan Asrar
Intel Security Group, USA

Email irfan_asrar@mcafee.com

ABSTRACT

Tizen is an open-source platform designed for multiple computing platforms such as smartphones, in-vehicle infotainment (IVI), smart TV, wearable devices, consumer electronics, etc. *Tizen* comes at a time when the threat against mobile computing is growing in tandem with the popularity of mobile devices. Compared to *Android*, *iOS* and *Blackberry*, *Tizen* offers several different options within the device structure to combat the rise in malware targeting mobile devices. This paper will examine these options and their ability to counteract malware and privacy threats.

1. INTRODUCTION

Tizen is a combined effort of *Intel* and *Samsung* under the auspices of the *Linux Foundation*, *LiMo Foundation*.

Similar to the architectural framework used in *Firefox OS* [1] and *Android*, *Tizen* uses a layered environment built upon a foundation of the *Linux* kernel. There are three main layers: the application, the core and the kernel [2].

The *Tizen* application architecture can further be divided into two layers. The web framework provides support/runtime to facilitate the execution of HTML5 and JavaScript. The Native framework is composed of system services and a set of native namespaces providing open-source APIs, with which applications can be developed using C/C++ [3].

The core layer consist of the following services: application framework, base, connectivity, graphics, location, messaging, multimedia, PIM (personal information management), security, system, web and telephony.

- The application framework, or AppCore, contains all the middleware, hardware-related services, the *Linux* kernel providing the framework needed to define hardware calls, and APIs for building both native applications and web runtime applications [4]. The following is a list of services for which the AppCore is responsible:
  - Life cycle management for applications
  - Application launch service
  - System event handlers
  - Application configuration
  - Application installation / uninstallation.
- Base contains the basic essential *Linux* system libraries.
- Connectivity provides network-related functionalities.
- Graphics/UI consists of the system graphic and UI stacks, also called the Native Framework.
- Location provides location-based services (LBS), including position information, geocoding, satellite information and GPS status.
- Messaging supports communication formats such as SMS, MMS, email and IM.
- Multimedia provides media support for functions such as video, audio and imaging.
- PIM enables management of user data on the device, including managing the calendar, contacts, tasks, and retrieving data about the device context (such as device position and cable status).
- Security is responsible for security enforcement across the system. It consists of platform security enablers, such as access control, certificate management, secure application distribution and secure I/O.

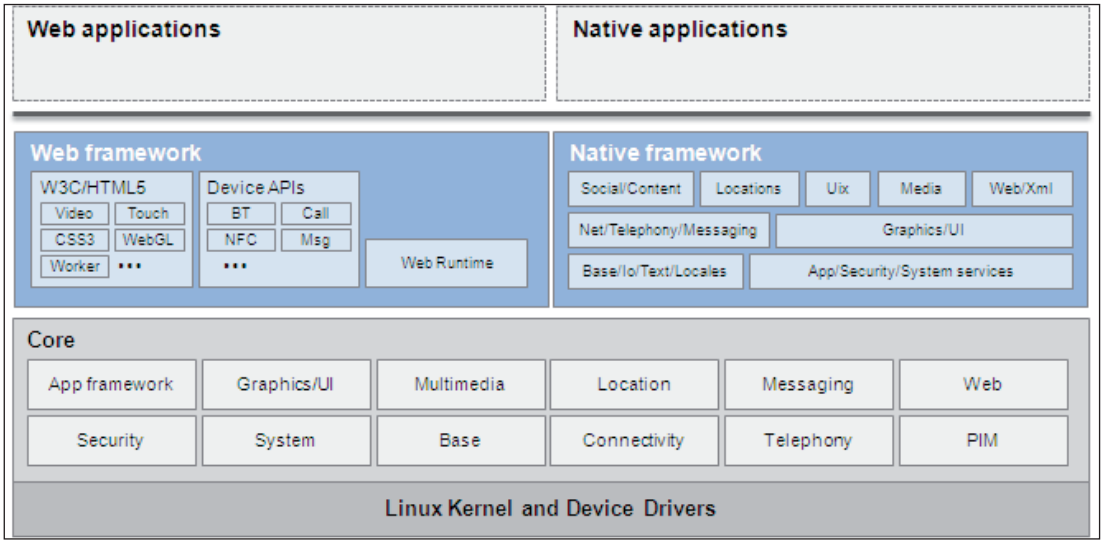

Figure 1: Tizen uses a layered environment built upon a foundation of the Linux kernel [2].

- System consists of system and device management features, including:
  - Interfaces for accessing devices, such as sensors, display, or the vibration device.
  - Power management, such as LCD display backlight dimming/switching off and application processor sleep.
  - Monitoring devices and handling events, such as USB, MMC, charger and ear jack events.
  - System upgrade.
  - Mobile device management.
- Telephony consists of cellular functionalities communicating with the modem:
  - Managing call-related and non-call-related information and services for UMTS and CDMA.
  - Managing packet service and network status information for UMTS and CDMA.
  - Managing SMS-related services for UMTS and CDMA.
  - Managing SIM files, phone book and security.
  - Managing SIM Application Toolkit services for UMTS.

Currently, the attack surface of the modules described above consists of bugs resulting from the consolidation of the previous platforms into *Tizen*, as well as introduction of new features into *Tizen*.

2. TIZEN PACKAGE STRUCTURES

A package is a container of executable content or apps that a system can install or uninstall. Each package has a unique identifier, called the 'PackageId', and each app within the package has a unique global 'AppId' as well as an 'AppName'. All apps in a package share resources as well as the privileges defined at the package level.

The Tizen Framework supports three types of packages: web application, native apps and hybrid apps (a mixture of native and web apps). *Tizen* also supports the installation of RPM packages.

Web application packaging is based on the W3C widget packaging specification. A web application package must conform to the following conventions:

- File format: ZIP archive file format
- File extension: .wgt (for example, sample.wgt)
- MIME type: application/widget.

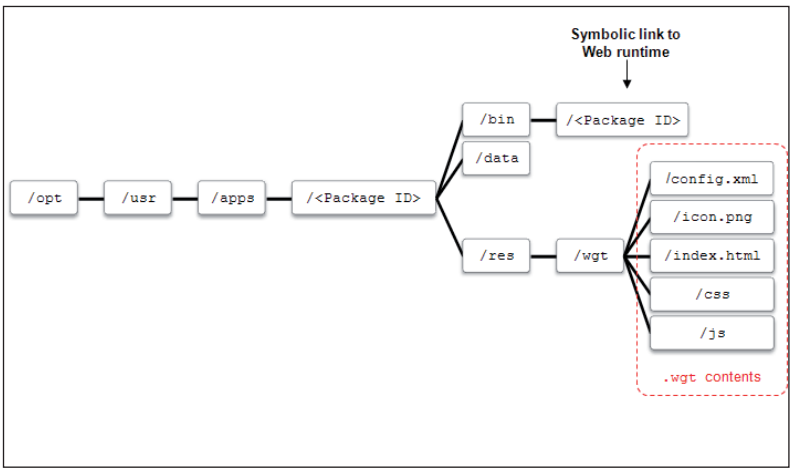

Figure 2: Web application packaging [5].

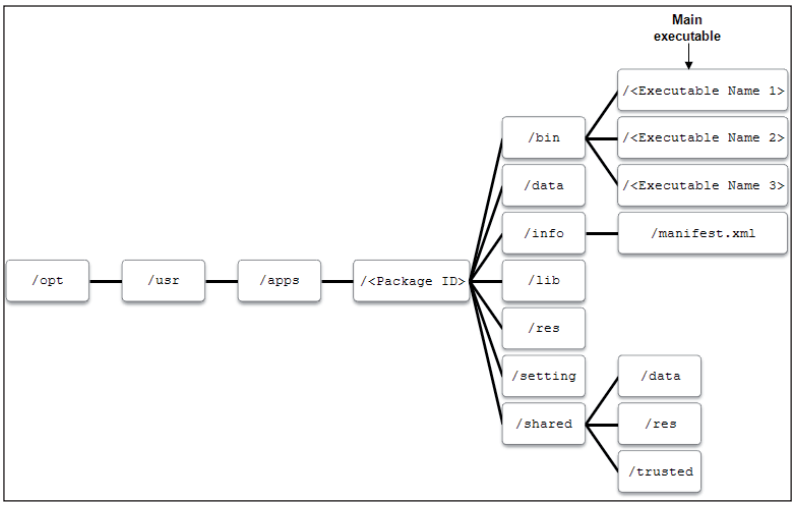

Figure 3: Package content [5].

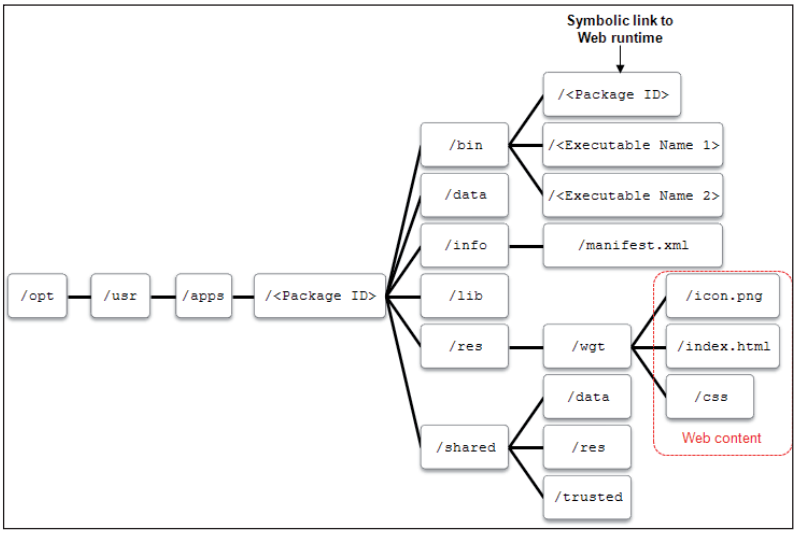

Figure 4: Web content [5].

A hybrid/native application package must conform to the following conventions:

- File format: ZIP archive file format
- File extension: .tpk (for example, sample.tpk)
- MIME type: application/x-tizen.package-archive

The root of the *Tizen* native package is the path of the ZIP archive, which contains reserved folders.

3. APPLICATION SECURITY/ACCESS CONTROL OVERVIEW

At the core of *Tizen*'s security enforcement is privilege control and application signing, which includes process isolation and mandatory access control [6].

The key principles are as follows:

- Applications will run under a non-root user ID.
- Daemons also run as a non-root user.
- An application is only allowed to read and write files in its home directory and shared media directory (/opt/usr/media).

Tizen operates Simplified Mandatory Access Control Kernel (Smack)-based access control and process isolation. The key principles of this are as follows:

- All applications are sandboxed by Smack.
- All Smack features from *Linux* kernel version 3.5 or later are respected.
- Applications run with Smack labels that are different from the predefined ones.

Tizen operates a secure execution environment:

- Native applications are launched by the application framework.
- Web applications are launched by the web runtime.
- There are no set-user-ID binaries in the device.

3.1 App/package signing

Similar to platforms such as *Blackberry* and *Android*, *Tizen* applications must be signed in order to execute. What is unique about the *Tizen* security model is the fact that application packages must have two signatures: an author signature and a distributor signature.

Author signature: Authors can digitally sign a package as a mechanism to ensure authorship and integrity. The author certificate, which is used for signing an author signature, must be registered with the Tizen Developer Certificate Authority.

Distributor signatures: These are generated by an application distributor, such as the *Tizen Store*, or a carrier, such as *NTT*, to confirm that the publisher has distributed the application package. The *Tizen* distributor signature also determines the privilege levels granted for a particular application.

RPM packages for *Tizen* are treated differently. Even though *Tizen* will prevent corrupted RPM packages from being installed, it will allow the installation of RPM packages that are unsigned – thus exposing users to risk from untrusted and malicious content [7]. This situation may change with future updates.

When creating/debugging applications locally, an author certificate is the only certificate that is required and needs to be created prior to an application being deployed on a emulator or a test device. A distributor signature is optional. Unlike *Android*, there is no generic debug certificate.

3.2 API-level access control

Tizen provides API-level access control for sensitive operations to protect user privacy and ensure system stability. Therefore, applications that use sensitive APIs must declare the required privileges.

The *Tizen* web API configuration document (config.xml) uses the following syntax:

```
<feature name="http://tizen.org/feature/network.nfc"/>
<tizen:privilege name="http://tizen.org/privilege/
application.launch"/>
```

The *Tizen* native API configuration document (manifest.xml) uses syntax as shown below:

```
<Requirements>
        <Feature Name="http://tizen.org/feature/
camera">true</Feature>
</Requirements>
<Privileges>
        <Privilege>http://tizen.org/privilege/
notification</Privilege>
</Privileges>
```

Once an application invokes a privileged API, *Tizen* checks whether the privilege is present in the config.xml or manifest.xml file. If the privilege is not present in either file, the system prohibits the execution of the application.

There are three levels of privilege:

- Public privileges are open to all *Tizen* application developers as a default.
- Partner privileges can only be invoked by developers that are registered as partners on the *Tizen Store*.

A developer must be fully identified and permitted by the partner policy of the *Tizen Store* to use both public- and partner-level privileges.

- Platform privileges are used in system APIs for managing the *Tizen* platform. These privileges are open only to a specific set of *Tizen* application developers.

Table 1 lists the official *Tizen* privilege names, mapped to the privilege levels described above.

3.3 Post install privacy control

An additional measure introduced in the *Tizen* framework allows users to install apps while denying some of the apps' attempts to collect the user's data – a solution to the

http://tizen.org/privilege/appmanager.certificate	Partner
http://tizen.org/privilege/appmanager.kill	Partner
http://tizen.org/privilege/datacontrol.consumer	Partner
http://tizen.org/privilege/secureelement	Partner
http://tizen.org/privilege/systemmanager	Partner
http://tizen.org/privilege/bluetoothmanager	Platform
http://tizen.org/privilege/bookmark.read	Platform
http://tizen.org/privilege/bookmark.write	Platform
http://tizen.org/privilege/lockmanager	Platform
http://tizen.org/privilege/packagemanager.install	Platform
http://tizen.org/privilege/packagemanager.setting	Platform
http://tizen.org/privilege/settingmanager.read	Platform
http://tizen.org/privilege/settingmanager.write	Platform
http://tizen.org/privilege/alarm	Public
http://tizen.org/privilege/application.launch	Public
http://tizen.org/privilege/bluetooth.admin	Public
http://tizen.org/privilege/bluetooth.gap	Public
http://tizen.org/privilege/bluetooth.spp	Public
http://tizen.org/privilege/calendar.read	Public
http://tizen.org/privilege/calendar.write	Public
http://tizen.org/privilege/callhistory.read	Public
http://tizen.org/privilege/callhistory.write	Public
http://tizen.org/privilege/contact.read	Public
http://tizen.org/privilege/contact.write	Public
http://tizen.org/privilege/content.read	Public
http://tizen.org/privilege/content.write	Public
http://tizen.org/privilege/datasync	Public
http://tizen.org/privilege/download	Public
http://tizen.org/privilege/filesystem.read	Public
http://tizen.org/privilege/filesystem.write	Public
http://tizen.org/privilege/messageport	Public
http://tizen.org/privilege/messaging.read	Public
http://tizen.org/privilege/messaging.write	Public
http://tizen.org/privilege/networkbearerselection	Public
http://tizen.org/privilege/nfc.admin	Public
http://tizen.org/privilege/nfc.common	Public
http://tizen.org/privilege/nfc.p2p	Public
http://tizen.org/privilege/nfc.tag	Public
http://tizen.org/privilege/notification	Public

Table 1: Official Tizen privilege names mapped to privilege levels.

fundamental problem with the commonly used security model of installing an app with an all-or-nothing approach.

Very similar to the feature that was dubbed 'AppOps' [8] in *Android 4.3*, *Tizen* allows post privacy controls, denying apps the ability access user data based on user-defined rules.

The rules set by the privacy policy apply to all apps in a package and cannot be defined on a per-app basis.

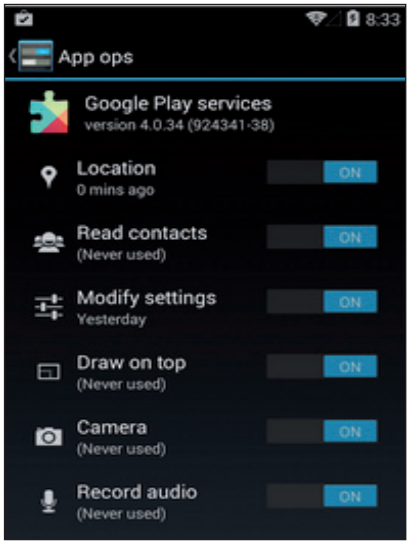

Figure 5: App ops [8].

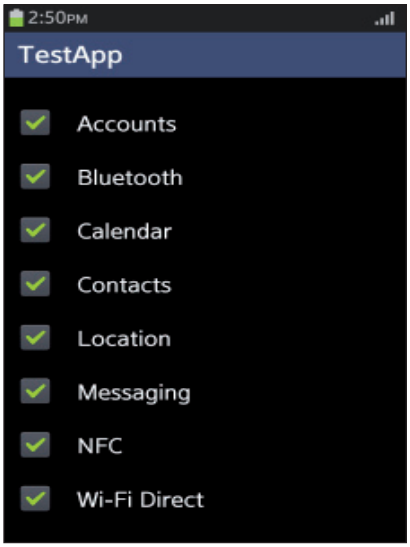

Figure 6: Post privacy controls.

3.4 Smack

Smack is a *Linux* security module (LSM) that determines how processes interact and behave; a kernel-based module implementing mandatory access control [9].

In *Tizen*, every application has its own Smack label. This is necessary to achieve application isolation (native/web). The Smack label, which is unique to the application, can be used to identify the application as well as to provide access controls.

The key concept behind Smack policy enforcement is that a subject can only access an object if their labels match or if there is an explicit permission granting access to the requested resource. A subject is any active entity. An object is any passive entity, examples of which include files, directories, message queues, and in the case of signals, other processes.

The rule format is:

```
[SubjectLabel] [ObjectLabel] [access(rwxa)]
```

Even though simplicity is a key strategy in the adoption of Smack as a security policy, the complication that arises is that the count of labels and rules tends to become large, increasing as the number of applications that are installed grows. This large number of rules can be difficult to administrate, not to mention hard to debug.

The role of Smack in *Tizen* is to ensure that malicious code is stopped dead in its tracks without the proper permission levels and security checks. *Tizen 3.0* will introduce a new feature, known as 'Cynara'. Cynara uses the application privileges assigned by the installer and the Smack label of the requesting application to determine if privilege use is appropriate. These data are presumed available by either mediators or services.

4. INTER-PROCESS COMMUNICATIONS

A growing concern regarding threats targeting smartphones is the interactions between separate applications – not just artifacts of single components – in order to carry out malicious activities [10].

In *Android*, IPC is largely a function of Binder. Binder is a kernel device driver that uses *Linux*'s shared memory feature to achieve efficient, secure communications. *Firefox OS* tries to isolate apps completely from direct communication – the only form of communication between apps is indirect. *Tizen*, on the other hand, makes extensive use of several protocols for interactions despite the fact that D-bus remains the officially recommended protocol of choice. Most of the OSS modules, such as ConnMan, use D-Bus for IPC, but other middleware – notably modules inherited from the SLP (*Samsung Linux Project*) – use custom forms of IPC, thus there is no definitive IPC protocol in *Tizen*. It is also worth mentioning that *Tizen 3.0* will see the introduction of k-dbus, which will attempt to consolidate IPC across the platform.

5. WEB RUNTIME (WRT)

Similar to the web framework used in *Firefox OS* called 'Gecko', the *Tizen* WRT supports the management of web apps (installation/uninstallation) and the execution of *Tizen* WebApps as well as W3C APIs and non-W3C APIs [11].

- The core is based on the WebKit framework
- Each widget is executed in its own separate processes space
- The application sandbox operates via Smack.

6. CONTENT SECURITY FRAMEWORK (CSF)

The CSF is a set of APIs/hooks that can be used to create security-related services. Developed by *McAfee*, these hooks were designed to be utilized by third-party security vendors to help harden the system. There are two types of scan engines that represent the ideal use case for the CSF: scan engines for data/content and site engines for URLs. Scan engines inspect content and are designed to use malware-matching patterns, as is typical of PC virus-scanning programs today. Site engines use a reputation system, in which the vendor categorizes URLs and creates block list policies by category (e.g. gambling, pornography, spyware, etc.) [12].

7. TIZEN STORE APP VALIDATION PROCESS

To ensure that any malware entering the *Tizen* ecosystem is immediately identified, applications submitted to the *Tizen Store* will be reviewed via a combination of static and dynamic analysis before they are published for public consumption.

The review process will subject the apps to a rigorous code review process: verifying their authenticity and integrity, using steps such as ensuring that requested permissions are used for the purposes stated, verifying that the use of implicit permissions is appropriate, and validating that any interfaces between privileged app content and unprivileged external content have the appropriate mitigations to prevent elevation of privilege attacks.

Similar to the Bouncer verification system used by *Android* [13], the *Tizen* app validation system uses an array of emulators to track app behaviour while attempting to simulate app usage in order to trigger/spot malicious actions.

Even though the review process can be viewed as a mix of the review processes used by the *Android* Security Team and the *Apple* App Review, the differentiator here is that the *Tizen* app validation process promises to deliver an assessment on an app within three days of the first submission of the app [14].

The three-day investigation period opens up the possibility of attacks where victims could be targeted post release using techniques such as ad networks with low or no security screening processes [15].

CONCLUSIONS

It would not be an exaggeration to say that security is a dominant factor when it comes to the architecture of *Tizen*. The concern for security is not only reflected in the design of the platform all through the *Tizen* ecosystem, as exhibited by the app validation process. But in the current threat landscape, where threats against mobile devices have peaked, discoveries of issues such as the first public vulnerability related specifically to a *Tizen* device [16] and risks of XSS attacks [17] targeting support for web apps means that *Tizen* will constantly have to keep evolving, not only at the platform level but also at the ecosystem level, to ensure that the available attack surface is always kept to a minimum. Even though the security measures taken in *Tizen* reflect best practices and lessons learned from the threat landscape, there are still improvements that can be made.

REFERENCES

[1] Vérez, A.; Hugues, G. Security Model of Firefox OS. 2013. http://anthony-verez.fr/docs/ffos_paper.pdf.

[2] Saxena, S. Tizen Architecture Overview. January 2014. https://www.tizen.org/sites/default/files/tizen-architecture-linuxcollab.pdf.

[3] Kim, B. W. Overview of the Tizen Native Application Framework. 2013. https://cdn.download.tizen.org/misc/media/conference2013/slides/TDC2013Overview_of_the_Tizen_Native_Application_Framework.pdf.

[4] Porting Tizen. https://source.tizen.org/sites/default/files/tizen_porting_guide_2.0_alpha.pdf.

[5] Jaygarl, H.J. et al. Professional Tizen Application Development. John Wiley & Sons, 2014.

[6] Im, B.; Ware, R. Tizen Security Overview. Tizen Developer Conference Session Slides. http://download.tizen.org/misc/media/conference2012/tuesday/ballroom-c/2012-05-08-1600-1640 tizen_security_framework_overview.pdf.

[7] https://bugs.tizen.org/jira/browse/TC-866.

[8] Eckersley, P. Awesome Privacy Tools in Android 4.3+. December 2013. https://www.eff.org/deeplinks/2013/11/awesome-privacy-features-android-43.

[9] Description from the Linux source tree. http://schaufler-ca.com/description_from_the_linux_source_tree.

[10] Marforio, C. et al. Analysis of the communication between colluding applications on modern smartphones. Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012.K. Elissa

[11] Aciicmez, O.; Blaich, A. Understanding the Permission and Access Control Model for Tizen Application Sandboxing. http://download.tizen.org/misc/media/conference2012/wednesday/seacliff/2012-05-09-0945-1025-understanding_the_permission_and_access_control_model_for_tizen_application_sandboxing.pdf.

[12] Pillutla, S. Content Security Framework. May2013. http://cdn.download.tizen.org/misc/media/conference2013/slides/TDC2013-Content_Security_Framework.pdf.

[13] Oberheide, J.; Miller, C. Dissecting the Android bouncer. SummerCon2012, New York (2012).

[14] Tizen App Validation Guide. https://developer.tizen.org/sites/default/files/documentation/tizen_validation_guide_ver_1.4_140529.pdf.

[15] Rogers, M. The Bearer of BadNews. Lookout. https://blog.lookout.com/blog/2013/04/19/the-bearer-of-badnews-malware-google-play/.

[16] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-6459.

[17] Guilherme, Í. A Simple SMS Reader. Iscaros. http://giscaro.wordpress.com/2012/04/10/a-simple-sms-reader/.